Modeling and Analysis of Reliable Peer-to-Peer Storage Systems

Julian Monteiro

UNIVERSITÉ DE NICE-SOPHIA ANTIPOLIS

ÉCOLE DOCTORALE STIC
SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION

T H È S E

pour obtenir le titre de
Docteur en Sciences
de l'Université de Nice - Sophia Antipolis
Mention : **INFORMATIQUE**

Présentée et soutenue par
Julian Geraldes MONTEIRO

Modeling and Analysis of Reliable Peer-to-Peer Storage Systems

Thèse dirigée par : **Stéphane PÉRENNES** et **Olivier DALLE**
préparée au sein du projet MASCOTTE - INRIA / I3S (CNRS/UNS)
soutenue le 16 novembre 2010

Jury :

President :	Alain JEAN-MARIE	Directeur de Recherche INRIA - LIRMM (Maestro)
Rapporteurs :	Virgílio ALMEIDA	Professeur - U. Federal de Minas Gerais, Brésil
	Fabien MATHIEU	Ingénieur - Orange Labs
	Pierre SENS	Professeur - U. Pierre Marie Curie, Paris VI (Regal)
Directeurs :	Stéphane PÉRENNES	Directeur de Recherche - CNRS (Mascotte)
	Olivier DALLE	Maître de Conférences - U. Nice Sophia (Mascotte)
Examinateurs :	Frédéric GIROIRE	Chargé de Recherche - CNRS (Mascotte)
	Alfredo GOLDMAN	Professeur - U. de São Paulo, Brésil

À minha querida Mayara e à minha família.
À ma dulcinée Mayara et à ma famille.
To my beloved Mayara and my family.

Acknowledgments

Firstly, I would like to thank my brave advisors Olivier Dalle, Stéphane Pérennes and Frédéric Giroire who accepted me as a student and spent a lot of time with me. Their complementary skills worked greatly! I've learned much from them. I would not have started my PhD without Olivier and his generosity. Stéphane and his incredible knowledge and personality gave me willingness to learn. Fredo, who really supported me every day (and night!) and put me to work with his enormous patience and kindness. I carry with me their friendship and profissionalism.

I thank the reviewers of this thesis, Virgilio Almeida, Fabien Mathieu and Pierre Sens for their invaluable feedback. Also, Alain Jean-Marie and Alfredo Goldman to be part of the jury. Thanks for the UbiStorage guys, Sébastian Choplin and Hung-Cuong Le.

I greatly appreciate the support of Afonso Ferreira and Alfredo Goldman, who brought me to this amazing group of people that is the Mascotte team. Led by the boss Jean-Claude Bermond, who has an incredible energy of life and keeps all of us together. I will not forget that he made me overcome the challenge of finishing a marathon (literally!).

Je remercie à tout le monde de la grande famille Mascotte qui m'ont donné la joie de passer les journées ensemble et m'ont laissé de souvenirs pour toute la vie: Aurélien, Christelle, David, Dorian, Fabrice, Florian, Gianpiero, Hervé, Ignasi, Issam, Joanna, Juan-Carlos, Judicael, Luc, Michel, Nathan, Nicolas, Remig, Sandeep et Pato. Un merci tout spécial à la gentillesse de nos assistants Patricia et Sandra. Et bien sûr, Fred Majus à 3 rose de vents qui nous a accueilli plusieurs fois avec ses barbecues animé par les caipirinhas de Napi.

Ahhh claro, um especial agradecimento aos sinceros amigos brasileiros! Napinho (Jean-Marc), Jujulio, Dodo, Joana, Gugu, Cristiana, Léo e Ronan, que compartilharam comigo esse período de grande experiência e que tornaram tudo muito mais fácil e divertido.

Agradeço à minha família e aos meus caros amigos lá de longe que não me deixaram em nenhum momento.

Admiro minha querida Mayara, que foi muito corajosa em me acompanhar! Esteve sempre presente e me deu seu incondicional apoio durante esses três anos longe da nossa terra natal. Agradeço ainda mais pela família que estamos formando juntos. Uma grande alegria.

Antibes
November 30, 2010

Abstract

Large scale peer-to-peer systems are foreseen as a way to provide highly reliable data storage at low cost. To ensure high durability and high resilience over a long period of time the system must add redundancy to the original data. It is well-known that erasure coding is a space efficient solution to obtain a high degree of fault-tolerance by distributing encoded fragments into different peers of the network. Therefore, a repair mechanism needs to cope with the dynamic and unreliable behavior of peers by continuously reconstructing the missing redundancy. Consequently, the system depends on many parameters that need to be well tuned, such as the redundancy factor, the placement policies, and the frequency of data repair. These parameters impact the amount of resources, such as the bandwidth usage and the storage space overhead that are required to achieve a desired level of reliability, i.e., probability of losing data.

This thesis aims at providing tools to analyze and predict the performance of general large scale data storage systems. We use these tools to analyze the impact of different choices of system design on different performance metrics. For instance, the bandwidth consumption, the storage space overhead, and the probability of data loss should be as small as possible. Different techniques are studied and applied. First, we describe a simple Markov chain model that harnesses the dynamics of a storage system under the effects of peer failures and of data repair. Then we provide closed-form formulas that give good approximations of the model. These formulas allow us to understand the interactions between the system parameters. Indeed, a lazy repair mechanism is studied and we describe how to tune the system parameters to obtain an efficient utilization of bandwidth. We confirm by comparing to simulations that this model gives correct approximations of the system average behavior, but does not capture its variations over time. We then propose a new stochastic model based on a fluid approximation that indeed captures the deviations around the mean behavior. These variations are most of the time neglected by previous works, despite being very important to correctly allocate the system resources.

We additionally study several other aspects of a distributed storage system: we propose queuing models to calculate the repair time distribution under limited bandwidth scenarios; we discuss the trade-offs of a Hybrid coding (mixing erasure codes and replication); and finally we study the impact of different ways to distribute data fragments among peers, i.e., placement strategies.

Keywords: Data storage, peer-to-peer, performance analysis, Markov chains, fluid models, queue models, simulations.

Resumé

Les systèmes pair-à-pair à grande échelle ont été proposés comme un moyen fiable d'assurer un stockage de données à faible coût. Pour assurer la pérennité des données sur une période très longue, ces systèmes codent les données des utilisateurs comme un ensemble de fragments redondants qui sont distribués entre différents pairs du réseau. Un mécanisme de réparation est nécessaire pour faire face au comportement dynamique et non fiable des pairs. Ce mécanisme reconstruit en permanence les fragments de redondance manquants. Le système dépend de nombreux paramètres de configuration qui doivent être bien réglés, comme le facteur de redondance, sa politique de placement et la fréquence de réparation des données. Ces paramètres affectent la quantité de ressources, telles que la bande passante et l'espace de stockage, nécessaires pour obtenir un niveau souhaité de fiabilité, c'est-à-dire, une certaine probabilité de perdre des données.

Cette thèse vise à fournir des outils permettant d'analyser et de prédire la performance de systèmes de stockage de données à grande échelle en général. Nous avons utilisé ces outils pour analyser l'impact de différents choix de conception du système sur différentes mesures de performance. Par exemple, la consommation de bande passante, l'espace de stockage et la probabilité de perdre des données, doivent être aussi faibles que possible. Différentes techniques sont étudiées et appliquées.

Tout d'abord, nous décrivons un modèle simple par chaîne de Markov qui exploit la dynamique d'un système de stockage sous l'effet de défaillance des pairs et de réparation de données. Puis nous établissons des formules mathématiques closes qui donnent de bonnes approximations du modèle. Ces formules nous permettent de comprendre les interactions entre les paramètres du système. En effet, un mécanisme de réparation paresseux (*lazy repair*) est étudié et nous décrivons comment régler les paramètres du système pour obtenir une utilisation efficace de la bande passante. Nous confirmons en comparant à des simulations que ce modèle donne des approximations correctes du comportement moyen du système, mais ne parvient pas à capturer ses importantes variations au fil du temps. Nous proposons ensuite un nouveau modèle stochastique basé sur une approximation fluide pour saisir les écarts par rapport au comportement moyen. Ces variations qui sont généralement négligées par les travaux antérieurs, sont très importants pour faire une bonne estimation des ressources nécessaires au système.

De plus, nous étudions plusieurs autres aspects d'un systéme de stockage distribué: nous utilisons un modèle de files d'attente pour calculer le temps de réparation pour un système avec bande passante limitée; nous étudions un système de codage hybride: en mixant les codes d'éffacement avec la simple réplication des données; enfin, nous étudions l'impact des différentes façons de distribuer des fragments de données entre les pairs, i.e., les stratégies des placements.

Mots clés: Stockage de données, pair-à-pair, analyse de performance, chaînes de Markov, modèle fluide, modèle de file, simulations.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

"Memory is the mother of all wisdom."
— Aeschylus, 525 – 456 B.C

1.1 Motivation

The amount of digital information produced by each individual every year is enormous and tends to grow continuously. For instance, the dissemination of digital media, digital photos, digital videos, electronic mail, personal blogs, etc. promotes the production of digital data on a daily basis by every computer user. These data, or at least part of it, must be stored reliably.

In the reports conducted since 2007 by the International Data Corporation (IDC) [53, 54], their authors state that the amount of digital data per person on the planet was already 75 gigabytes in 2009. They estimate that *"Between 2009 and 2020, the information in the Digital Universe will grow by a factor of 44; the number of files to be managed will grow by a factor of 67, and storage capacity will grow by a factor of 30"*. In the same report, they claim that if people want to store every gigabyte of digital content created, there is already a gap of 35% in the space available. This gap is likely to increase in the next years, as depicted in Figure 1.1. IDC data shows that 25% of this information is unique. Even though much of this content is not that important, that is, not requiring a high degree of protection, the other fraction of these data indeed needs a high level of protection and redundancy in order to ensure its availability over a long period of time. This enormous growth incites the research on *reliable and long-term data storage solutions that consume a low amount of resources.*

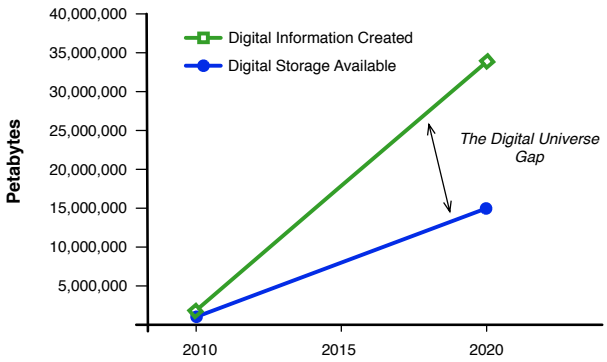


Figure 1.1: Growth estimation of produced data versus available storage [54].

Traditional solutions to make a primary backup of data are magnetic tapes, optical media or external backup-drives. Although they are commonly used, these solutions are not renowned to be durable. Moreover, their operation requires a manual maintenance procedure to keep the physical medium protected and up-to-date. Nowadays, another options to maintain a safe backup are the on-line services, data centers and high-end NAS (Network Attached Storage) appliances. These approaches can be highly reliable, but also tend to be very expensive. Moreover, if durability is the first concern, then relying on only one external backup provider can be risky. The data also need to be replicated somewhere else, for the sake of redundancy and geographic dispersion (i.e., to keep data safe from natural catastrophes).

Recently, networked devices and bandwidth have become cheaper and widely available, allowing new forms of data storage on distributed architectures. It seems that peer-to-peer storage is a natural evolution for data backup. The popularization of peer-to-peer overlay networks by many applications incites the use of this technology to develop a large-scale platform that provides reliable and safe data storage. These highly distributed solutions are foreseen as an interesting alternative to the traditional data centers and in-house backup solutions. The advantages of a peer-to-peer network can be many: one can achieve a highly reliable system at low cost, that by nature distributes the data among peers that are in different geographical regions of the globe. Moreover, this architecture has a high potential to be scalable.

Challenges

In the past few years many peer-to-peer storage systems have been proposed as reliable and cost effective solutions. But their deployment in practice is still a challenging endeavor. Distributed systems are prone to peer disconnections, peer unavailability, disk failures and malicious behaviors. To ensure high durability and high resilience over a long period of time the system must distribute redundant data among different peers. A self-repair mechanism needs to handle the dynamic and unreliable behavior of peers by continuously reconstructing the missing redundancy. This process consumes bandwidth, which is often the scarcest resource of a peer-to-peer network, at least when compared to storage space and computing capacity. If the peers' bandwidth is very limited, the reconstruction of the lost redundancy takes long time to finish, which substantially impacts the expected data lifetime. Furthermore, the implementation of such a system raises plenty of questions: How much redundancy should be added? How much bandwidth is used by the repairing process? When to trigger the repairing? What is the probability of data loss? Where to place the redundant replicas? How the constraints of bandwidth affect the data lifetime?

Our Goal

This thesis aims at providing tools to analyze and to predict the performance of a general large-scale data storage system. These techniques range from: formal analysis, using Markov chains and fluid models; simulations, using a custom cycle based simulator;

and experimentation, using the Grid5000 platform. We use these tools to analyze the impact of different choices of system design on a set of performance metrics of interest. In particular, our goal is to provide practical contributions that can enhance the utilization of a peer-to-peer storage system. For instance, the bandwidth consumption, the storage space overhead, and the probability of data loss should be as small as possible. Moreover, we give special attention to harness the variations around the average system performance.

1.2 Background on Peer-to-Peer Storage Systems

The concepts of the studied peer-to-peer storage systems are presented in this section. First, we give a brief description of its general characteristics, followed by some background on peer-to-peer networks in Section 1.2.2, which is the main substrate used by the large-scale distributed storage systems. Then, in Section 1.2.3 we give an overview of the basic techniques to introduce data redundancy (replication and erasure codes), followed by the details of the repairing service in Section 1.2.4. Finally, in Section 1.2.5 we present a compilation of distributed storage systems proposed in the literature.

1.2.1 Description

Data. The purpose of a storage system is to keep data persistent and accessible. The data stored in the system can be of any nature: personal files, company documents, public content, etc. Hereafter and throughout this thesis, we generalize that notion and denote the user data as *data blocks*, or simply *blocks* for short. However, the type and format of these data can be of any kind: files, blobs, raw bits, etc. Information security is also considered, we assume that the original data is encrypted by the user before the archival process. Nonetheless, other kind of security concerns, as privacy, malicious peers, etc. are not addressed in this thesis.

Peers. We denote as *peer* (or *node*) the entity of the network that participates of the storage system. These peers can be laptops, desktop computers, enterprise servers, brick storage devices or any computer that stays turned on for a considerable amount of time. As opposed to peers of *file-sharing* applications, which are often opportunists: they connect to network only when they want some content, and after they never come back. Hence, a distributed storage system relies on peers that are regularly connected to the network (e.g., during working hours) and share some of their resources (e.g., bandwidth and storage space). Usually, it is a assumed that a peer-to-peer overlay network is used as main infrastructure. Although, our methods and analysis can be applied to any kind of distributed storage system.

Redundancy. Distributed (or peer-to-peer) storage systems are prone to disk failures (or peers that permanently leave the network). Hence, redundancy data need to be

introduced and distributed among peers to ensure high availability, durability and resilience. The addition of redundant data can be done by trivial *Replication* [23, 108], in which full-copies of the data are sent to different peers in the system; or be based on the more complex and space efficient *erasure codes* [97, 79], such as Reed Solomon [101], Digital Fountains [26], Regenerating Codes [42] or Hierarchical Codes [46].

Repair. Even with a high degree of redundancy, to achieve long term fault tolerance the storage system needs to have a monitoring and a repairing service to cope with peer failures. These services continuously monitor the redundancy level of data blocks and decide if a repairing process of missing redundancy (namely *reconstruction*) needs to be done.

Usage. Two kinds of system usage are easily distinguished: the first group is a typical distributed *file-system usage* (e.g, Farsite [23], CFS [35], Pastis [25]) that supports continuous read and write operations; the second group is a *versioned backup* or *data archival system* (e.g., Intermemory [56], Glacier [59]), in which the stored data is immutable. The write operations are solely the introduction of new blocks. Read operations are very occasional. Our models and analysis are valid for both kind of usages.

Metrics. We focus our analysis on the bandwidth required by the self-repairing mechanism to achieve a certain degree of reliability (i.e., probability of data loss) and occuping a certain amount of storage space. The resources needed for the user accesses or writes to data should be accounted separately. In fact, one can also be interested in another metric: the performance of access time. That is, the latency of read and write operations and the throughput of transfers.

1.2.2 Peer-to-Peer Networks

Although there is no general consensus on the definition of Peer-to-Peer (P2P) networks, researchers agree that "decentralized", "node equality", and "resource sharing" are the main keywords used in their definitions [78]. The linguistic construction "peer-to-peer" already suggests that a P2P network is composed of network nodes (called peers) which are treated equally and communicate directly with each other, as opposed to the traditional client-server network, in which several clients connect to one server and no one connects to the clients. The resulting communication topology is hence different. In a P2P network the majority of peers (ideally all of them) make resources available to other peers. Resources are of various natures: computational time, storage space, services, bandwidth, etc. Thus, P2P networks are nowadays massively used by different kind of applications, for example, content distribution, file sharing, storage services, telephony, video streaming, etc. Within the following definitions, instead of storage applications, we give examples of content distribution and file sharing applications.

Classification

According to the survey by Theotokis et al. [115], the P2P hierarchy could be categorized in three big groups: decentralized, hybrid, and centralized.

- **Decentralized:** in a pure decentralized P2P network, all peers play the same role in the network and there is no central coordination. If they leave, the network stays alive (e.g., Gnutella v0.4 [64], Freenet [29], BitTorrent with distributed tracker [30]).

- **Hybrid:** in hybrid P2P networks (or partially centralized as defined in [115]), some of the peers assume a more important role. Usually, they have more knowledge about the network than others (e.g., Kazaa [74], super-peers in Gnutella v0.6 [67]), and act as local central indexes to provide local coordination.

- **Centralized:** in centralized P2P networks, its operation depends on the existence of one or more central servers (e.g. Napster, Seti@Home, BitTorrent tracker). These central servers are exogenous entities (they are not peers) that provide to peers the knowledge required to operate in the P2P network. Such servers do not share content of physical resources, they act as directories that provide addresses of peers that contain the requested resource.

In the P2P survey by Lua et al. [78], an other detailed classification is given. According to its organization, the P2P (or *overlay*) networks are divided into two categories:

- **Unstructured**: peers are distributed as a *random graph* in a flat or hierarchical manner and use flood, random walks or expanding-ring time-to-live to search for content. Examples of this organization are Freenet, Gnutella, KaZaa, BitTorrent with tracker. The searching mechanisms in unstructured networks have the advantages of simplicity and the intrinsic capabilities to query for unknown content (e.g., it is easy to implement a "search" for content primitive based on keywords). However, they have several issues regarding scalability and persistence.

- **Structured**: the topology of the overlay network determines the placement of the content. Distributed Hash Table (DHT) [112] is the most common structural architecture for novel P2P overlay networks, which provides a "store" and "lookup" service similar to the one of a traditional hash table. In this architecture a unique identification is assigned to every peer. Every data object also has an identification, namely *key*, chosen in the same identifier space of the peers. This structured graph enables efficient "lookup", "store" and "retrieval" operations of data objects using the given keys. The operation to lookup a given key is performed in $O(\log N)$ hops (where N is the total number of peers), which allows this architecture to be very scalable. But note that, in its simplest form, this class of architecture does not support complex queries based on the data content (as is possible when using unstructured architectures). The key to be searched needs to be known by other means.

DHT Implementations. In general, peer-to-peer storage systems use this structured overlay architecture as a base infrastructure. Thus, here follows some examples of some DHT implementations. Proposed by Stoica et al. in 2001, Chord [112, 75] is one of the first distributed hash lookup services, derived from the work by Plaxton [95]. In Chord the nodes are organized on a virtual ring, each node has a list of successor and predecessor nodes (logical neighbors) and a finger table that is used for routing in the ring space. The key lookups are performed by iterating over the peers' finger tables, which is achieved in $O(\log N)$ hops. Proposed by Rostron and Druschel, Pastry [107] is another well known key-based routing hash lookup substrate, which is very similar to Chord. The main difference is its identifier space that is not organized as ring, but based on the proximity of numerical identifiers. Several other scalable DHTs have been proposed, with different properties. For example, CAN [100], Kademlia [81], Koorde [65], Kelips [58] and Tapestry [132]. It is not our intention to give a detailed explanation of such protocols.

Churn and Failure Model

When developing a durable storage system, the first two performance metrics that come to mind are data availability and durability [28]. These two metrics are associated to the peers' dynamics, namely *churn*. The term churn represents the oscillations on the number of peers that participate in the network, which are mainly occasioned by peer joins, transient departures, and failures. A peer failure represents a permanent departure, which could be caused by a hardware crash or a network disconnection.

Different models to characterize churn (or availability) can be found in the literature. The simplest one is the use of a single variable that represents the *fraction of nodes that are available at any given time* [125, 44]. This approach does not give any insights about the distribution of node departures and arrivals, but allows us to estimate an average behavior.

Thus, more refined models are often found in the literature. Bhagwan et al. in [19], present the *host turnover*, i.e., the rate that new nodes arrive in the system and the rate that existing hosts leave permanently. These metrics are important for peer-to-peer systems that rely upon long-term host membership. It does not account for short time disconnections. Other studies, for example Rhea et al. in [103] and Stutzbach et al. in [113], characterize the churn with two metrics: the *session length* (the elapsed time between when a node joins the network until the next time it leaves); and the *lifetime* (the time between the node enters the network for the first time and leaves the network permanently). Hence, the sum of a node's session times divided by its lifetime is often called its *availability* [103, 19].

In the model by Rodrigues and Liskov [105] the peer can leave the system for short periods of time, as for example during restarts or power outages. If a peer stays disconnected for a time smaller than a given timeout τ (usually a few hours), the system does not react as a failure. Otherwise, the peer is considered to have failed permanently. In this thesis, we focus our discussions on this kind behavior as we argue that it represents

more accurately the behavior of distributed storage systems, where peers are supposed to be regularly connected to network.

Remark on availability scenario. Should be noted that in a P2P storage system, the peers are interested in participating in the network to gain storage space (as the business model of Wuala [129], for example, where on-line time can be "traded" for storage space). Many works analyze the performance of P2P storage in a highly dynamic scenario, with the use of some real world traces as a reference, e.g., Skype trace, Overlab, Planetlab, Gnutella [59, 41]. Unfortunately, to this date, there are no public available traces for P2P storage systems. Hence, we believe that the behavior of such a system is closer to the PlanetLab traces, where nodes are highly available (for a timeout of 24 hours, we obtain an expected peer lifetime of around 90 days [41]). In that case, departures are due to long maintenance or failures.

Failure Model. In our models a peer failure represents a disk crash or a peer that definitively leaves the system. In both cases, it is assumed that all the data on the peer's disk are lost. Following other works in the literature [98, 14, 73], these events occurs *independently* of one another, according to a memoryless process. For a given peer, the probability to fail at any given time step is $\alpha = 1/MTTF$, where $MTTF$ is the expected lifetime of peers (Mean Time To Failure). The probability for a peer to be alive after T time steps is $(1-\alpha)^T$. It is important to note that if, for a single disk, this is a rare event, a system with thousands of disks continuously experiences such failures [91].

However, other failure models, for example having a different rate for different disk age, can be modeled with more complex Markovian models.

1.2.3 Data Redundancy

In the following, we give an overview of the two classic methods to introduce redundancy to data: replication and erasure codes.

Simple Replication

When employing a replication scheme, the original data is copied as-is to k different nodes of the network. In that case, even if $k-1$ peers are not available or have failed, the original data can still be recovered. Figure 1.2 depicts the introduction of a file with replication factor $k = 3$. The repair process when a replica is lost is straightforward: it is sufficient that one of the remaining replicas sends a new copy to a different peer. We define the *stretch factor* (or *space-overhead*) as the ratio between the amount of space consumed over the original size of the data. The inverse is named *useful space*. In the simple replication the stretch factor is k, the useful space is $1/k$.

The replication scheme is the most natural method to add data redundancy. Every copy of the original data is equivalent, which makes the system development easier.

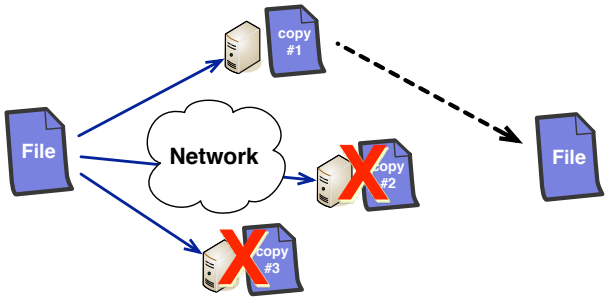


Figure 1.2: Example of System using simple replication, with $k = 3$ replicas. It tolerates $k - 1$ failures and consumes k times the original space.

However, to achieve high availability the space-overhead of replicated systems is very high.

Erasure Coding

Error Correcting Codes (ECCs) [60] are being employed for a long time to prevent loss of information in digital communications. These codes were developed to *detect* and *correct* bits of errors that occur during the transmission of a data stream. A class of ECC is the *erasure codes* [97], which is used when the system can distinguish in advance that one of the encoded fragments is missing or is corrupted. The erasure codes can be seen as a generalization of RAID (Redundant Array of Inexpensive Disks) parity schemes [88].

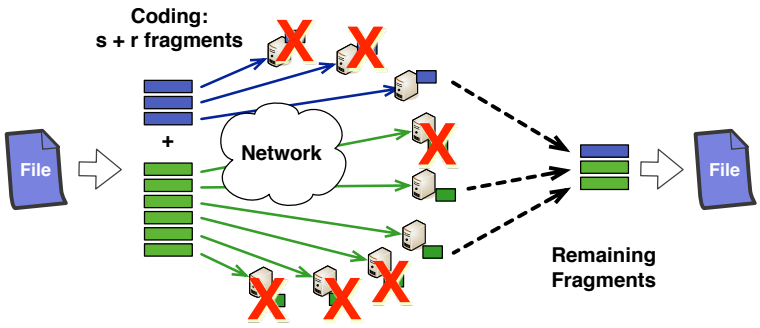


Figure 1.3: Example of system using erasure codes, with $s = 3$ and $r = 6$. It tolerates r failures and consumes $\frac{s+r}{s} = 3$ times the original space.

In general, a data block b of size l_b is cut into s equally sized initial *fragments* (or pieces) of size $l_f = l_b/s$. Then, r pieces of redundancy are added, in such a way that the initial data can be reconstructed from any subset of s pieces among the $s + r$. The stretch factor is then $(s + r)/s$. The *useful* space is $s/(s + r)$. Note that replication can be seen as a special case of erasure codes, when $s = 1$. Figure 1.3 depicts an example of erasure codes with $s = 3$ and $r = 6$. In that case, the system can tolerate up to 6 permanent failures. But note that the space-overhead in this example is only $(3 + 6)/3 = 3$, which

is the same overhead as the replication example above that tolerates only two failures (Figure 1.2)[1].

When the original data can be recovered from any combination of s fragments among $s + r$, we say that the corresponding erasure coding is a Maximum Distance Separable (MDS) code. The classic Reed-Solomon [101] is an example of such a code.

The process of repairing the missing redundancy fragments is more complex than the trivial replication. For example, to regenerate one missing fragment when using the Reed-Solomon codes, it is necessary to first gather s fragments from the network, then reconstruct the whole original data block, and then regenerate the missing fragment. This process is discussed more deeply in the next section.

Reed-Solomon Codes. The most used code in practice is indeed the Reed-Solomon. Hence, in the following we give a primer on how it works (as described in the tutorial by Plank [93]). The RS coding method splits a data block b of size l_b into s fragments of size l_b/s each. Let us define $\mathcal{D}$ as a vector containing the s initial fragments $d_1, \ldots, d_s$, and $\mathcal{C}$ as the vector to be generated containing the $s + r$ redundancy fragments $c_1, \ldots, c_{s+r}$, in such a way that the loss of any r fragments of $\mathcal{C}$ can be tolerated. The vector $\mathcal{C}$ is generated from an operator $\mathcal{F}$, which is an encoding matrix of size $(s + r) \times s$ with the following property: *any sub-matrix formed by deleting r rows of the matrix $\mathcal{F}$ is invertible*. This property guarantees that the original data can be recovered from any s rows. It is also interesting to have the first s rows as the identity matrix ($s \times s$ matrix). This property guarantees that the first s encoded fragments of $\mathcal{C}$ remains the same as the original $\mathcal{D}$.

Indeed, the matrix $\mathcal{F}$ can be easily derived from a Vandermonde matrix, $V_{i,j} = j^{i-1}$ [93], for example. Finally, to generate the redundancy fragments $\mathcal{C}$, it is sufficient to multiply $\mathcal{F}\mathcal{D} = \mathcal{C}$, as follows

$$\overbrace{\left.\begin{matrix}\\\\\\\\\\\\\\\\\end{matrix}\right\}}^{s}{\scriptstyle s}\left\{\begin{matrix}\\\\\\\\\\\\\\\\\end{matrix}\right.\left(\begin{matrix}1 & 0 & 0 & \ldots & 0\\ 0 & 1 & 0 & \ldots & 0\\ \vdots & \vdots & \vdots & & \vdots\\ 0 & 0 & 0 & \ldots & 1\\ 1 & 1 & 1 & \ldots & 1\\ 1 & 2 & 3 & \ldots & s\\ 1 & 4 & 9 & \ldots & s^2\\ \vdots & \vdots & \vdots & & \vdots\\ 1 & 2^{r-1} & 3^{r-1} & \ldots & s^{r-1}\end{matrix}\right)\left(\begin{matrix}d_1\\ d_2\\ \vdots\\ d_s\end{matrix}\right) = \left(\begin{matrix}c_1 & = & d_1\\ c_2 & = & d_2\\ \vdots & & \vdots\\ c_s & = & d_s\\ c_{s+1} & & \\ c_{s+2} & & \\ \vdots & & \\ c_{s+r} & & \end{matrix}\right) \quad (1.1)$$

In this case the first s elements of the encoded fragments $\mathcal{C}$ are equal to the original elements. The others, $c_{s+1}, \ldots c_{s+r}$, are linear combinations of the rows of $\mathcal{D}$.
Recovering from failures: from a vector $\mathcal{C}' \subseteq \mathcal{C}$ of size at least s (that is, at most r elements missing), we can regenerate the original data vector $\mathcal{D}$. The indices of these remaining

[1]Some traditional works use the notation (n,m) or (n,k), to express respectively, the total number of encoded fragments and the original number fragments. In our notation, it is equivalent to note (s+r, s).

elements of $\mathcal{C}$ should be known (i.e., the erasure property). Let $\mathcal{F}'$ be a matrix whose rows are a subset of $\mathcal{F}$ with the same indices as $\mathcal{C}'$. By construction we know that $\mathcal{F}'\mathcal{D} = \mathcal{C}'$ holds (we know that $\mathcal{F}'$ is invertible). Then, the original data vector can be calculated trivially with $\mathcal{D} = \mathcal{C}'\mathcal{F}'^{-1}$. One can solve that system using Gaussian Elimination.

Note that all the Reed-Solomon calculations are done using the algebra in the *Galois Fields* of size 2^w elements (denoted $GF(2^w)$). Usually w is chosen as 8 or 16, to match the machine byte length. In that case, the values of s and r are subject to the constraints $s + r < 256$ or $s + r < 65536$, respectively. It should be noted that the costs of encoding and decoding are quadratic on the number of fragments, which can make the choice of very large values of s and r infeasible in today's computers [94]. The ZFec library is an open-source implementation of Reed-Solomon erasure codes. It can be found in [126].

Regenerating Codes

Proposed by Dimakis et al. [41], the Regenerating Codes seem a promising alternative to obtain a reliable and space efficient erasure codes, in a manner that consumes less communication resources to repair the missing redundancy fragments. These codes are based on Network Coding, which is a field of Information and Coding Theory. The basic idea is to combine small parts of packets flowing in a network (usually making linear combinations or simple XORs), in contrast to just storing and forwarding them.

To prove the efficiency of their codes, they use an *Information Flow Graph* with a network coding theory. Then, they derive the bounds on the amount of information that needs to be exchanged in order to recreate one missing fragment, as a function of the repair degree d (the number of peers that are contacted to reconstruct the fragment). They show that the amount of information to transfer can be close to the size of one fragment. There are two special cases of these codes: the Minimum-Bandwidth Regenerating Codes (MBR), that consumes less bandwidth to repair but cost more in space overhead. The second, Minimum-Storage Regenerating Codes (MSR), has the same space efficiency as the Reed-Solomon, but consumes slightly more bandwidth to repair.

However, in their first proposal, these codes do not have the Systematic Repair property. That is, when a fragment is regenerated, it is not exactly the fragment that was lost, but rather a different linear combination of the remaining ones. This property is named Functional Repair. Recent works do study Regenerating Codes with the Systematic Repair property, but it still in early stages development. Its feasibility was proved only for some small values of s and r [42]

Moreover, it is not yet proved that these codes could be applied in real systems. Duminuco et al. in [47] experimented these codes in a statical environment. They show that the amount of meta-data to be exchanged among peers (mainly sub-fragments coeficients) could be prohibitive.

Availability Expressions

Data availability means the probability that the data block is accessible at any given time. That is, it gives the ability for the user to retrieve its contents when desired. Whereas *data*

durability means that the data block is not lost due to permanent disk failures. Maybe the data is not available at a given moment, due to transient disconnections, but as soon as the missing peer(s) comes back the data will be made available again. Note that availability implies durability, but the opposite is not true. Data durability can also be defined as the expected lifetime of the data.

In the following, we give two expressions to estimate the availability of *one data block* when using replicated and erasure coded systems. We assume that the peers are independent of each other. Let a be the peer's availability, that is the probability to be present in the system at any given time step.

Data Availability of Replicated System. Since any single peer among the k peers that hold one of its copies is enough to recover the original block, the resulting block availability is trivial [105]:

$$A_{repl} = 1 - (1 - a)^k$$

Data Availability of Erasure Codes. Here we give the availability expression for *one data block* coded with standard MDS codes (e.g., Reed Solomon, Regenerating Codes, etc.) [105, 41]:

$$A_{erasure} = \sum_{i=s}^{n} \binom{n}{i} a^i (1 - a)^{n-i}$$

These simple expressions are often found in the literature [125, 18, 76, 105, 41], however they do not consider the failure rate of peers and the repairing time of the lost redundancy, which can take too long to finish and impacts negatively this metric. Throughout this dissertation we give methods that indeed takes into account these processes.

Furthermore, other codes (e.g., Hierarchical, Hybrid, etc.) may need the presence of less or more than s fragments, which invalidate this expression.

1.2.4 Repair Service

In this section we give details of the two crucial sub-systems that any distributed storage system should have: a *monitoring* subsystem and a *repairing* subsystem.

Monitoring

The monitoring subsystem needs to continuously monitor the redundancy level of data blocks to decide if a repairing process of missing redundancy (namely *reconstruction*) needs to be started. We assume that a general P2P overlay network has a layer that can *detect node departures*. Different architectures are possible to monitor the system's state. As explained in Section 1.2.2, the organization of the P2P overlay can be centralized or decentralized. If it is centralized, one (or many) dedicated central server keeps the metadata information with the localization of each redundancy fragment of every data block.

This central server monitors if peers leave the system. In a fully decentralized architecture, the management of the monitoring system is also distributed, usually with the employment of a Distributed Hash Table substrate. In that case, each peer is responsible for monitoring a subset of other peers (neighbors). Additionally, the integrity of the data has to be checked periodically.

Repairing

When the monitoring system detects that a data block b needs to be repaired, a reconstruction takes place. Thus, either a central server or a peer is chosen to be in charge of the reconstruction of the missing redundancy fragments of b. This peer is called the *reconstructor*. The repair mechanism of a MDS code is done in three consecutive phases: (1) *retrieval*, in which the reconstructor has to download s fragments among the remaining block's fragments; (2) *recoding*, in which the reconstructor recreates the data block; and finally (3) *sending*, in which the reconstructed missing fragments are sent to different peers (including the reconstructor itself that may keep one of the regenerated fragments). We consider here that the CPU recoding time is negligible compared to the network transmission time. Therefore, the *reconstruction time* is the sum of the retrieval and sending phases, along with the elapsed waiting time to start the reconstruction. Figure 1.4 illustrates that process.

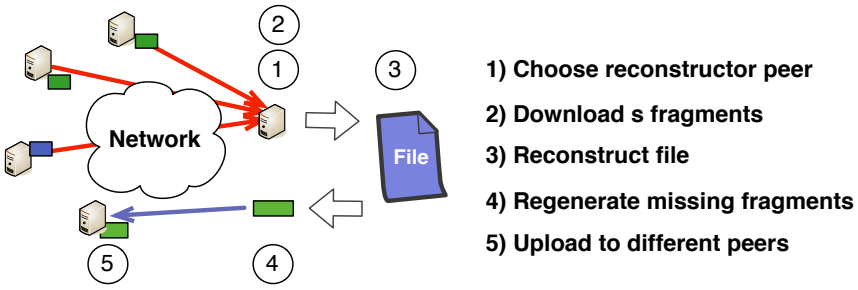


Figure 1.4: Repair process of Reed-Solomon erasure codes.

The amount of data transmitted per block b is then $(s + r - 1)L_f$ in total, with L_f the size of a fragment in bytes.

Remark on Monitoring Traffic. Usually, in an environment with collaborative peers and where disk crashes are the main cause of failures, then knowing which peers have left is enough to trigger the reconstruction process. The *control traffic* induced by this layer is, mainly, composed of periodic pings to test if the machines are up and running. Since the amount of traffic induced by the reconstruction transfers is much higher than the monitoring traffic, this later can be considered negligible here. Thus, the bandwidth consumption studied in this thesis is due solely to the reconstruction process.

Bandwidth Consumption

The bandwidth is one of the most scarce resource of the system. It should be noted that this repairing process consumes an important amount of network bandwidth to repair one missing fragment. Furthermore, if the peers bandwidth is limited then the elapsed time to finish the reconstruction process can be very long, which increase the probability of losing data exponentially. We discuss this behavior in detail throughout the thesis. In the following we give an overview of different reconstruction strategies that can be considered to reduce this traffic overhead.

Lazy Strategy. Delaying the reconstruction, i.e., waiting for a block to lose more than one fragment before rebuilding it, amortizes the costs of bandwidth usage over several failures. Hence, we study a *threshold based reconstruction policy* (often called Reactive Lazy Repair [20, 38]).

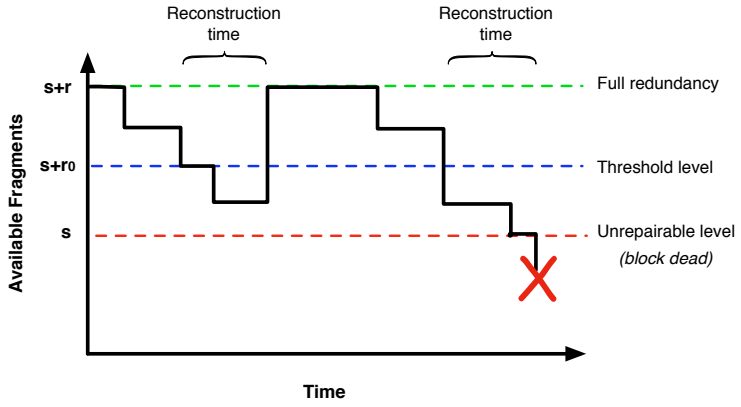


Figure 1.5: Block redundancy level over time. When using a deterministic lazy reconstruction strategy the repair process starts when the level reaches the threshold r_0.

As depicted in Figure 1.5, the reconstruction starts when the number of fragments of a block drops to a threshold value r_0. Note that, when r_0 is set to $r - 1$, the reconstruction starts as soon as a first piece is lost. This special case is called *eager policy* and the induced cost to reconstruct the block is very high, because it is necessary to transfer $s + 1$ fragments to reconstruct only 1 fragment. Setting a lower value for r_0 decreases the number of reconstructions (as the reconstruction starts only after that $r - r_0$ pieces are lost), but increases the probability of losing a block. *The lazy repair strategy is studied in more detail in the Chapters 3,4, and 7.*

Hybrid solution. Another approach to reduce the overhead of reconstruction is to keep a full-replica of the data along with the erasure coded fragments. That is, to use a *Hybrid* solution, mixing erasure codes and replication on the same system. Then, most of the time the communication overhead of reconstruction is very low: if the full-replica is present, the reconstruction is a trivial copy of a fragment. When the full-replica is lost a normal reconstruction is needed. *We study the advantages of this approach in Chapter 7.*

Proactive repair. In proactive strategies, the repairing process runs periodically or at random intervals of time to check the redundancy of blocks and proceed with the repairs. It is named *Fixed Repair Epoch* in [125]. These strategies can have some advantages, for instance a smooth usage of bandwidth. In a similar approach, Sit et al. [110] propose a solution that continuously introduces new replicas in the system whenever there is bandwidth available. Duminuco et al. [48] also study this approach but using erasure codes and mixing with reactive repair.

1.2.5 Related Work

In the following we present some studies that compare the use of the two redundancy schemes: simple replication and erasure codes. Then, in the next section we compile a list with the characteristics of several proposed systems found in the literature.

Replication versus Erasure Codes

Weatherspoon and Kubiatowicz [125] show that, in most of the cases, erasure codes use an order of magnitude less bandwidth and storage space than replication to provide similar system durability. The bandwidth accounted in their work is due to periodic repair procedures.

Bhagwan et al. [20] also came to the same conclusion when studying data availability for the TotalRecall system. They propose two interesting equations to estimate the amount of redundancy needed to obtain a certain level of data availability given the peer's availability.

Utard and Vernois [120] do a more detailed analysis that takes into consideration the repair time of lost fragments along with the peer's availability. They show that when the availability of peers is very small (high churn), then it is better to use replication instead of erasure codes, because of the intrinsic cost to repair the erasure codes which is higher than replication (at least s peers need to be available during a time window to reconstruct the data). However, in their study they assume an eager repair strategy, which is resource consuming.

Rodrigues and Liskov in [105] compare replication with the Hybrid solution (mixing both replication and erasure codes). They studied some real world traces (Overnet, Farsite and Planetlab) and proposed a failure model based on a membership timeout. After the timeout expired, the node state changes to "failed", and their data need to be repaired. They state that erasure codes are interesting for certain scenarios of availability. In some cases, however, the complexity of having such solutions does not pay the gains in storage efficiency. If the desired reliability is high, erasure codes are an interesting solution.

In this thesis, we focus our discussions on systems that use erasure codes, as we are *interested* in highly durable systems that consumes less resources. However, our methodology applies to both techniques. Recall that replication is a special case of erasure codes when $s = 1$.

Related Systems

The need for long-term archival system is not a recent topic. Many systems have been proposed in the past 15 years, mainly motivated by the growing interest in digital libraries. Some examples of early works are the Eternity Service [15], the Intermemory [56], and the Stanford Archival Repository Project [34]. Since then, many practical storage systems have emerged as peer-to-peer storage solutions. In the following we present a representative list of them. The survey by Hasan et al. [61], the thesis of Duminuco [45] and Dandoush [36] also compile some information. A summary is shown in Table 1.1.

- **Farsite** [23]: is an implementation of a server-less distributed file-system that exploit the idle resources of Desktop PCs at Microsoft. Reliability is ensured by using replication of the whole file among nodes (the use of erasure codes is also mentioned). They also propose placement schemes that take into account the peer's availability.

- **OceanStore** [69]: proposes an architecture to provide a global storage service based on a federation of untrusted servers that supports nomadic data. Their work have many folds, many of them are addressed in the Weatherspoon's PhD Thesis [124]. Pond [102] is an OceanStore prototype, that implements a file-system-like interface. It is based on the Tapestry DHT overlay network to store files (namely, data object). Their architecture is layered and also mixes the use of replication (to have a primary replica) and erasure codes (for archival). Any machine in the system may store archival fragments. The primary replica uses Tapestry to distribute the fragments uniformly throughout the system.

- **Cooperative File System (CFS)** [35]: it is a read-write storage system with NFS-like interface. Built on top of the Chord DHT location protocol, they propose the *DHash* put/get interface. A file is divided into blocks, to each block is assigned a different key. They proposed the use of replication of blocks as a redundancy scheme. The replicas are sent to the peers on the closest successors peers on the Chord ring. CFS is a read-only system from the perspective of the users. However, the publishers can update their work. Deletes are not allowed.

- **PAST** [108]: similar to CFS in function, it is as a large-scale persistent storage for immutable data built on top of Pastry DHT. It uses replication to add data redundancy. They propose a *replica diversion* scheme to allow load balancing, on which replicas of files are placed on peers outside of its neighborhood.

- **Ivy** [83]: is a multi-user read-write peer-to-peer file system that provides a NFS-like interface to clients. It is based on log files that record the changes made to files. These logs are stored and replicated using a DHT.

- **Pastiche** [33]: is a simple peer-to-peer backup system. The data are organized in chunks that are replicated to *buddies* chosen among the Pastry DHT neighbors.

These buddies are nodes that store similar content, hence storage space can be saved when the content are equal.

- **pStore** [17]: it is a distributed backup system with file versioning that has focus on security. In the pStore the data are cut into fragments that are then replicated among all nodes of the network at random.
- **TotalRecall** [20]: is another peer-to-peer storage system that stores files using either replication or erasure codes. Its architecture is similar to the other systems. Furthermore, they propose different repair strategies: the eager reconstruction (when using replication), and they propose a lazy repair (when using erasure codes).
- **Glacier** [59]: is a backup storage system proposed to be tolerant against many correlated failures. To achieve that it uses proactive repair and maintains a high level of redundancy using both replication and erasure codes. The placement strategy stores objects with similar keys on the same set of peers.
- **Pastis** [25]: is a multi-user decentralized read-write file system. It is built on top of a modified version of PAST/Pastry DHT. Pastis stores file's blocks in inode structures similar to the Unix Files System (UFS). Thus, it provides an NFS-like interface. The fault-tolerance and availability is provided by the PAST layer, hence, full replication of the data blocks.
- **BitVault** [130]: is another storage system of immutable data within a local network. It is a continuation of the RepStore [131] project. Peers are organized in a Xring DHT [133] that provides membership services. It uses replica of files for redundancy. The repair is done eagerly.

Table 1.1: Characteristics of some proposed distributed storage systems

System	DHT	Object type	Redundancy scheme	Repair policy	Data access	Data placement
Farsite	Groups	file	replication	eager	read-write	availability
OceanStore	Tapestry	file/block	both	eager/lazy	read/write	global/neighbors
CFS/Dhash	Chord	block	replication	eager	updates	neighbors
PAST	Pastry	file	replication	eager	read-only	neighbors
Ivy	Chord	modif. log	replication	eager	read/write	neighbors
Pastiche	Pastry	chunk	replication	-	load	buddies
pStore	Chord	block	replication	eager	read/write	random/global
TotalRecall	any	file	erasure/repl.	lazy	DHT	random/global
Glacier	any	any	both	proactive/eager	read/write	Bloom filter
Pastis	PAST/Pastry	inode/block	replication	eager	read/write	neighbors
BitVault	XRing	object	replication	eager	load balance	load balance

The objective of this compilation is to gather the general properties of the proposed systems that are relevant to this thesis. We do not aim at specifying the working details of a new storage system, but instead we aim at studying models that harness its behavior. Indeed, these models can then be used to improve its overall performance in several aspects. In this compilation we do not address other, not less important, issues, e.g., security, privacy, access efficiency, etc.

Open-source Solution

There is an effort by the open-source community to create a global storage cloud. The Tahoe-LAFS [127] project have been widely disseminated and is already included in the packages of the Ubuntu Linux distribution. Its implementation uses the Reed-Solomon erasure codes to add redundancy. Its source code is also available as open-source. Any user can create its own P2P network or participate in public ones.

Commercial Solutions

Commercial companies, like Wuala [2], CrashPlan [3], and Ubistorage [4], exploit the erasure coding technology to deliver a reliable backup solution. They use the network formed by client peers to store the redundancy. In Wuala, a user can *buy* storage space, or, more interestingly, *trade* local storage space and bandwidth time in order to gain reliable storage in the Wuala network. The amount of earned storage is a function on the amount of space given and the amount of time the computer is plugged into the network. A minimum of 4 hours per day is needed[5].

Ubistorage is a French company that provides data storage solutions for small and medium companies. Their services comprise the deployment of a dedicated terminal on their clients. This terminal acts as a Network Attached Storage (NAS) device that stay turned on permanently. The company then exploit these devices to store the redundant data coded with erasure coding. In the case of failure, the terminal is replaced and the data is recovered.

Other solutions that are rapidly growing are the on-line storage services. Mainly motivated by the *cloud computing* [16] era that requires data storage on-demand. That is, a service that is flexible and elastic, while keeping data reliably. Examples of these services are the AmazonS3 [6], the Google Storage [7], and the Microsoft SkyDrive [8]. Their services are focused on global availability of data, fast access and some level of durability.

For instance, in the Amazon S3 service agreement[9], they propose two kinds of storage solutions: the reliable one, *"Designed to provide 99.999999999% durability and 99.99% availability of objects over a given year. Designed to sustain the concurrent loss of data in two facilities."*; or a less expensive Reduced Redundancy Storage, which gives 99.99% of durability, which is designed to sustain the loss of data in a single facility only. The business model is based on monthly fees per GB used and a cost per million of operations performed.

[2] http://www.wuala.com
[3] http://www.crashplan.com
[4] http://www.ubistorage.com/
[5] (as service agreement in August 2010)
[6] http://aws.amazon.com/s3/
[7] http://code.google.com/intl/pt-BR/apis/storage/
[8] http://explore.live.com/windows-live-skydrive
[9] (as accessed in August 2010)

1.3 Summary

In this section, we compile the main design decisions and the variables that have an influence on the performance of P2P storage systems. Figure 1.6 shows a mind map of the main topics around P2P storage that are discussed throughout this text. The interaction between these branches are complex as they affect each other. As depicted in the map, the system analysis is divided in three big dimensions: *Design*, *Environment*, and *Metrics*. They are detailed as follows:

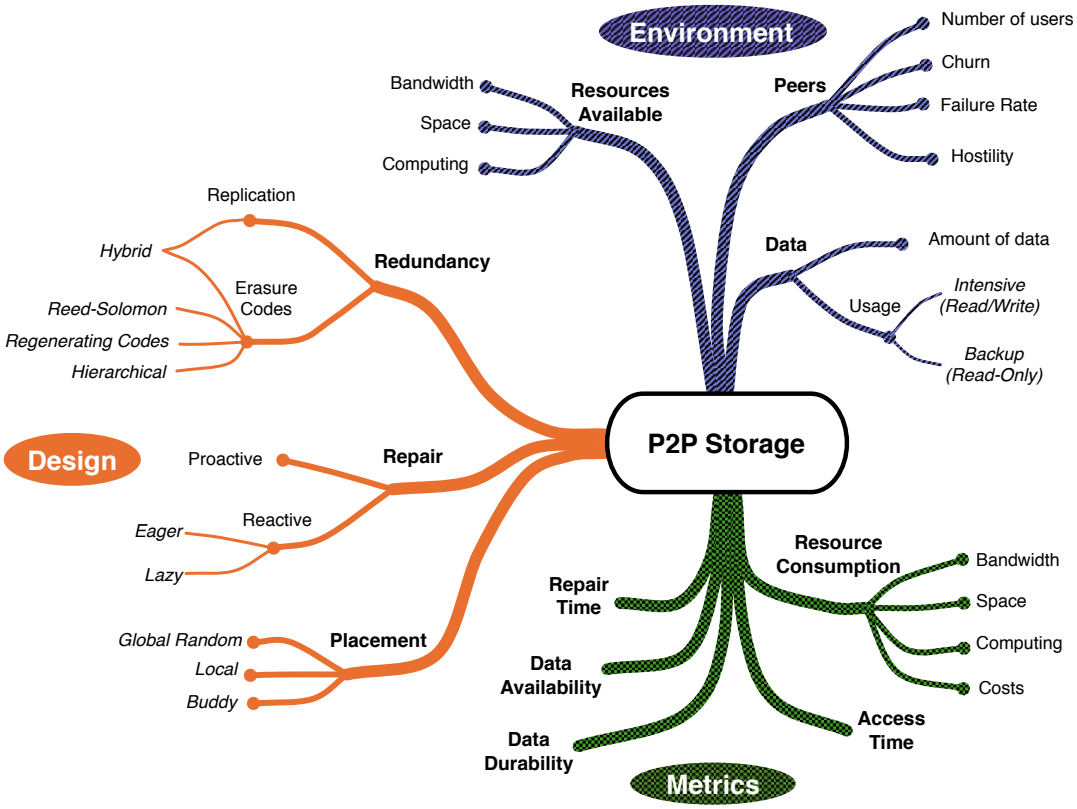


Figure 1.6: Sketch of the peer-to-peer storage system evaluation: design decisions, environmental variables and analyzed metrics.

- **Design Decisions** (on the left): are the parameters chosen when specifying, developing or configuring the system. Usually they are chosen according to the environmental variables that the system is planned to work with. Also, the design depends on desired performance metrics. The design decisions relevant to this thesis are be branched into three topics: redundancy scheme, repair mechanism and placement policy.

- **Environmental Variables** (on top): are the variables that affect directly or indirectly the system's behavior during run-time. The system analysis depends strongly on the environmental variables that the system is aimed to work for. As

pointed above, they also help the decision of the system design. Examples of environmental variables are: the kind of system usage (file-system, archival, etc.), the behavior of peers (availability rate, failure rate, space, etc.) and the amount of resources available (bandwidth and space).

- **Performance Metrics** (on bottom): the performance metrics for such systems are numerous. The most common ones are: data availability, data durability, space usage and bandwidth consumption. Our goal is to obtain a system with a maximum of data availability and durability, but using the minimum of space overhead and bandwidth consumption.

1.4 Contributions and Organization

The contributions of this thesis are organized as follows:

- **Chapter 2 - Methodology:**

  We give an overview of the analysis techniques used throughout the thesis. First we give some background on the use of discrete time Markov chains to estimate the mean behavior of dynamic systems. Then, we present a simple example of modeling the populations of rabbits. This small example already exhibits the problematic of capturing the variations around the mean value. Then, we describe how to better capture the system dynamics by presenting the insights of a stochastic Fluid model. We finish with the description and discussion of simulation techniques.

- **Chapter 3 - Mean Behavior and Guideline to Lazy Repair:** We model a general storage system by using a Markov chain model. This model allows us to take into account the effects of disk failures along with the time consumed by the self-repairing process. We confirm that a lazy repair strategy can be employed to amortize the repairing cost, mainly bandwidth. We then derive closed-form mathematical expressions that estimate the system average behavior. These formulas give a good intuition of the system dynamics. Our contribution is a guideline to system designers and administrators to choose the best set of parameters.

  The results presented in this chapter appeared in [3, 13] and were published in [4, 10].

- **Chapter 4 - Capturing the Variations:** We propose and study fluid models that assess the variations on the bandwidth consumption and the probability of data loss. These variations are caused by the simultaneous loss of multiple data blocks that results from a peer failure (or a peer leaving the system). In addition to its expectation, it gives a correct estimation of its variations.

  The results presented in this chapter appeared in [3] and were published in [4].

- **Chapter 5 - Analysis of the Repair Time under Limited Bandwidth:** The speed of the repair of a data block is crucial for its survival. This speed is mainly determined by how much bandwidth is available. But in practice, concurrent reconstructions

do compete for the same bandwidth. In this chapter, we propose a new analytical framework that takes into account this correlation. Mainly, we introduce queuing models in which reconstructions are served by peers at a cadence that depends on the available bandwidth. Moreover, we study the efficiency of different scheduling policies when serving data. The goal of this study is to provide a tight estimation of the probability of data loss under constrained bandwidth.

A paper with the results of this chapter is under review.

- **Chapter 6 - Data Placement Strategies:** We study the impact of different data placement strategies on the system performance. This study is motivated by practical peer-to-peer storage systems that store data in logical neighbors. We use simulations and combinatorial models to show that, without resource constraints, the average values are the same no matter which placement policy is used. However, the variations in the use of bandwidth are much more bursty under the local policies (in which the data are stored in logical neighbors). When the bandwidth is limited, these bursty variations induce longer maintenance time and henceforth a higher risk of data loss.

  The results presented in this chapter appeared in [2] and were published in [9, 1].

- **Chapter 7 - Hybrid Coding:** The reconstruction process of storage systems based on erasure codes (e.g., Reed-Solomon) have an important overhead in bandwidth consumption. In this chapter, we study the use of Hybrid coding: mixing erasure codes and simple data replication, i.e., keep one replica of the data along with the erasure codes. We model this systems using Markov chains, and then derive closed-form expressions to approximate the bandwidth usage and the system durability. We show when the Hybrid system has better trade-offs between bandwidth, space usage, and durability.

  A paper with the results of this chapter is being prepared for publication.

Appendices: We discuss briefly two orthogonal works done during this thesis:

- The first is the use of Evolving Graphs for performance evaluation of routing protocols for mobile ad hoc networks with known connectivity patterns. This study started during my Master studies, at University of São Paulo, Brazil. *The results of this work were published in [5, 7, 8].*

- The second work, is the proposition of a stackable Copy-on-Write file versioning device using Linux Device-Mapper (namely *Corral*). *This work was presented in [12].*

CHAPTER 2

Methodology

This chapter gives an overview of the modeling techniques used throughout the thesis. First we give some background on the use of discrete time Markov chains to estimate the mean behavior of dynamic systems. Then, we present a toy example of modeling a rabbit population. The idea is to give the insights that will be useful in the Chapter 3. This small example already exhibits the problematic of capturing the variations around the mean value. Then, we describe how to better capture the system dynamics by presenting the insights of a Fluid model, which will be useful in the Chapter 4. We finish with the description and discussion of simulation techniques used to validate the models proposed in this thesis.

2.1 Markov Chains

A model is an abstraction of a *system* that captures some part of its behavior in order to be analyzed. When studying dynamic systems that evolve during time, we can model them as a time dependent *stochastic process*. That is, the behavior of the system is represented as a probabilistic model that evolves randomly in time. Thus, a discrete time stochastic process X is family of ordered random variables $\{X(t) : t = 0, 1, 2, \ldots\}$ on a given probability space. The set of possible values of $X(t)$ is known as the *state space* S of the stochastic process [22]. At a time t the process is in state $X(t) \in S$.

A stochastic process constitutes a *Markov chain* if the next probabilistic behaviour of the process depends only on the present state of the process. It is not influenced by its past history. This is called the *Markovian property*. Despite its very simple structure, the Markov chain model is a extremely useful tool to understand the behavior of a wide variety of dynamic systems. Specifically, a Markov process allows us to model the uncertainty in many real-world systems that evolve dynamically in time. The basic concepts of a Markov process are *state* and *transition*. The "art" of modelling is then to find an adequate state description such that the associated stochastic process exhibits the Markovian property [116].

2.1.1 Definition

Formally, consider a sequence $\{X(t) : t \geq 0\}$ of a random variable X on a discrete state space. Let us assume X in a finite state space S whose elements are numbered $[0..n]$, where n is the cardinality of S minus one. If $X(t) = i$, then the process is said to be in state i at time t. Whenever the process is in state i, we note P_{ij} the fixed probability that it

will be next in state j. In a Markov chain, the conditional distribution of any future state $X(t+1)$ depends only on the present state, $X(t)$. That is,

$$P\{X(t+1) = j | X(t) = i\} = P_{ij}.$$

The probabilities P_{ij} are called *one-step transition probability*, and represents the probability that the process will, when in state i, next make a transition into state j. We have that

$$0 \leq P_{ij} \leq 1, \forall i, j \in [0..n], \text{ and } \sum_{j=0}^{n} P_{ij} = 1, \forall i \in [0..n].$$

Then, $\mathbf{P}$ is the one-step transition matrix defined by the probabilities P_{ij}, for every value of i, j. We assume that this matrix have rows that sum to 1. Given an initial state $X(0)$ at time zero, the Markov chain evolves over time. That is, at each time step the next step is defined according to the one-step transition probabilities, P_{ij}.

When studying the behavior of X during time, one can be interested in the probability of the system to be in a given state at a given time step t. We define then for each $t \geq 0$, the *t-step* transition probabilities as

$$P_{ij}^{(t)} = P\{X(t) = j | X(0) = i\}, \forall i, j \in [0..n],$$

that is, the probability to reach the state j at time t from state i at time 0. By the Chapman–Kolmogoroff equations we have that the *t-step* transition probabilities $P_{ij}^{(t)}$ can be calculated recursively from the one-step probability $P_{ij}^{(1)}$. Note that $P_{ij}^{(1)} = P_{ij}$. Indeed, the *t-step* transition matrix is the matrix product $\mathbf{P}^t = \mathbf{P}^{t-1} \cdot \mathbf{P}$.

2.1.2 The Equilibrium Probabilities

In this section we discuss the equilibrium distribution probabilities of the Markov chain. The equilibrium distribution, noted by the state vector $\pi_j, \forall j \in [0..n]$, is the probability of the process to be in a given state j after a long-run. A finite Markov chain has an unique equilibrium distribution if it is *ergodic*, that is, for every state i it is *aperiodic* and *irreducible*.

Periodicity: the period q_i of a state i is given by $q_i = gcd\{n : P[X(n) = i | X(0) = i] > 0\}$, where gcd denotes the greatest common divisor. Then, the state i is aperiodic if $q_i = 1$, i.e., the returns to state i occur at irregular times.

Irreducibility: the state i is irreducible if from i it is possible to reach every other state $j, \forall j \in [0..n]$. If there is an absorbing state, the chain is not irreducible.

If the Markov chain has these properties, applying the operator $\mathbf{P}$ multiple times, shows that $P_{ij}^{(t)}$ converges to some value as $t \to \infty$, which is the same for all i. In other words, the system converges to the equilibrium distribution of the process to be in state j after a large number of transitions. Note that this value is independent of the initial state.

Formally, the probability distribution π_j, $j \in [0..n]$, is said to be a stationary equi-

librium distribution for the Markov chain with one-step transitions P_{ij}, when $\pi_j = \lim_{t\to\infty} P_{ij}^{(t)}$, $j \in [0..n]$. Then, π_j is the unique non-negative solution of:

$$\pi_j = \sum_{k=0}^{n} \pi_k P_{kj},\ j \in [0..n] \quad \text{and} \quad \sum_{j=0}^{n} \pi_j = 1.$$

Solving. There are many methods to calculate such equilibrium distribution [116]: it can be calculated iteratively, iterating with the matrix $\mathbf{P}$ until the vector π reaches the equilibrium; another method is to calculate directly by solving the above linear system using Gauss-Jordan elimination; more generally, one can use any known method to find the eigenvector with eigenvalue equals 1. The Perron-Frobenius theorem ensures that such a vector exists for a stochastic matrix. That is

$$\pi = \mathbf{P}\pi,$$

which is equivalent to solve $\pi(\mathbf{P} - I_n) = 0$, where I_n is the identity matrix of size n. This system can be solved by adding the normalization equation $\sum_{j=0}^{n} \pi_j = 1$.

2.1.3 Toy Example

In the following, we give an example of a rabbit population. Every rabbit can be either inside (IN) the jail or be playing outside (OUT) in the garden. The time is discrete with a granularity of one day. Hence, starting at time $t = 0$ the system advances one day per time-step. Two kinds of events could happen to this population at each time-step:

Figure 2.1: Raving Rabbids (*Les lapins crétins*). Copyright Ubisoft.

- When it occurs to be a *sunny day*, the rabbits that are inside the jail can go out to play outside, with a certain probability p_{exit}. The outside rabbits remain outside.

- When it occurs to be a *cloudy day*, the outside rabbits could get inside, with a probability p_{enter}. The rabbits inside remain inside.

Sunny and cloudy events are treated as a sequence of Bernoulli trials [106]. Sunny days have probability p_{sunny} to occur and cloudy days $1 - p_{sunny}$.

Note that, the outside population can be deduced from the number of inside rabbits. We are interested in the *system expected state*. That is, the *probability that the system is in a given state* at a random moment. What is the expected number of rabbits inside and outside, for given values of p_{sunny}, p_{enter} and p_{exit}?

2.1.4 Modeling one Rabbit

First, we construct a Markov chain model to represent the behavior of one single rabbit. From that chain, it is very easy to estimate the expected number of inside and outside rabbits at the system's steady state. As depicted in Figure 2.2, the chain has 2 states, $X = [0, 1]$ (note, 0 := IN and 1 := OUT), and has 4 transitions as follows:

$$\begin{aligned} P_{0,0} &= (1 - p_{sunny}) + p_{sunny}\cdot(1 - p_{exit}) = 1 - p_{sunny}\cdot p_{exit} \\ P_{0,1} &= p_{sunny}\cdot p_{exit} \\ P_{1,0} &= (1 - p_{sunny})\cdot p_{enter} \\ P_{1,1} &= p_{sunny} + (1 - p_{sunny})\cdot(1 - p_{enter}) = 1 - (1 - p_{sunny})\cdot p_{enter} \end{aligned}$$

In other words, when the rabbit is inside, $X(t) = 0$, then it remains inside if it is a cloudy day $(1 - p_{sunny})$ or if it is a sunny day and it has decided not to go outside $p_{sunny}\cdot(1 - p_{exit})$. If the rabbit is outside, $X(t) = 1$, it can get inside if it is a cloudy day and it has decided to get in $(1 - p_{sunny})\cdot p_{enter}$. The other cases are symmetric.

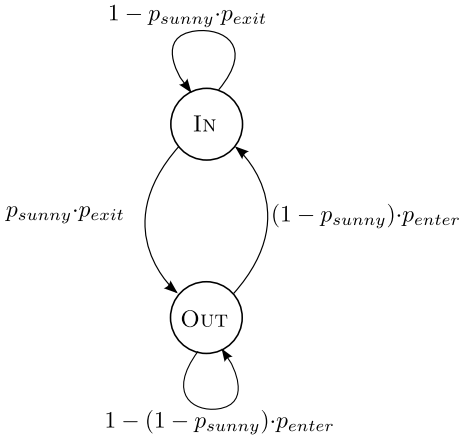


Figure 2.2: Simple Markov chain to model the one rabbit with 2 states.

Solving. For example, when $p_{sunny} = 1/2$, $p_{exit} = 1/2$ and $p_{enter} = 1/2$, we obtain the one-step transition matrix as follows:

$$\mathbf{P} = \begin{pmatrix} P_{0,0} & P_{0,1} \\ P_{1,0} & P_{1,1} \end{pmatrix} = \begin{pmatrix} 3/4 & 1/4 \\ 1/4 & 3/4 \end{pmatrix}$$

When iterating the transition matrix, we note that after a small number of iterations the probability distribution converges to the values $\pi = \{0.50, 0.50\}$ (when rounding to two decimal places).

$$\mathbf{P}^1 = \begin{pmatrix} 0.75 & 0.25 \\ 0.25 & 0.75 \end{pmatrix}, \mathbf{P}^2 = \begin{pmatrix} 0.62 & 0.38 \\ 0.38 & 0.62 \end{pmatrix}, \mathbf{P}^3 = \begin{pmatrix} 0.56 & 0.44 \\ 0.44 & 0.56 \end{pmatrix}, \ldots, \mathbf{P}^7 = \begin{pmatrix} 0.50 & 0.50 \\ 0.50 & 0.50 \end{pmatrix}.$$

This means that in a long run, the probability of the rabbit being inside is one half. Another simple method to obtain the stationary distribution is to solve the trivial system of

two variables, π_0 and π_1,

$$\pi_0 = 0.75\pi_0 + 0.25\pi_1$$
$$\pi_1 = 0.25\pi_0 + 0.75\pi_1$$

together with the normalizing equation $\pi_0 + \pi_1 = 1$. The solution is $\pi_0 = \pi_1 = 0.5$.

In simple words, this result means that the probability of the rabbit being inside or outside is the same 0.5. If we have a population of N elements, it is a common procedure to assume that all the elements of the population are independent of each other, then the equilibrium probability of a given state is the fraction of elements that are in that state. In our example, for a population of N rabbits, the equilibrium distribution gives the expectation of $N/2$ rabbits inside.

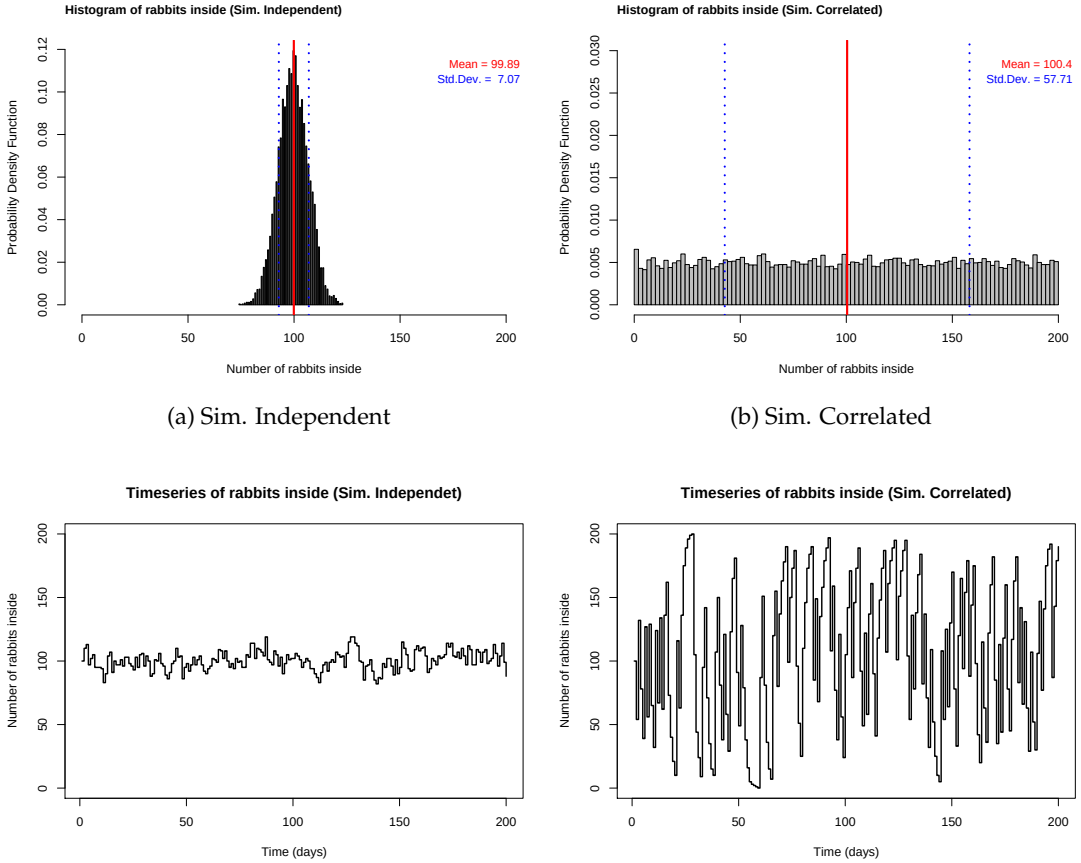
The Problem of Correlations. In this example, if we model one rabbit and we assume that it represents the behavior of the whole population, then we are neglecting the fact that rabbits on the same geographical place are strongly correlated. Indeed, rabbits that are on the same place are affected on the same way by sunny or cloudy events. When it is a sunny day the rabbits follow the same trajectory. There is no rabbit that will come inside in this day; the only possibility is to go outside. The extreme case is when all the population is correlated, as studied in the following.

Indeed, the strong correlation of these events is not well captured by this first Markov chain model. That is, the simple Markov chain gives the mean value of inside and outside rabbits at the steady state. However, the equilibrium probability does not give the distribution of probabilities at given moment of time. The remaining problem is: *for a given day, what is the probability that x percent of rabbits are inside at that day?* What is the distribution function of states probabilities? What are the variations around the mean?

To better illustrate that difference, we perform two kinds of numerical simulations of the system (a kind of Monte-Carlo simulations [106]). The first, as shown in Listing 2.1 (page 34), simulates the behavior of rabbits assuming they are independent of each other. That is, for every rabbit we choose two random numbers uniformly (i.e., perform two Bernoulli trials [106]): one for sunny or cloudy day; the other to move or stay. The second, Listing 2.2 (page 35), simulates the system taking into account the correlation of sunny and cloudy days. That is, first performs a Bernoulli trial to choose for sunny or cloudy day; then for every rabbit in the corresponding state (inside if sunny, or outside if cloudy) we perform the other Bernoulli trial. The difference is subtle. These two code snippets are appended at the of this chapter. They can be executed using the R statistical software [114].

Probability Distribution. Figure 2.3 shows the results of a simulation of $N = 200$ rabbits with equal probabilities $p_{sunny} = p_{enter} = p_{exit} = 1/2$. On the left side (a) and (c) are the results of the independent rabbits simulation (Listing 2.1), on the right side (b) and (d) are the results assuming a correlated system (Listing 2.2). On top we present the histogram of inside rabbits, and on the bottom we can check its behavior during time (only a short time window is shown). Note that both simulations give the same mean value of rabbits

inside (100). But, on the left side the values are very concentrated around the mean value. In fact this distribution is equivalent to N Bernoulli trials with probability $p = 1/2$, hence the standard deviation $\sigma = \sqrt{N \cdot p \cdot (1-p)} = \sqrt{N}/2 = 7.07$. On the right side, the simulation of the correlated system shows a somehow counter intuitive result. The distribution of the probability of rabbits inside is uniform. This means that at any given time, every combination of rabbits inside or outside is equiprobable.



(a) Sim. Independent

(b) Sim. Correlated

(c) Sim. Independent

(d) Sim. Correlated

Figure 2.3: Probability density function and timeseries of the rabbit population simulation ($N = 200$, and all probabilities are $1/2$). Both simulations show an average of 100 rabbits inside, but the variations are very different.

We conclude that modeling the dynamics of a population by one of its entities needs to be done with care. In the following, we give a second Markov chain model that indeed captures the real distribution of rabbits inside and outside at any given time.

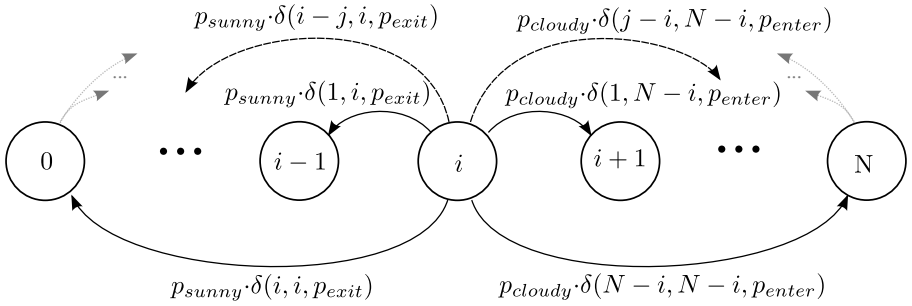


Figure 2.4: Complete Markov chain to model the population of rabbits with N+1 states. Only the transitions from state i are shown. Note here $p_{cloudy} = 1 - p_{sunny}$.

2.1.5 Modeling a Population of Rabbits

To take into consideration the correlation of the sunny and cloudy events presented above, we describe a more complete Markov chain. Indeed, in this complete chain every state represents the number of rabbits that are inside, from 0 to N. Recall that the number of rabbits outside can be deduced from the number of rabbits inside.

The transitions build a complete directed graph, every state having a transition to every other state (see Figure 2.4). Each transition P_{ij} is the exact probability to change from i rabbits to j rabbits in one day. More precisely, every transition P_{ij} is written as:

$$P_{ij} = \begin{cases} p_{sunny}\cdot\delta(i-j,i,p_{exit}) & \text{when } j < i \\ (1-p_{sunny})\cdot\delta(j-i,N-i,p_{enter}) & \text{when } j > i \\ 1 - \sum_{j=0,j\neq i}^{N} P_{ij} & \text{when } j = i \end{cases}$$

where $\delta(k,n,p) = \binom{n}{k}p^k(1-p)^{n-k}$ is the binomial distribution, i.e., the probability to obtain exactly k success in n trials with probability p. The first line ($j < i$) represents the sunny days, where the transitions go to lower indexes, i.e., the rabbits go outside. The second line ($j > i$) represents the cloudy days, when the outside rabbits go inside.

The stationary distribution of this chain gives the probability density function of the number of rabbits inside at any given time. The R code to obtain the stationary state of this chain is given in Listing 2.3 (page 35).

The results obtained by this chain match very precisely the results obtained by simulation. To give an idea of how the state vector evolves during time, Figure 2.5 shows the first five iterations of the complete chain, starting from the state vector with exactly 50% of rabbits inside. The probabilities are $p_{sunny} = p_{exit} = p_{enter} = 1/2$. We see that the system converges very quickly to the uniform distribution. As predicted by the simulations.

For the sake of completeness, let us try different values of probabilities to verify the results of the complete chain. Figure 2.6 shows the histogram of a simulation with probabilities $p_{sunny} = 0.25$ (assume they are in England!), $p_{exit} = 0.9$, and $p_{enter} = 0.3$.

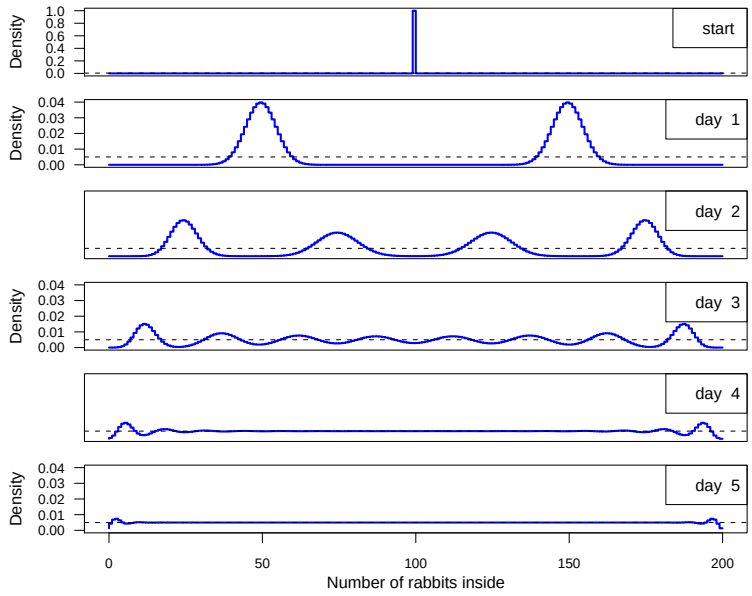
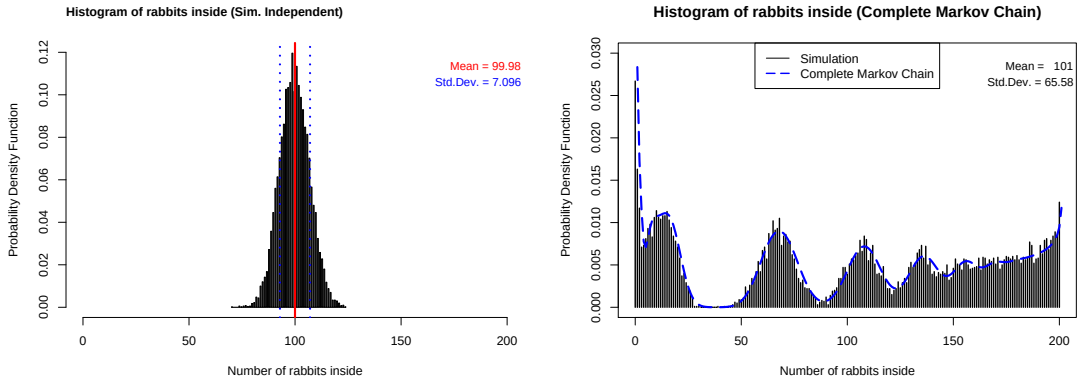


Figure 2.5: Probability density function of rabbits for the first 5 iterations. At the beginning, the state vector $X(0)$ (on top) is exactly with half of the rabbits inside. Then, it converges very quickly to the uniform stationary distribution.



(a) Sim. Independent

(b) Sim. Correlated and Complete Markov chain

Figure 2.6: Probability density function of rabbits simulation ($N = 200$, and probabilities $p_{sunny} = 0.25$, $p_{exit} = 0.9$ and $p_{enter} = 0.3$).

These values are chosen in such way that the expected number of rabbits inside and outside is still half and half (actually, any values that holds $p_{exit} = \frac{(1-p_{sunny})\cdot p_{enter}}{p_{sunny}}$ will have $\pi = [0.5, 0.5]$ as stationary equilibrium, this comes from the stability equations). On the left side, as expected, the simulation of the independent system has values very concentrated around the mean. However, the correct distribution of the correlated system has a strange form, and the analytical result obtained by the complete Markov chain matches very close the simulation (Figure 2.6(b)). For the curious, in this scenario

the probability of having between 30 and 40 rabbits inside in the steady state is close to zero; meanwhile, the probability of having no rabbits inside is considerable.

Conclusion. In this section, we have shown that the dynamics of the system can be precisely modeled by a Markov chain. However, this model suffers some scalability issues: the number of states depending linearly on the size of the population and the number of transitions depends quadratically.

Furthermore, note that the construction of this complete chain is feasible only because the system is very simple. If we model a system with more than two possible "locations" (e.g., inside, outside and eating), the chain would have an enormous number of states. Which turns the use of that approach unpractical. For example, if the number of rabbit locations is three, the number of possible states would be of order $O(N^2)$, and the number of transitions of order $O(N^4)$.

This engenders the development of practical methods that could give an intuition of the system dynamics. In the next section we discuss a stochastic Fluid model, which is more efficient in execution time and memory usage.

2.2 Stochastic Fluid Models

Fluid models are used to harness the dynamic characteristics of a population by simplifying its interactions. The principle of the fluid model is that when the number of elements in the population is large, the model can be approximated by a deterministic equation of its average behavior (often called "mean-field" approach) [24, 71]. This technique allows us to describe the macro behavior of the system dynamics in a simple way.

The stochastic fluid approach is used to simplify the complex stochastic Markov model by considering the average behavior of its entities. But the usefulness of this approach is that it allows us to capture the system averages and its variation in an efficient way.

2.2.1 Modeling a Very Large Population of Rabbits

Following the example of the rabbit population given in the last section, we show here its modeling based on a stochastic fluid model for the correlated case. The idea is to have a state vector Y with two states $Y_0 \in [0 \,..\, N]$, and $Y_1 \in [0 \,..\, N]$, (Y_0 := number of rabbits inside, Y_1 := number of rabbits outside). To this vector we apply one of two operators at each time step. One represents the events of sunny days and the other of cloudy days. They are constructed as follows:

$$\mathbb{M}_{sunny} = \begin{bmatrix} 1-\mu(p_{exit}) & 0 \\ \mu(p_{exit}) & 1 \end{bmatrix} \qquad \mathbb{M}_{cloudy} = \begin{bmatrix} 1 & 1-\mu(p_{enter}) \\ 0 & \mu(p_{enter}) \end{bmatrix},$$

where $\mu(p)$ is the fraction of rabbits that change states when an event occurs, and p the probability to move (either p_{exit} or p_{enter}). These operators are stochastic matrices that

represent the flow of rabbits when an event occurs. The first M_{sunny} is applied with probability p_{sunny}, and the other M_{cloudy} with probability $1 - p_{sunny}$.

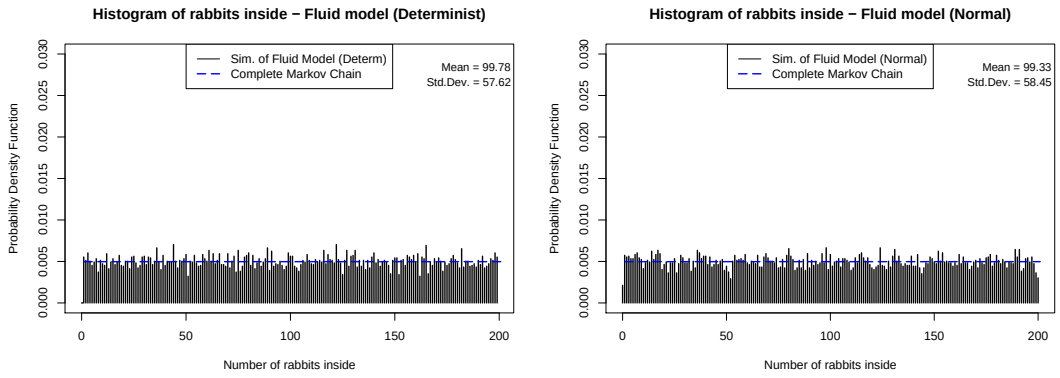
Then, the transition of the system is defined as

$$Y(t+1) = M(t) \cdot Y(t), t \geq 0$$

where $M(t)$ is a randomly chosen matrix of, either $\mathbb{M}_{sunny}$ with probability p_{sunny}, or $\mathbb{M}_{cloudy}$ with probability $(1 - p_{sunny})$.

First Order Model. First, we present a simple model, namely *deterministic*, in which the fraction of rabbits that change state is exactly the probability to move, that is $\mu(p) = p$.

Second Order Model. The other model, more refined, takes into consideration the small fluctuations around the exact fraction. Namely, *binomial*, it considers that the amount of rabbits that change states follows a binomial distribution with N elements and probability p. Then, we normalize to obtain a fraction $\mu(p) = randBinom(p, N)/N$. In fact, this almost [1] corresponds to the simulated system, in which for every rabbit we pick a random variable to decide if it changes state or not. Formally, it corresponds to N Bernoulli trials with probability p that, when normalized, give an average p and a standard deviation $\sigma = \sqrt{N \cdot p \cdot (1-p)}/N$. Remark that the standard error $\sigma/(N \cdot p)$ becomes very small when N is large, which means that for very large N the results of the second model leads to the first model.



(a) Fluid Model (determ.)

(b) Fluid Model (binomial)

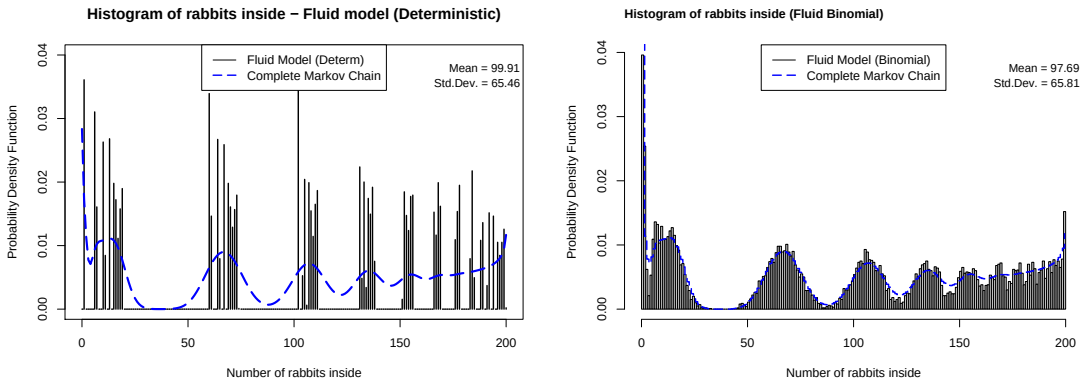
Figure 2.7: Probability density function of rabbits using the Fluid models ($N = 200$ and equal probabilities 0.5). Both models give almost the same results.

The first fluid model (deterministic) can be simulated using the source code shown in Listing 2.4 (page 36). The more refined (binomial) is shown in Listings 2.5 (page 36).

[1] Indeed, we say that it is almost the behavior of the real correlated system because we chose the binomial for a population with size *fixed as N*. Indeed, in the real system the number of rabbits in these trials depends on the system evolution. However, choosing a fixed value N already gives a very good approximation.

Figure 2.7 shows the probability distribution for $N = 200$ and probabilities $p_{sunny} = 0.5$, $p_{exit} = 0.5$, and $p_{enter} = 0.5$ (same parameters as in Figure 2.3). The results obtained using the *deterministic* and *binomial* fluid models are close to the complete Markov chain calculations, and the results of both models are very close.

Let us try different parameters. Figure 2.8 shows the results for a population of $N = 200$ rabbits and probabilities $p_{sunny} = 0.25$, $p_{exit} = 0.9$, and $p_{enter} = 0.3$ for both Fluid models. The first remark is that both models give the same average and the same standard deviation. The *binomial* model is very similar to the correct distribution calculated using the complete Markov chain. However, the *deterministic* Fluid model has a peaky distribution instead of the smooth behavior of the *binomial* model. This behavior is determined by the fact that there is no "noise" in the Fluid property, when an event occurs, exactly the same amount of elements will change state. Hence, some states are never reached.



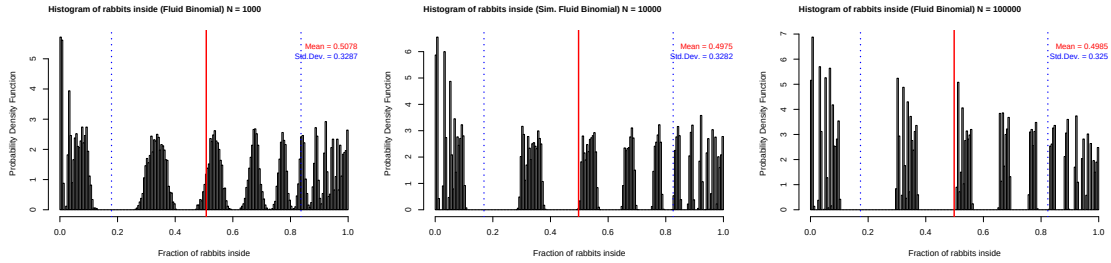
(a) Fluid Model (deterministic)

(b) Fluid Model (binomial)

Figure 2.8: Probability density function of rabbits using Fluid models (deterministic and binomial), $N = 200$ and the probabilities $p_{sunny} = 0.25$, $p_{exit} = 0.9$ and $p_{enter} = 0.3$. Their results are compared to the complete Markov chain.

To complete this study, we show in Figure 2.9 three scenarios with increasing amount of rabbits from 10^3 to 10^5. The probabilities remain the same (as in Figure 2.8). Note that the results for 10^5 rabbits are very close to the deterministic Fluid model. Which comes from the fact that for large values of N the binomial distribution is very concentrated around the mean value, hence it is closer to the deterministic model. These results comfort our claim that the deterministic Fluid model represents correctly the behavior of the system for large values of N.

The advantage of these fluid models is that they can be calculated very quickly! They do not depend on the number of entities in the system. Furthermore, the mean value and standard deviation can also be calculated analytically.



(a) $N = 10^3$ (b) $N = 10^4$ (c) $N = 10^5$

Figure 2.9: Probability density function of rabbit population using the binomial Fluid model, with same parameters as Figure 2.8, but with increasing values of N. As the values of N increase the distribution tends to the deterministic model.

Solving Analytically. The expected number of rabbits in each state can be derived from

$$\mathbb{E}[Y(t+1)] = \mathbb{E}[M(t)]\mathbb{E}[Y(t)]$$

The expectation of $M(t)$ is given by

$$\begin{aligned} \mathbb{E}[M(t)] &= \mathbb{E}[(p_{sunny}\cdot\mathbb{M}_{sunny} + (1-p_{sunny})\cdot\mathbb{M}_{cloudy})] \\ &= p_{sunny}\cdot\mathbb{E}[\mathbb{M}_{sunny}] + (1-p_{sunny})\cdot\mathbb{E}[\mathbb{M}_{cloudy}] \\ &= p_{sunny}\cdot\mathbb{M}_{sunny} + (1-p_{sunny})\cdot\mathbb{M}_{cloudy} \\ &= \mathbf{P}. \end{aligned}$$

Hence, $\mathbb{E}[M(t)]$ is equivalent to the transition matrix of the Markov chain, $\mathbf{P}$, for one rabbit. Then, since $\mathbb{E}[Y(t+1)] = \mathbf{P}\cdot\mathbb{E}[Y(t)]$, $\mathbb{E}[Y(t)]$ converges to the same stationary distribution π of the independent system already calculated in Section 2.1.4 (page 24).

Moreover, we can also calculate analytically the variation around the mean value. The standard deviation of the coordinates of the vector $Y(t)$ is calculated as follows

$$\sigma(Y_i(t)) = \sqrt{\mathbb{E}[Y_i(t)^2] - \mathbb{E}[Y_i(t)]^2}$$

we know that $\mathbb{E}[Y(t)] = \pi$, but $\mathbb{E}[Y(t)^2]$ is more difficult to calculate. It comes from $Y(t+1)^2 = (M(t)\cdot Y(t))^2$. Then we note that it depends on all the cross-products of $Y_iY_j, \forall i,j \in [0,n]$, which can be calculated by a linear system of equations. We omit the details here, but a equivalent analysis is done in Chapter 4 for the P2P storage model.

e

2.3 Network Simulation

Computer simulation is a cheap and powerful method to analyze the behavior of complex network systems. Similarly with the analytical models, simulation is all about creating models to represent (or mimic) the reality. Indeed, these models are simplified ab-

stractions of the real behavior of the studied system. If the abstractions are too complex, the simulation models are more subject to implementation errors, making it difficult to certify and validate the results. Further on, the simulation of complex models could take too long to finish, which makes the observation of rare events a difficult task. On the opposite, if the models are too simple then the accuracy could not be enough to observe the desired metrics. With that in mind, one needs to find what is the good trade-off between complexity of implementation and accuracy of the model when planing simulations.

In the early stages of this work, an extensive research was done on the pre-existent simulators of P2P and large-scale networks. Our findings show that there are many kinds of simulators available, each one with different goals. NS2 [85], OmNet++ [123] are the *de facto* standard for network simulation. Their behavior is highly trusted within the networking community. They are discrete-event simulators, organized according to the OSI model. However, they are built to simulate packet level communications with great detail, e.g., the TCP stack. Usually, the feasible scenarios evaluated with these simulators are around hundred of nodes, and the simulated time ranges from minutes to weeks.

By nature, the P2P systems comprise a large network with a high level of interaction between its entities. Hence, the simulation complexity of such systems also reaches large levels. In our case, we want to simulate storage systems with thousands, or even millions of nodes, each storing tens of thousands data fragments. A real storage system has millions, or even billions of objects. On such enormous systems, we want to study permanent data loss, which should be "rare" events. Hence, the constraints are multiple, usually in processing capacity and memory consumption.

In order to cope with this extremely challenging scalability issue, one possibility is to lower the complexity of the model [104]. For example, many works that study large-scale networks try not to simulate the lower levels of the network (packet transmissions, congestion, etc.), instead they create analytical models to simplify the simulation [21, 70]. As stated by Riley and Ammar [104], the simulation of large-scale systems with great detail is a challenging task.

Some surveys on P2P simulators can be found in the literature [84, 117, 111]. They provide a study and a classification of the simulators mostly used in research papers. Among them, the simulators that are close to our needs are the PeerSim [89], OverlayWeaver [86], FreePastry [51], OverSim [87], PlanetSim [77], SimGrid [27], and ProtoPeer [52]. Most of the time, the objective of these simulators is to implement new routing paradigms (DHT, Gossip, etc.) or to evaluate different applications (e.g., file-sharing protocols, video streaming, etc.).

P2P Storage System Simulation. We are interested in evaluating the characteristics of a large-scale P2P storage system that keeps data safe and reliable for long time. Hence, we are evaluating the durability and reliability of the system. Since the probabilities that are involved are very small, it is necessary to make long simulations to harness the occurrence of such rare events. In other words, we want to observe the macro-behavior of the system. This means that in our simulations we are not interested in modeling the

low level details of the network, neither in the interactions of the base protocols that are used by the system (e.g., TCP stack, DHT protocols, etc.)

Cycle-based Simulators. Our simulator is a *cycle-based* (or cycle-driven) simulator that is similar to PeerSim [89] in concept, that was developed mainly to evaluate epidemic (or gossip) protocols. In this kind of simulators the notion of time is discrete. Each time step has the same length, which is the *granularity* of the system and corresponds to one cycle of simulation. This granularity depends on the modeling assumptions and on the metrics that are being studied. For example, in our case, to evaluate the durability of the distributed storage system we use hours or seconds as granularity. However, when evaluating the epidemic protocols, each time step corresponds to one "cycle" of message exchanges. In this case the notion of time is not that important because the import metric is the number of messages transmitted/received to achieve a certain state (or goal). The advantages of using that approach are the simplicity of implementation and the gains in the speed of execution. This improvement comes when many events are aggregated and processed together (e.g., peer failures). As opposed to the traditional event-based simulator in which every event is triggered and treated separately.

2.4 Appendix: Listings

In this section we give the source code used to perform the simulations and to evaluate the complete Markov chain model presented in this chapter. These code snippets can be executed using the R statistical software [114].

Listing 2.1: Simulation of Rabbits (Independent system)

```
# Input: L: sim. length; N: number of rabbits; Psunny, Pexit, Penter: probabilities
# Return: vector with the number of rabbits inside over time
simIndep = function(L, N, Psunny, Pexit, Penter) {
    Xtime = array(0,L)
    Xtime[1] = N/2 #fraction of inside rabbits
    for (i in 2:L) {
        X = 0
        for (j in rep(0, Xtime[i-1]))                # inside rabbits
          if (runif(1) < Psunny && runif(1) < Pexit) # if sunny go outside
            X = X - 1
        for (j in rep(0, N - Xtime[i-1]))                  # outside rabbits
          if (runif(1) < (1-Psunny) && runif(1) < Penter) # if cloudy get inside
            X = X + 1
        Xtime[i] = Xtime[i-1] + X
    }
    return(Xtime)
}
```

Listing 2.2: Simulation of Rabbits (Correlated system)

```
# Input: L: sim. length; N: number of rabbits; Psunny, Pexit, Penter: probabilities
# Return: vector with the number of rabbits inside over time
simCorrelated = function(L, N, Psunny, Pexit, Penter) {
    Xtime = array(0,L)
    Xtime[1] = N/2   # fraction of inside rabbits
    for (i in 2:L) {
        X = 0
        if (runif(1) < Psunny) {            # sunny days
            for (j in rep(0, Xtime[i-1]))   # inside rabits
              if(runif(1) < Pexit)          # go outside
                X = X - 1
        } else {                            # cloudy days
            for (j in rep(0, N-Xtime[i-1])) # outside rabits
              if (runif(1) < Penter)        # get inside
                X = X + 1
        }
        Xtime[i] = Xtime[i-1] + X
    }
    return(Xtime)
}
```

Listing 2.3: Complete Markov Chain Model (Prob. distribution of rabbits inside)

```
# Input N: number of rabbits, Psunny, Pexit, Penter: probabilities
# Return: the probability distribution of rabbits inside
mcmCompleteChain <- function(N, Psunny, Pexit, Penter) {
    size <- N+1
    M <- matrix(rep(0,size*size), nrow=size, ncol=size);
    # create the transitions of M
    for (i in 0:N) {
        for (j in 0:N) {
            if (j < i) {
                M[j+1,i+1] <- Psunny*dbinom(i-j, i, Pexit) # left arrows
            } else if (j > i) {
                M[j+1,i+1] <- (1-Psunny)*dbinom(j-i, N-i, Penter) # right arrows
            }
        }
    }
    diag(M) <- -colSums(M)      # fill the diagonals
    M[1,] <- rep(1,size)        # add an equation (sum X = 1)
    b <- c(1,rep(0,size-1))
    X <- solve(M,b)             # solve M*X = b
    return(X)
}
```

Listing 2.4: Stochastic Fluid model (deterministic)

```
# Input: L: sim length; N: number of rabbits; Psunny, Pexit, Penter: probabilities
# Return: vector with the number of rabbits inside over time
fluidSimDeterm = function(L, N, Psunny, Pexit, Penter) {
    Msunny = matrix(c(1-Pexit, Pexit, 0, 1), ncol=2)
    Mcloudy = matrix(c(1, 0, Penter, 1-Penter), ncol=2)
    Y = c(N,0)                # initial state vector (1: inside, 2:outside)
    Ytime = array(0,L)        # Y(t)
    for (i in 1:L) {
        if (runif(1) < Psunny) {
            Y = Msunny %*% Y
        } else {
            Y = Mcloudy %*% Y
        }
        Ytime[i] = Y[1]
    }
    return(Ytime)
}
```

Listing 2.5: Stochastic Fluid model (binomial)

```
# Input: L: sim. length; N: number of rabbits; Psunny, Pexit, Penter: probabilities
# Return: vector with the number of rabbits inside over time
fluidSimBinom = function(L, N, Psunny, Pexit, Penter) {
    Y = c(N,0)           # initial state vector (1: inside, 2:outside)
    Ytime = array(0,L)   # Y(t)
    for (i in 1:L) {
        if (runif(1) < Psunny) {
            p = rbinom(1,N,Pexit)/N             # fraction of rabbits go outside
            Msunny = matrix(c(1-p, p, 0, 1), ncol=2)
            Y = Msunny %*% Y
        } else {
            p = rbinom(1,N,Penter)/N            # fraction of rabbits get inside
            Mcloudy = matrix(c(1, 0, p, 1-p), ncol=2)
            Y = Mcloudy %*% Y
        }
        Ytime[i] = Y[1]
    }
    return(Ytime)
}
```

CHAPTER 3

Mean Behavior and Guideline to Lazy Repair

In this chapter, we study a general storage system that uses erasure coding as redundancy scheme. To ensure durability, such distributed systems must have a self-repairing mechanism that maintains a minimum number of redundant fragments available in the network, even after multiple failures. This process, however, consumes an important amount of bandwidth, which is presumably the most scarce resource of the network. As shortly mentioned, the *lazy repair* strategy is an easy and practical solution to reduce the bandwidth consumption induced by the self-repairing process. Recall that in the lazy repair, the block reconstruction process starts when the number of available redundant fragments drops below a given threshold level. Hence, the dynamics of the system becomes more complex, which engenders the development of more refined models. We model and analyze the system steady-state using a discrete-time Markov chain. This model takes into account the effects of disk failures along with the time consumed by the repairing process. Then, it allows us to estimate the probability of losing data and the bandwidth usage. We also discuss a fundamental question for such systems that is how to choose the basic set of parameters, such as s, r, and r_0, to obtain an efficient utilisation of the bandwidth for a desired level of reliability.

The results presented in this chapter appeared in [3, 13] and were published in [4, 10].

Our Contribution

Our contribution is twofold.

First, we propose a simple Markov chain model to harness the storage system dynamics. From this model we derive approximated closed-form mathematical expressions that are then validated by simulations. These formulas give a good intuition of the system average behavior in function of its parameters. We evaluate the bandwidth consumption and the data loss rate for different scenarios. Then, we confirm that a lazy repair strategy can be employed to amortize the repairing cost even on high availability scenarios.

Second, from a practical point of view, we give a guideline of how to choose the best set of system parameters to obtain a desired level of reliability under a given constraint of bandwidth consumption, or vice-versa. We considered two scenarios: in the first one we study the trade-off between the bandwidth consumption and the loss rate for a *fixed storage space*; in the second scenario, for a given reliability, we show how to *provision the space overhead* to obtain an optimal bandwidth usage. We also discuss the impact of

the environment variables on the performance metrics, such as the number of peers, the amount of stored data, and the disk failure rate.

Related work.

Many works study the use of erasure codes schemes to add reliability to data storage. Distributed Hash Tables (DHT) based systems have been studied formally without the storage layer [75], but they have different requirements (e.g., network connectivity instead of data durability).

The behavior of a storage system using full replication is studied in [98], where a Markov chain model is used to derive the lifetime of the system, and other practical metrics like storage and bandwidth overhead. Piconni et al. [90], use a similar Markov chain to study a replicated system using DHTs and propose a method to estimate the rate of reconstruction under constrained bandwidth scenarios. They do not study the more complex case of erasure coding.

Datta and Aberer in [38] study analytical models for systems based on erasure coding for different lazy maintenance strategies. Wu et al. in [128] use differential equations to characterize the interplay between data repair and bandwidth consumption. Their analysis does not provide closed form expressions neither discussions on the choice of parameters. Utard and Vernois [120] propose a more sophisticated Markov chain that takes into account the availability of peers along with durability. However they assume the simple eager repair strategy and they do not give closed-form equations.

Similarly to our work, Alouf et al. in [14] use a Markovian analysis to evaluate the performance of systems based on erasure coding. They study two different schemes of data recovery (centralized vs. distributed) and estimate the data lifetime and availability. Their analysis does not address the bandwidth consumption.

Our work differentiates from those as we analyze the *bandwidth efficiency* along with the *data loss rate* of erasure codes using *lazy repair* strategy. Furthermore, we explore the parameter space to give a simple procedure to estimate their values.

Organization

The remainder of this chapter is organized as follows: in the next section we present a simple Markov chain model, along with a compilation of closed-form formulas to estimate the system metrics in Section 3.2. Then, in Section 3.3 we describe the Simulation Model. In Section 3.4, the results obtained by the simulations are compared to Markov chain model for a different set of scenarios. Then we give a method to choose the best system parameters in Section 3.5. Finally, in Section 3.6 we describe how to model different reconstruction times when using the Markov chain modeling.

3.1 Markov Chain Model (MCM)

In this section, we present the simple Markov Chain Model that we evaluate in the rest of the chapter. Similarly to the approaches found in the literature [98, 14, 38], this chain represents the behavior of a general distributed storage system by modeling a single data block. We derive the average values that characterize the system behavior (probability of data loss and average bandwidth consumption) from its stationary distribution.

3.1.1 States and Transitions

The behavior of a single block is modeled by a finite discrete time Markov Chain with time step τ.

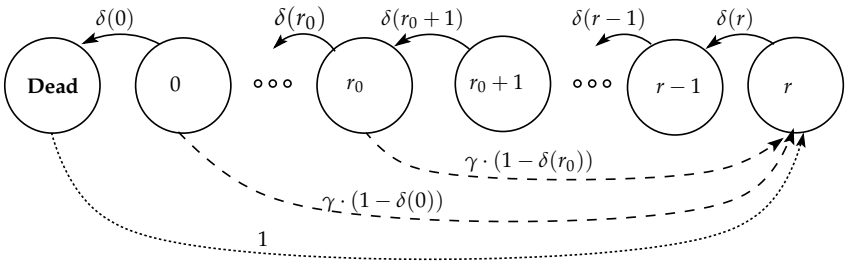


Figure 3.1: Markov chain modeling the behavior of one block.

The chain (as depicted in Figure 3.1) has $r+2$ states, that are the r levels of redundancy of a block b, plus a level 0 (no more redundancy), and a *Dead* state. We note $r(b)$ the number of remaining redundancy fragments of block b. Three different kinds of states can be distinguished:

- **Non-Critical:** when $r_0 + 1 \leq r(b) \leq r$;
- **Critical:** when $0 \leq r(b) \leq r_0$;
- **Dead:** when the block has less than s fragments,

Recall that r_0 is the threshold level to start the reconstruction process (i.e., *lazy repair*). A block can be affected by two different kinds of events: a *fragment loss*, which occurs when a peers that contain one of the fragment of block b fails; or the *block reconstruction*, which is the process of repairing the lost fragments of block b.

Fragment Loss

The probability for a block at level i to lose one fragment during a time step is denoted by $\delta(i)$ and is given by

$$\delta(i) := (s+i)\alpha$$

where α is the probability for a peer to experience a failure during the time step. That is, a block at level i has $s + i$ fragments present in the network. Hence, the probability $\delta(i)$ is the probability to lose one fragment among the i available fragments. A block with no more redundancy fragments may die with probability $\delta(0)$.

Note on the complete chain: For the sake of clarity of exposition, we do not describe here the most accurate and complex chain, but rather a simplified version (where unlikely transitions are ignored). For instance, a block at level i could lose many fragments during one time step. Hence, in the complete chain we add transitions from every state i, for every possibility of multiple failures j, from 1 to i, with probability

$$\delta'(i,j) = \binom{s+i}{j} \alpha^j (1-\alpha)^{s+i-j+1}.$$

The simplified chain, however, give very good approximations and provide the intuition of the system behavior for small values of α. We actually use the more sophisticated chain in our computations and comparisons.

Reconstruction

When a block becomes critical, $r(b) \leq r_0$, the reconstruction starts. The reconstruction is modeled as follows: the average duration of a reconstruction being noted θ (expressed in time steps), at each time step, a critical block has a probability $\gamma := 1/\theta$ to be rebuilt. In that case it goes to the top, r. Note that we also assume that if the block loses a fragment during a time step, it cannot be reconstructed during the same time step. If a block loses more than $r(b) + 1$ redundancy fragments before being reconstructed, it goes to the *Dead* state. In that chain, we choose to implement the reconstruction after the impact of failures. The other choice is also possible, it changes only slightly the transitions.

A Poisson reconstruction time is used, for mathematical tractability, and because we think it approximates well the random nature of network delays. Note that most other types of reconstruction could be captured by MCMs. For example, a reconstruction lasting a deterministic time can be modeled by labeling the states of a critical block by the progress of the reconstruction. A more detailed discussion on that follows in Section 3.6

In our model, due to the stability assumption (the number of blocks is constant), if a data block is lost then it is replaced by a new block with full redundancy and spread at random among peers (expressed by the transition from dead state to level r with probability one). This purely formal assumption does not affect the system behavior because dead blocks are rare events, but it makes the analysis more tractable.

More Refined Models. Furthermore, the Markov Chain approach allows us to model more complex systems or behaviors. For example several classes of peers, with different disk size, failure rate, availability, or bandwidth, can be introduced easily. For k families of peers, the state of a block would be a k-uple $\{n_i\}, i \in \{0, k-1\}$ where n_i is the number of fragments on peers of family i. This would lead to Markov chains with n^k states that

could be solved as long as k remains small.

A summary of the notations used throughout this chapter is given in Table 3.1.

Table 3.1: Summary of the main notations.

N	Number of peers
s	Number of initial fragments of a block
r	Number of redundancy fragments of a block
$r(b)$	Number of remaining redundancy fragments of block b
r_0	Reconstruction threshold value
L_{total}	Total amount of data in the system, in bytes
L_f	Size of a fragment, in bytes
L_b	Initial size of a block, in bytes ($L_b = s \cdot L_f$)
B	Total number of blocks in the system ($B = L_{total}/L_b$)
F	Total number of fragments in the system ($F = B(s+r)$)
$\overline{F}$	Expected number of fragments at steady state
$MTTF$	Peer mean time to failure (hours)
α	Peer failure rate ($\alpha = 1/MTTF$)
$\delta(i)$	Probability for a block at level i to lose one fragment
$\delta(i,j)$	Probability for a block at level i to lose j fragments
θ	Average number of time steps to reconstruct one block
γ	Prob. for a block to be reconstructed at a time step ($\gamma = 1/\theta$)
τ	Time step of the model

3.2 Approximations

In this section we present explicit expressions based on the simplified Markov chain that estimate the system main metrics: the average bandwidth, the data loss rate and the peek of bandwidth. These expressions give an intuition of the system behavior in function of its parameters, such as the erasure code settings: s, r, r_0, and fragment size L_f; and as the system characteristics: amount of data B, peer failure rate $\alpha = 1/MTTF$, and the reconstruction rate $\gamma = 1/\theta$. We provide good approximations for ratios $\alpha/\gamma \ll 1$, which is the case for practical systems where the block reconstruction process is much faster than the peer failure rate.

3.2.1 Stationary Distribution

The finite Markov chain presented above is irreducible and aperiodic. Hence, the probability to be in a state converges towards a unique stationary distribution denoted by P, where P_i is the stationary probability to be in state i. In a system where the blocks are distributed uniformly at random and the peers fails independently, we can say that each state in the chain represents the fraction of blocks at that state.

The stationary distribution can be computed exactly in time polynomial in r by finding the eigenvector with eigenvalue 1 or simply by a Howard Perron Frobenius iteration. The complexity is independent of the number of blocks B or of the number of peers N.

Note that, because of the simple form of the system (the system without the first row is lower triangular) it is easy to derive simple closed-formulas that estimate the distribution of the blocks' redundancy level. The stationary distribution of that chain can be computed by the stability equations as follows:

$$\begin{aligned}
P_{r-1} &= \tfrac{\delta(r)}{\delta(r-1)} P_r \\
P_{r-2} &= \tfrac{\delta(r-1)}{\delta(r-2)} P_{r-1} = \tfrac{\delta(r)}{\delta(r-2)} P_r \\
.. &= .. \\
P_{r_0+1} &= \tfrac{\delta(r)}{\delta(r_0+1)} P_r \\
P_{r_0} &= \tfrac{\delta(r_0+1)}{\delta(r_0)+\gamma\cdot(1-\delta(r_0))} P_{r_0+1} \\
.. &= .. \\
P_0 &= \tfrac{\delta(1)}{\delta(0)+\gamma\cdot(1-\delta(0))} P_1 \\
P_{dead} &= \delta(0) P_0
\end{aligned}$$

Hence, we can easy distinguish three groups of equations:

$$P_i = \begin{cases} \frac{\delta(r)}{\delta(i)} P_r, & \text{if } r_0 < i \leq r \\ \frac{\delta(i+1)\cdot P_{i+1}}{\delta(i)+\gamma\cdot(1-\delta(i))}, & \text{if } 0 \leq i \leq r_0 \\ \delta(0) P_0, & \text{if } i = dead \end{cases}$$

All probabilities can be expressed in function of $P(r)$, which in turn is computed by the normalization

$$\sum_{i=0}^{r} P_i + P_{dead} = 1.$$

Developing this expression and assuming small values of the ratio α/γ, the fraction of blocks at level P_r can be simplified as

$$P_r \approx \frac{1}{(s+r)\cdot(H_{s+r} - H_{s+r_0})},$$

where $H_n = \sum_{k=1}^{n} \frac{1}{k}$ is the harmonic number of n. Note that, hereafter we use the approximation $\ln(n) \approx H_n$. The fraction of blocks at each Non-Critical state i is then $P_i \approx \frac{(s+r)\cdot P_r}{(s+i)}$, which evaluates as

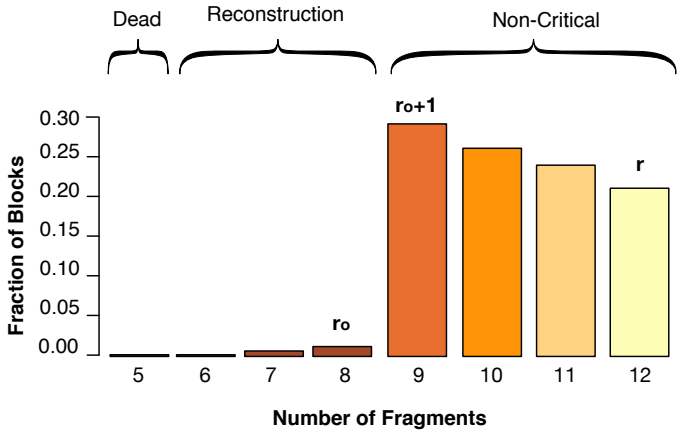


Figure 3.2: Distribution of blocks' redundancy level at the steady state. Parameters $N = 2000$, $B = 2{\cdot}10^5$, $s = 6$, $r = 6$ $r_0 = 2$, $MTTF = 180$ days, $\theta = 18$ hours.

$$P_i \approx \frac{1}{(s+i)\cdot \ln(\frac{s+r}{s+r_0})}, \text{ if } r_0 < i \leq r. \tag{3.1}$$

A rough approximation of the number of blocks in the Non-Critical states is:

$$P'_i \approx \frac{1}{r - r_0}, \text{ if } r_0 < i \leq r.$$

In fact, this approximation assumes that the blocks in Non-Critical states are uniform while other states are negligible. It is close to the exact values when $\alpha/\gamma \ll 1$, s is large and $r - r_0$ is small.

3.2.2 Distribution of Blocks' Redundancy Level

If every block is at maximum redundancy, the total number of fragments stored in the system is $F = B{\cdot}(s + r)$. However, at the steady-state this is not true. In that case, the average redundancy level, $\mathbb{E}[P]$, is in between r and r_0. Figure 3.2 illustrates the average number of blocks in each redundancy level. Note that the number of blocks in the Non-Critical states (levels 8 to 11) are not evenly distributed. We can then estimate the total number of fragments at the steady-state $\overline{F} = B \cdot \mathbb{E}[P]$, which is approximated by

$$\overline{F} \approx B{\cdot}(s + \frac{r + r_0}{2}). \tag{3.2}$$

3.2.3 Estimating the Bandwidth Consumption

To estimate the bandwidth consumed by the reconstructions, we first need to define the repair bandwidth inefficiency $\epsilon(i)$, as the amount of data that need to be transferred to

reconstruct i missing fragments: $\epsilon(i) = (s+i-1)\cdot L_f$, where L_f is the size of a fragment. That is, the peer in charge of the reconstruction must download s fragments from other peers and then send $i-1$ reconstructed fragments (assuming that the peer in charge keeps a fragment). When a block needs to be reconstructed, the number of missing fragments is in most of the cases $r - r_0$. Sometimes it could be a little bit larger if two fragments were lost during the reconstruction, which is rare when the ratio $\alpha/\gamma \ll 1$.

Average bandwidth consumption. At the steady-state the number of blocks that finish the reconstruction at each time step is equal to the number of reconstructions that start (by a cut argument in the chain). Then the average bandwidth consumption comes straightforward from the transition of the state r_0+1 to the state r_0. That is, the fraction of blocks that goes from the last *Non-Critical* state towards r_0, given by $\delta(r_0+1)\cdot P_{r_0+1}$.

For a system with B blocks, the average number of blocks finishing the reconstruction is

$$R_{avg} = B\cdot(s+r_0+1)\cdot\alpha\cdot P_{r_0+1}.$$

For each block, the amount of information to be transferred is, in most of the cases, $\epsilon(r-r_0)$. If the reconstructions are uniformly distributed, the average bandwidth consumption *per peer* is

$$BW_{avg} \approx \frac{R_{avg}\cdot\epsilon(r-r_0)}{N\cdot\tau},$$

which evaluates as

$$\boxed{BW_{avg} \approx \frac{B\cdot\alpha}{N\cdot\ln(\frac{s+r}{s+r_0})\tau}\cdot(s+r-r_0-1)\cdot L_f.} \tag{3.3}$$

Note that BW_{avg} does not depends on the reconstruction rate γ. This expression is valid for systems with ratio $\alpha/\gamma \ll 1$ (see Figure 3.5 in page 52, and corresponding discussion).

For the sake of verification, when $r_0 = r-1$ (eager repair) the expression evaluates to $BW_{avg}(eager) \approx \alpha B(s+r)L_f/N$. Recall that $B(s+r)L_f = L_{total}$ is the total amount of data in the system, then

$$BW_{avg}(eager) \approx \frac{\alpha\cdot L_{total}}{N}.$$

Peek of bandwidth consumption. It is known that the dynamics of a system with many blocks is not smooth at all. There are peeks of resource consumption when a peer fails, because many fragments are lost at the same time and trigger many reconstructions. The number of blocks that start reconstruction when a peer fails is

$$R_{start} = \frac{\overline{F}}{N}\cdot P_{r_0+1},$$

where $\overline{F}/N$ is the average number of fragments per disk, and P_{r_0+1} is the fraction of

these fragments that belongs to blocks that are at the last Non-Critical state (i.e., need to start the reconstruction). The amount of data transfer induced by these reconstructions is $Q_{peek} = R_{start} \cdot \epsilon(r - r_0)$ and can be approximated as

$$Q_{peek} \approx \frac{B \cdot (s + r)}{N \cdot (s + r_0 + 1) \cdot \ln(\frac{s+r}{s+r_0})} \cdot (s + r - r_0 - 1) \cdot L_f.$$

Note that, when $r_0 = r - 1$, the amount of data to be transferred is $Q_{peek}(eager) \approx B(s + r)L_f \cdot s/N$. This expression evaluates to $L_{total} \cdot s/N$. Hence, s times the average amount of data stored per peer, which confirms the high overhead of repairing a failed disk when using the eager repair.

Indeed, Q_{peek} can be used to calculate the average time to reconstruct the data of a failed peer, which in turn can be used to re-estimate the reconstruction rate γ for a given bandwidth capacity. A detail discussion on that metric is done in Chapters 4 and 5.

3.2.4 Estimating the Data Loss Rate

The evaluation of $P_{dead} = \delta(0) \cdot P_0$ gives the fraction of blocks that are lost per time step τ. Hence in a system with B blocks the data loss rate can be calculated as $LossRate = B \cdot P_{dead}/\tau$. When $\alpha/\gamma \ll 1$ it is closely approximated as

$$\boxed{LossRate \approx \frac{B}{(s + r_0 + 1) \cdot \ln(\frac{s+r}{s+r_0})\tau} \cdot \frac{(s + r_0)!}{(s - 1)!} \cdot \left(\frac{\alpha}{\gamma}\right)^{r_0+2}.} \quad (3.4)$$

3.3 Simulation Model (SM)

We developed a custom cycle-based simulator to evaluate several characteristics of the real system. The simulator does not aim at capturing the low level details of the network (such as packet level communication, traffic congestion or latency), but it focuses on the global evolution of blocks' states in the presence of peer failures and reconstructions[1].

Our goal is to simulate the behavior of the storage system for many years. Then, the choice of a cycle-based design is based on performance of execution, facility of implementation and validation. The time step (granularity) of the simulated system is one *hour*. Hence, all the events that occur on the same time step can be aggregated and processed together. This granularity is not far from reality. Note that, in practice it corresponds to a system where the control is done every hour to check if the peers are alive.

The simulator monitors precisely the evolution of the blocks in the system, that is, their redundancy level and location at each time step. For each peer, it stores a list of all the blocks having a fragment stored on it. The overview of the simulator is depicted in the Procedure 3.1. We describe here the default behavior of the simulator, but several other aspects were implemented: for example, different data placement policies; different reconstruction strategies; different failure models.

[1]The simulator can be downloaded from http://www-sop.inria.fr/members/Julian.Monteiro/p2p-storage-sim/

3.3.1 Initialization

For a general analysis we assume that the system is initialized with B blocks of data. Every block b in the system has the same size L_b. Also, every block has exactly the same amount of redundancy fragments, $s + r$ (full redundancy). The fragments are distributed uniformly at random among N peers. Thus, each peer starts with an average of $B(s + r)/N$ fragments.

3.3.2 Execution

When a disk failure occurs, the simulator updates the state of the blocks that have lost a fragment. The blocks that have reached the threshold level start the reconstruction process. To each of these blocks is assigned a reconstructor peer. We modeled two kinds of reconstruction processes: a *deterministic*, in which a reconstruction lasts a period of exactly θ time steps; and a *poissonian*, at each time step every block has a probability $1/\theta$ to finish the reconstruction.

We study the system characteristics in the steady-state. Hence, we make the simplifying assumption that the amount of data stored in the system B is kept constant over the time. Dead data blocks (i.e., blocks that have less than s fragments of redundancy) are replaced by a new block, or somehow re-injected in the system. The number of peers N is also kept constant over time, that is, failed peers are replaced by new peers. New peers appear empty.

3.3.3 Transient Phase (Warm-up)

We focus on the properties of the stationary state of the system. However, during the initial steps of the simulation the system is in a *transient phase*. The cycles corresponding to this phase are removed, hence they are not considered into the aggregated results.

There are different system metrics that can be considered to achieve the stationary state. For instance, when analyzing the Markov chain, one can use the second eigenvalue to estimate the speed of converge to the equilibrium distribution.

In the simulated system, the transient phase mainly occurs because the initial distribution of fragments in the system is different from the system steady state. For instance, in the initialization phase all blocks have the full redundancy ($s + r$ fragments), and the amount of data is evenly distributed among all peers. But, as already pointed out, in a system with lazy repair, this is not true after some period of execution. Figure 3.3 shows a timeseries of one year of simulation for a system with parameters $s = 6$, $r = 6$, $r_0 = 2$, $N = 2000$, $B = 2{\cdot}10^5$, $MTTF = 180$ days and $\theta = 18$ hours. We see that after approximately one hundred days (2400 time steps) the average number of blocks at each redundancy level reaches a constant fraction. Its distribution is the same as the one presented in Figure 3.2 (page 43). Nonetheless, this period of one hundred days is not sufficient to characterize the system steady state. Figure 3.4 depicts the average and std. deviation of the amount of fragments per disk. In the beginning all disks contain more or less the same amount of fragments $B(s + r)/N = 1200$. During the execution, the

Procedure 3.1 Overview of the cycle-based P2P storage simulator

```
initialize (N, B, s, r); {distribute B*(s+r) fragments among N disks}
time ← 0
while time not finished do

  {1. generate disk failures}
  for all block b in failed disks do
    update the state of b
  end for

  {2. monitor blocks' states:}
  for all block b in failed disks do
    if r(b) ≤ r_0 then
      assign a reconstructor peer for b
      add b to the reconstruction set
    end if
    if reconstructor peer of b has failed then
      re-assign a reconstructor and re-start reconstruction of b
    end if
  end for

  {3. handle the reconstructions:}
  for all block b in reconstruction set do
    if reconstruction finished then
      spread the rebuilt missing fragments at random
      remove b from the reconstruction set
    end if
  end for

  {4. ensure the stability of the system:}
  reintroduce fragments of the dead blocks
  replace crashed peers with new empty ones

  {5. collect statistics.}

  time ← time + 1
end while
{6. summarize results}
```

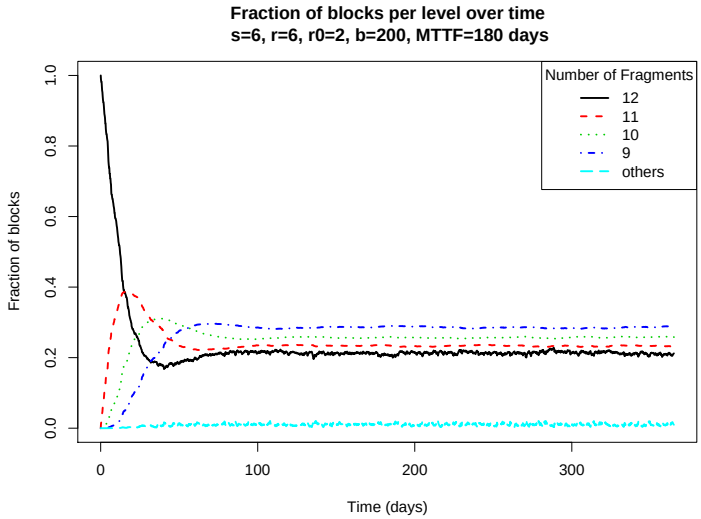


Figure 3.3: Timeseries of the number of fragments per block during the first year of simulation. After 100 days the system reaches the steady-state for this metric.

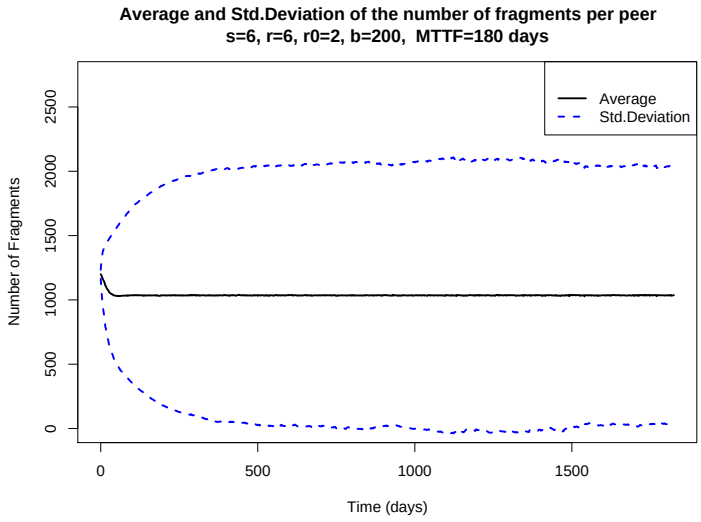


Figure 3.4: Timeseries of the number of the fragments stored per peer (average and std. deviation) during five years. After approximately 500 days the system reaches the steady-state for this metric.

average amount of fragments converges very quickly to a constant value of more or less 1000 fragments, as estimated by the Equation (3.2).

However, the variations around the mean value take more time to converge. In the given example, for a peer $MTTF$ of 180 days, the std. deviation of fragments per peer reaches the steady state only after 500 days. Again, this happens because in the steady state the number of fragments per disks are not close to the average. That is, failed disks

are replaced by empty disks. Hence, old disk have more fragments than young ones.

Summarizing, it is important to remove the first part of the simulation traces when studying the steady state properties. An estimation of this period could be done with the time where almost all disks have been replaced at least once.

Simulation Time. We evaluate several scenarios with different parameters. When studying the bandwidth consumption, for example, it is sufficient to simulate a few years of the system (e.g., 5 to 20 years) to achieve a good confidence interval in that metric. However, when studying the data loss rate (which are rare events) it is necessary to simulate much more time to achieve a good certitude on the number of dead blocks. Thus, for some scenarios we evaluate hundreds of years. Even with this simulated time, it is difficult to obtain precise results of the occurrence of rare events, which motivates us to develop precise analytical models that estimate correctly the system behavior.

3.3.4 Measured Metrics

At the end of each time step (cycle) the simulator keeps track of many performance metrics. The main ones are:

- number of reconstructions in progress (hence the bandwidth usage);
- number of starting, restarting, and finishing reconstructions;
- total number of fragments available;
- average reconstruction time;
- amount of fragments transferred (upload and download);
- average redundancy level of blocks;
- number of blocks at each redundancy level;
- number of failed disks;
- average and std. deviation of disk's occupancy;
- average and std. deviation of disk's lifetime.

Note that some of them are kept for the sake of verification (e.g., disk's lifetime). After the simulation finished, these traces are aggregated and summarized. For each metric we keep the average, std. deviation, minimum and maximum of its values.

Furthermore, for every finished reconstruction, a more detailed information is kept in a separated trace file: e.g., the respective block id, the finish cycle, the elapsed reconstruction time, the number of re-starts, the number of fragments reconstructed.

3.4 Average System Behavior

In this section we discuss the performance metrics for different system characteristics. We compare the average behavior of the system given by the Simulation Model (SM) with the one predicted by the MCM and its approximations.

Simulation scenarios. A large number of simulations with different sets of parameters were performed. We first give an example of a medium size network composed by $N = 4000$ peers (different scenarios with N spanning from 50 to 1 million of peers are also evaluated); Let us choose $s = 8$, $r = 6$ and $r_0 = 3$ (we discuss about these choices in the next section) and a fragment size $L_f = 512$ KB. We obtain a block size $L_b = s \cdot L_f = 4$ MB and a system-wide number of blocks $B = 8 \cdot 10^5$ (i.e., the space overhead $(s+r)/s = 1.75$, hence a total of total 5.6 TB). The initial amount of data per disk is 1.4 GB (at the steady state it is 1.2 GB)[2]. Disk capacity is chosen to be 8 times the average amount of data per disk, i.e., 10GB.

The average time to reconstruct a block is $\theta = 6$ hours. It includes the timeout delay to detect that a peer has disappeared (temporary churn) and the delay to perform the reconstruction, i.e., the time to collect the remaining fragments, to recalculate the erasure code and redistribute the missing fragments. The average lifetime of a disk or Mean Time To Failures (MTTF) is assumed to be 1 year (see e.g. [91, 109] for a discussion). This value is less than a typical time-span of warranties applied by major hardware vendors, which is 3 years. Indeed, this value is a conservative choice and comprises the probability of other hardware failures and of software maintenance. In general, the simulation time T_{sim} was chosen to be 10 years, with a time step of one hour, which leads to 87600 cycles.

For such parameters, the average bandwidth usage per peer $BW_{avg} \approx 1.23$ Kbit/s (total of 4.93 Mbit/s). When a peer fails, the total amount of data to be transferred $Q_{peek} \approx 1.28$ MB per peer (total of 5.12 GB). For a provisioned reconstruction time $\theta = 6$ hours the $LossRate \approx 3 \cdot 10^{-3}$ per year.

3.4.1 Discussion

In the following, we discuss the behavior of the system under different scenarios. Table 3.2 presents the average and standard deviation of bandwidth consumption, and Table 3.3 presents the data loss rate per year for a representative subset of our experiments. In all the studied scenarios only the evaluated parameter is changed, while the others remain constant.

We observe that the MCM and the approximated equations (Equation (3.3) page 44 and Equation (3.4) page 45) give a very precise estimation of the system metrics, except for some extreme values. For example, the loss rate when r_0 is close to r differ from 30%. The reason is that, for high values of r_0, the probability to lose a block becomes very small, and these results are an average over rare events which is difficult to obtain by simulations.

Remark on data loss. In practice the system parameters are set in a way that the probability of a data loss is very low (e.g., in the order of 10^{-10}). However, it is difficult to simulate such rare events in a reasonable time. To solve this issue, in the evaluation of the loss

[2]To be able to execute the simulations in a reasonable amount of time, we choose a system with disk size 100 times smaller than the one expected in practice As a matter of fact, to simulate 4000 peers with small disks of size 5GB, the simulator needs to deal with 40 millions of fragments. Hence, the importance to propose scalable analytical models that can accurately estimate the behavior of very large systems.

rate (Table 3.3) we set the MTTF of disks as low as 90 days and kept a reconstruction threshold of only $r_0 = 2$.

Reconstruction Threshold (r_0)

We first evaluate a scenario with fixed parameters and different values of the reconstruction threshold (from $1 \leq r_0 \leq 5$). As shown in Table 3.2a, the amount of bandwidth consumption increases very rapidly when r_0 is close to r, as stated by the Equation (3.3) in page 43. However, as shown in Table 3.3a, when increasing r_0 the probability to experience a failure decreases exponentially. The parameter r_0 is discussed more deeply in the next section.

System Size (B and N)

To evaluate the behavior of different system sizes, we divided our experiments in three different scenarios. The first is a system with varying amount of total data (absolute number of blocks B), while keeping the number of peers N constant. Thus, the amount of data per peer increases with B. Tables 3.2b and 3.3b show that the bandwidth consumption and the loss rate increase linearly with the amount of data in the system, as estimated by the Equation (3.3) of BW_{avg} per peer and Equation (3.4) of $LossRate$.

The second scenario, is a system with fixed amount of data but varying the number of peers (from 100 to 10^6). In this case the amount of data per peer decreases as N increases. We confirm by simulations that the total average bandwidth consumption remains the same when the number of peers increase (see Table 3.2c). Hence, the amount of bandwidth consumed *per peer* is reduced as N increase, which confirms the Equation (3.3). Note that, as soon as the absolute number of blocks B remains constant, the data loss rate does not change when increasing N (see Table 3.3c). Furthermore, we highlight that increasing N the standard variation of the bandwidth consumption decreases. In other words, for the same amount of data, a larger system behaves much more smoothly. This behavior is discussed in detail in Chapter 4.

Another possible scenario is varying the amount of data and the number of peers at the same rate. That is, the amount of data stored per peer $b = \frac{B}{N}$ remains constant. Then, we remark that BW_{avg} per peer does not change. But, in this case $LossRate$ increases when the system is larger (it depends on the absolute number of blocks in the system, B, we discuss the choice of this parameter in the next section).

Block Reconstruction Time ($\theta = 1/\gamma$)

When the reconstruction time θ is increased, the results obtained by simulations and by MCM show that it affects very slightly the average bandwidth consumption (see Table 3.2d). In the approximations, as we assumed a ratio $\alpha/\gamma << 1$, this small factor is neglected. Note that when increasing θ, data loss rate increases exponentially (see Table 3.3d).

Peer Lifetime ($MTTF = 1/\alpha$)

Increasing the values of the peer lifetime $MTTF$, affects the bandwidth consumption almost linearly, as shown in Table 3.2e. As expected, the loss rate decreases exponentially when α decrease (see Table 3.3e).

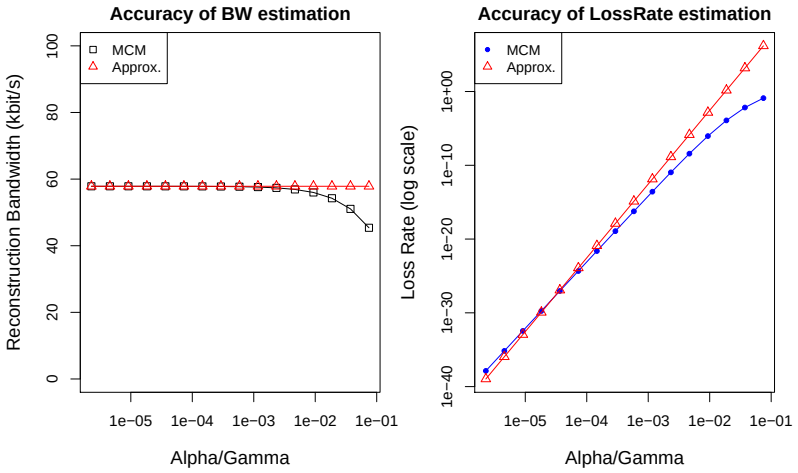


Figure 3.5: Accuracy of estimations for different ratios α/γ.

3.4.2 Validation of Approximation

Figure 3.5 shows the accuracy of Equations (3.3) and (3.4) compared to the MCM for different ratios of α/γ. Note that for values of $\alpha/\gamma < 10^{-3}$ the results obtained by the equations are very close to the MCM. For such values of α/γ our experimentation with different parameters confirmed the accuracy of the equations.

3.5 How to Set the System Parameters

The choice of the system parameters depends on multiple constraints, for instance, the storage space-overhead, the bandwidth consumption, the desired level of reliability, etc. In this section, we propose a methodology to choose the main system parameters B, L_f, s, r, r_0, for a desired reliability (probability of data loss) or a given limit on the bandwidth consumption.

In brief, we start defining a suitable value for s and L_f, which depends on the system architecture and usage. Then, if the space-overhead ($\frac{s+r}{s}$) is fixed, we choose the best r_0 for the given constraints. Otherwise, we first choose r_0 and then calculate the best value of r to minimize the bandwidth consumption.

Table 3.2: Average and standard deviation of *total* bandwidth usage (in Mbits/s) obtained using the SM, MCM and approximations, for different values of r_0, B, N, θ and $MTTF$.

(a) Reconstruction threshold (r_0)

r_0	1	2	3	4	5
SM	3.22 ±2.2	3.85 ±2.6	4.90 ±3.2	6.95 ±4.4	12.94 ±8.1
MCM	3.19	3.86	4.92	6.97	12.98
Approx	3.19	3.86	4.93	7.00	13.09

(b) Total amount of data (B x 10^3)

B	400	600	800	1000	1200
SM	2.49 ±1.6	3.69 ±2.4	4.90 ±3.2	6.10 ±4.0	7.38 ±4.8
MCM	2.46	3.69	4.92	6.15	7.38
Approx	2.47	3.70	4.93	6.16	7.39

(c) Number of peers (N)

N	100	1000	10000	100000	1000000
SM	5.05 ±19.45	4.90 ±6.3	4.92 ±2.1	4.92 ±0.6	4.92 ±0.2
MCM	4.92	4.92	4.92	4.92	4.92
Approx	4.93	4.93	4.93	4.93	4.93

(d) Reconstruction time ($\theta = 1/\gamma$, in hours)

θ	1	6	12	18	24
SM	4.92 ±10.7	4.90 ±3.2	4.88 ±2.2	4.87 ±1.8	4.86 ±1.5
MCM	4.93	4.92	4.91	4.89	4.88
Approx	4.93	4.93	4.93	4.93	4.93

(e) Peer failure rates ($MTTF = 1/\alpha$, in years)

$MTTF$	1	2	3	4	5
SM	4.90 ±3.2	2.49 ±2.2	1.67 ±1.8	1.25 ±1.5	1.00 ±1.3
MCM	4.92	2.46	1.64	1.23	0.99
Approx	4.93	2.47	1.64	1.23	0.99

Table 3.3: Data loss rate per year (number of blocks) obtained using the SM and MCM, for different values of r_0, B, θ and $MTTF$. See the remark on data loss.

(a) Reconstruction threshold (r_0)

r_0	1	2	3	4
SM	$3.4 \cdot 10^3 \pm 80$	$1.1 \cdot 10^2 \pm 11$	4.5 ± 2.1	$2.5 \cdot 10^{-1} \pm 0.5$
MCM	$3.4 \cdot 10^3$	$1.1 \cdot 10^2$	4.2	$1.9 \cdot 10^{-1}$
Approx	$2.6 \cdot 10^3$	$0.9 \cdot 10^2$	3.3	$1.1 \cdot 10^{-1}$

(b) Total amount of data (B x 10^3)

B	400	800	1200	1600
SM	2.2 ± 1.5	4.5 ± 2.1	6.1 ± 2.5	8.0 ± 2.9
MCM	2.1	4.2	6.2	8.3
Approx	1.7	3.3	5.0	6.6

(c) Number of Peers (N)

N	100	1000	10000	100000	1000000
SM	3.9 ± 9.1	4.1 ± 2.0	4.3 ± 1.9	4.3 ± 2.0	4.3 ± 2.0
MCM	4.2	4.2	4.2	4.2	4.2
Approx	3.3	3.3	3.3	3.3	3.3

(d) Reconstruction time ($\theta = 1/\gamma$, in hours)

θ	1	6	12	18	24
SM	0	4.5 ± 2.1	$7.1 \cdot 10^1 \pm 8.5$	$3.5 \cdot 10^2 \pm 19.2$	$1.0 \cdot 10^3 \pm 34.2$
MCM	$4.9 \cdot 10^{-3}$	4.2	$7.1 \cdot 10^1$	$3.4 \cdot 10^2$	$1.0 \cdot 10^3$
Approx	$5.0 \cdot 10^{-3}$	3.3	$10.5 \cdot 10^1$	$8.4 \cdot 10^2$	$3.3 \cdot 10^3$

(e) Peer failure rates ($MTTF = 1/\alpha$, in days)

$MTTF$	30	60	90	120	180
SM	$8.2 \cdot 10^2 \pm 29.7$	$3.0 \cdot 10^1 \pm 4.2$	4.5 ± 2.1	0.9 ± 0.9	0.3 ± 0.5
MCM	$8.2 \cdot 10^2$	$3.0 \cdot 10^1$	4.24	1.0	0.1
Approx	$7.0 \cdot 10^2$	$2.5 \cdot 10^1$	3.3	0.9	0.1

Parameters

In the following experiments, we assume a network composed by $N = 500$ peers and $L_{total} = 20$ TB of data to be stored. Let us choose $s = 16$, $r = 16$ and $r_0 = 8$ (we discuss about this choice in the next section) and a fragment size $L_f = 320$ KB. We obtain a block size $L_b = s \cdot L_f = 5$ MB and the total number of blocks $B = L_{total}/L_b = 2^{22}$. The initial amount of data per disk is 82 GB (at the steady state it is 72 GB). For such parameters, the average bandwidth usage per peer $BW_{avg} \approx 57.8$ kbps. When a peer fails, the total amount of data to be transferred $Q_{peek} \approx 246$ GB (504 MB per peer). For a provisioned reconstruction time $\theta = 12$ hours the $LossRate \approx 5.7 \cdot 10^{-8}$ per year.

3.5.1 Determining the Block Size (L_b)

The total number of blocks in the system is defined by $B = L_{total}/L_b$. For a given amount of data L_{total}, how do we choose L_b? Similarly, knowing that $L_b = s \cdot L_f$, how do we choose s and L_f?

In this discussion we assume that r and r_0 are defined as a factor of s, that is, $r = k \cdot s$ and $r_0 = k_0 \cdot s$. Hence, increasing s means increasing r and r_0 proportionally. By rewriting r and r_0 in the Equation (3.3) in page 3.3, we note that BW_{avg} does not almost depend on the ratios between B, s and L_f, but mainly on the constant L_{total}. Hence, the choice of the block size is based only on the $LossRate$ equation.and the system's usage and architecture.

Architectural issues: from a theoretical point of view, to obtain lower values of $LossRate$, B should be as small as possible, and therefore L_b as big as possible.

However, in practice we can deduce a lower bound for B based on R_{start}, the number of blocks that start the reconstruction when a peer fails. To balance the load among peers, every peer should process the reconstruction of at least one block, thus $R_{start} \geq N$. Let us take the eager scenario ($r_0 = r - 1$) for the sake of exposition, then R_{start} evaluates as $R_{start} \approx B(s + r)/N$. Then we have $B(s + r)/N \geq N$, which gives $B \geq N^2/(s + r)$ as lower bound of B.

The choice of L_b also depends on the main usage of the storage system. Two main groups of usage can be distinguished. For an *archival usage*, in which the access to the stored data is very rare, the block size L_b could be very large. Conversely, in a *file-system usage* that supports continuous read and write operations, it is interesting to have a very small block size, such that the overhead of accessing and modifying a block is low.

The choice of s: For a fixed $L_b = s \cdot L_f$, s should be as large as possible and L_f as small as possible. Figure 3.6 shows the bandwidth consumption and probability of data loss (in log scale) when using the MCM, for a system with fixed B, increasing s from 4 to 64, and proportionally decreasing L_f. In this experiment, $k = 1$ and $k_o = 1/2$. As expected, the results show that larger values of s do not impact the bandwidth consumption, whereas the probability of data loss decreases exponentially, as stated in Equation (3.4).

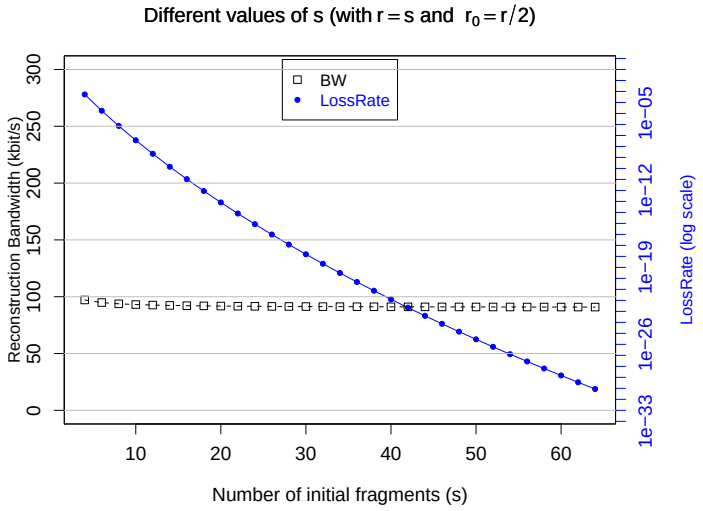


Figure 3.6: System with fixed number of blocks B, increasing s and decreasing L_f. The redundancy r and reconstruction threshold r_0 are chosen as a factor of s, respectively, $r = s$ and $r_0 = s/2$.

But note that the size of L_f should not be too small. Some practical limits (e.g., hard-disk sector size, etc.) impose a value of at least 4 KB, which is also the common value of file system's block size. Moreover, the amount of metadata, m_f, that should be kept is linearly dependent on the number of fragments of a block. In a practical system $m_f \ll L_f$, if not, the metadata takes up an important space that could be used to achieve more redundancy.

Another factor on the choice of s is the encoding and decoding rate of the erasure codes. Some implementations of the classic Reed-Solomon use words of length *one byte* to improve the efficiency (they work on the Galois Field $GF(2^8)$), which leads to the practical limitation $s + r \leq 256$. The overhead of the encoding is of order $O(s \cdot r)$. Very high values of s could impact negatively the encoding throughput. For instance, when s and r are large (e.g., $s = r = 128$) the encode throughput could be as low as 20 Mbps, on a Core 2 Duo 2.16Ghz, Otherwise, in our experiments we achieved a throughput of 140 Mbps for values of $s + r = 32$, and 600 Mbps for values of $s + r = 16$. We measured the *in memory encoding* using the Zfec library [126] (see [94] for more information).

3.5.2 Determining the Reconstruction Threshold (r_0)

For a given s, the choice of the threshold value r_0 depends on two factors: the desired reliability and the bandwidth capacity. The reliability can be calculated using Equation (3.4). It is sufficient to find the smallest r_0 that matches the desired *LossRate*. If r is not chosen yet, then it can be replaced with $r = r_0 + 1$, and the choice of r_0 is conservative.

Figure 3.7 shows the trade-off between the bandwidth consumption and the data loss rate (in log scale). In this experiment the space-overhead $\frac{(s+r)}{s}$ is fixed to 2, this means

$r = s$. Increasing r_0 means more reliability (*LossRate* decreases exponentially) at the cost of more bandwidth consumption. Note that the bandwidth consumption increases very fast when r_0 is close to r. For example, to provision a system to have $LossRate < 10^{-20}$ (20 nines of reliability, which is more than many RAID and NAS systems), and peer's $MTTF$ of 1 year, we find the value $r_0 = 10$ using the Equation (3.4). Then, the bandwidth consumption comes directly from Equation (3.3).

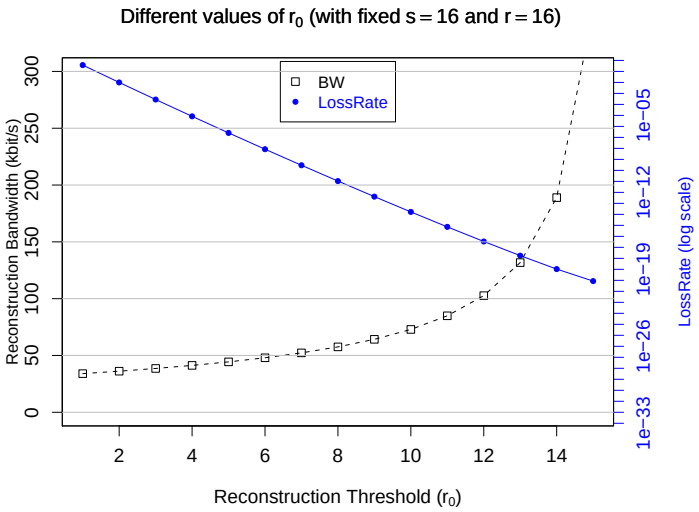


Figure 3.7: System with fixed space-overhead of 2. The parameters $s = r = 16$. The choice of r_0 depends on the desired reliability (prob. of data loss) and the bandwidth consumption.

3.5.3 Determining the Redundancy (r)

When s and r are defined, provisioning the system is easy and relies on the choice of the best r_0 that matches the resource constraints. However this is not always the case. If the space-overhead is not a problem, the parameter r can be chosen in such way that the bandwidth consumption is optimal. Figure 3.8 shows an experiment with fixed $s = 16$ and $r_0 = 6$, and increasing values of r. The results show that higher values of r decrease slightly the *LossRate*, however, the BW_{avg} follows a parabolic shape, for low values of r (extreme case $r = r_0 + 1$, the eager policy) the bandwidth consumption is very high, then it decays very fast. At a certain point the bandwidth consumption starts to grow with r.

Intuitively, we increase the value of r to delay the repair process because the overhead of the blocks' reconstruction. Mainly, we aim at reducing the fraction of blocks at the last Non-Critical state ($s + r_0 + 1$). This strategy has a strong effect when r is close to r_0 (see Equation (3.1)) but it decreases slowly when $r - r_0$ is large. However, at a certain point, the cost of having more fragments outweigh the gains of reducing the fraction of blocks at the state $s + r_0 + 1$.

To obtain the best bandwidth consumption for a given s and r_0, it is sufficient to find

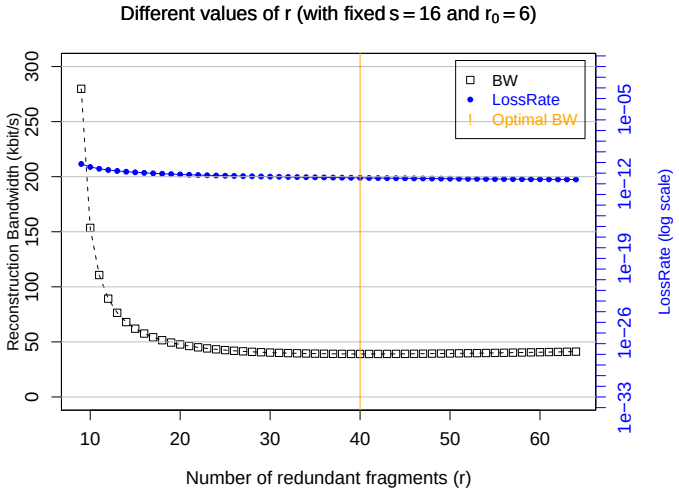


Figure 3.8: System with fixed values of s and r_0, and increasing values of r.

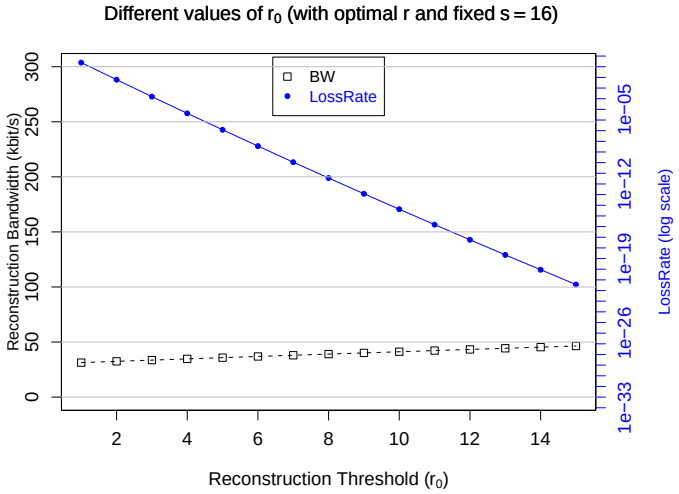


Figure 3.9: System with fixed s and increasing values of r_0. The value r is defined by the optimal bandwidth utilisation.

the derivative of Equation (3.3) with respect to r. The best r is given by $\frac{\partial BW_{avg}}{\partial r} = 0$, which evaluates to the following equation:

$$r_0 - s - r + (s + r) \cdot \ln\left(\frac{s + r}{s + r_0}\right) = 0. \tag{3.5}$$

The term r can then be isolated numerically (using the Maple software, for example), but it does not have a nice readable form:

$$r = s \cdot (e^{W(-\frac{r_0}{(s+r_0)\cdot e})+1} - 1) + r_0 \cdot e^{W(-\frac{r_0}{(s+r_0)\cdot e})+1}, \tag{3.6}$$

where W is the Lambert W function [32].

In the given example, for $s = 16$ and $r_0 = 8$, the optimal value of r equals 40 (space-overhead of 3.5). In this case, the average bandwidth consumption is 39.1 Kbps per peer, instead of 57.8 Kbps when the space-overhead is 2. Figure 3.9 shows a system with increasing values of r_0, and r chosen according to Equation (3.5). The bandwidth consumption is optimal for those values of s and r_0. Note that the bandwidth consumption increases very slowly, while the *LossRate* decreases exponentially.

3.6 Different Distributions of Reconstruction Times

In the previous sections, we presented models for a geometric distributed reconstruction time. This was mainly, as briefly mentioned, for the clarity of the presentation (smaller and simpler models) and because of the possibility to derive closed-form formulas in this case. In fact, our method is more general and can be adapted for large families of distributions. Unfortunately, in some ill-behaved cases, the models are too large (too many states) and the computations are not possible. Nevertheless, we show that, by reducing these large chains to *semi-markovian processes* with only three states, we are able to compute at least the average behavior even for complex continuous or discrete reconstruction time distributions.

Discussion about Real Distribution Times. Several models of the reconstruction time have been introduced or can be considered depending on the system implementation specifics (time to detect peer failures, periodic control time, scheduling of the reconstruction) and on the characteristics of the network (high available bandwidth, latency, losses, etc.). All these different parameters affect the block reconstruction time. To cite a few: the reconstruction time can be constant, exponential (or its discrete counterpart, geometric).

3.6.1 Modeling a Constant Reconstruction Time

In systems having a large detection time of failures (let us say many hours, or some days), the reconstruction time can be approximated by a constant time. This is possible because the detection time is much larger than the time to reconstruct a block (or a failed disk).

These systems can also be modeled by a Markov Chain, as shown in Figure 3.10.
States: each reconstruction "state" is now modeled by a path of length θ (reconstruction time, in time steps). When in reconstruction, at each time step, a block advances by one step. The chain has $\theta \cdot (r_0 + 1) + r - r_0 + 1$ states.
Transitions: the transitions are from (j, i) to $(j, i+1)$, with $0 \leq j \leq r_0$ and $0 \leq i \leq \theta - 2$. At the end of the path (i.e., end of the reconstruction), we have a transition from the state $(j, \theta - 1), 0 \leq j \leq r_0$, to the state of full redundancy r.

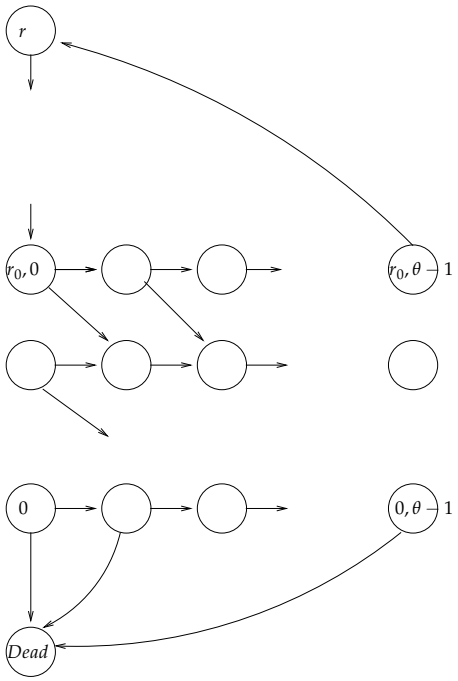


Figure 3.10: Modeling a system with constant reconstruction time.

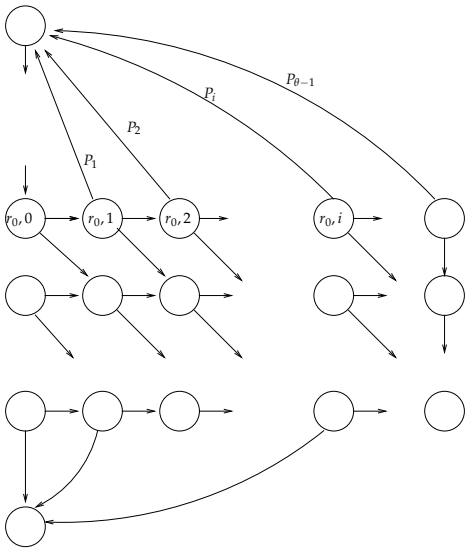


Figure 3.11: Modeling a system with a general reconstruction time distribution.

3.6.2 Modeling More General Distributions

More generally, any discrete time distribution can be modeled by a Markov chain, see Figure 3.11. We note p_j the probability to be reconstructed in time j, $j \in \mathbb{N}$. Similarly to the previous model for deterministic time, each reconstruction "state" is modeled by a path but of length $\max\{j; p_j > 0\}$. Note that the length is potentially infinite. In the previous model, we had one transition from the states $(i, \theta - 1)$, $0 \leq i \leq r_0$ towards the state with full redundancy (r). Now, we have one transition per state of the path, (j, i), to the state r taken probability p_j (if $p_j = 0$, we consider that there is no transition).

Note that, in case of an infinite chain, states corresponding to very long reconstruction times will practically never be attained, either because: 1) the block will die before this event; 2) in practical system, infinite reconstruction time has no meaning. Hence, these infinite chains can be closely approximated by truncating the distribution, leading to a finite chain.

Note that a continuous distribution can be approximated by a discrete Markov Chain with a small enough time step.

Discussion on the Number of States

As soon as the reconstruction is not poissonian, the probability to be reconstructed depends on the time of the past when the reconstruction started. Hence, the reconstruction time θ has to be hard coded in the chain states. The number of states thus depends lin-

early on the time granularity of the chain. If the time step τ is small (in comparison to θ), the number of states can be big. There is some flexibility on the choice of the time step, but unfortunately there are some limits: the time step cannot be chosen close to the average reconstruction time ($\gamma << 1$ is better), as it would lead to errors in the estimation of bandwidth usage and of the number of dead if at the same time αN is large (αN close to one or larger). For most systems, it is not a problem, as the computation of eigenvectors is polynomial in the number of states of the system. But nevertheless, for some ill-behaved systems, the number of states could be probibitively high. Fortunately this difficulty can be overcome with the use of semi-markovian processes.

3.6.3 Semi-Markovien Processes

A semi-markovian process (see for example [106]) is a stochastic process with states indexed by $S \subseteq N$. When it enters state i, it stays there for a random time having mean μ_i and then makes a transition to state j with probability P_{ij}. This random time can depend of j and is chosen according to a distribution F_{ij}. Note that when the time spent in a node is always one, the process is just a discrete Markov chain. Note that a semi-markovian process is not markovian in the sense that it does not satisfy the Markov property (indepandance from the past) that, if $t > \tau_1 > \tau_2$,

$$P[X(t) = i | X(\tau_1) = j \text{ and } X(\tau_2) = k] = P[X(t) = i | X(\tau_1) = j],$$

where X_t is the state of the system at time t. The main result that we use here, is that we can compute the stationnary distribution of a semi-markovian process directly from the one of its underlying markov chain.

We define P_i as the proportion of time that the process is in state i. To calculate P_i, we consider π_i, the proportion of transitions that take the process into state i. If we note X_n the state of the system after the nth transition. $\{X_n; n \geq 0\}$ is a Markov chain with transition probabilities P_{ij}. Then π_i is just the stationary probability of this chain. We have the following theorem:

Theorem 1. *Under some conditions on the semi-markovian processus (see [106] p. 453), we have, for $i = 1, 2, \ldots N$,*

$$P_i = \frac{\pi_i \mu_i}{\sum_{j=1}^{N} \pi_j \mu_j}.$$

Construction of the Semi-Markovian Process

We here transform the Markov Chain with a large (possibly infinite) number of states into a semi-markovian process with three states, see Figure 3.12:

- a state *OK*, corresponding to a contraction of the states $r_0 + 1 \leq i \leq r$;
- a state *R*, corresponding to the states (i, j) with $0 \leq i \leq r_0$;
- and a last state, *Dead*, for the single state Dead of the Markov chain.

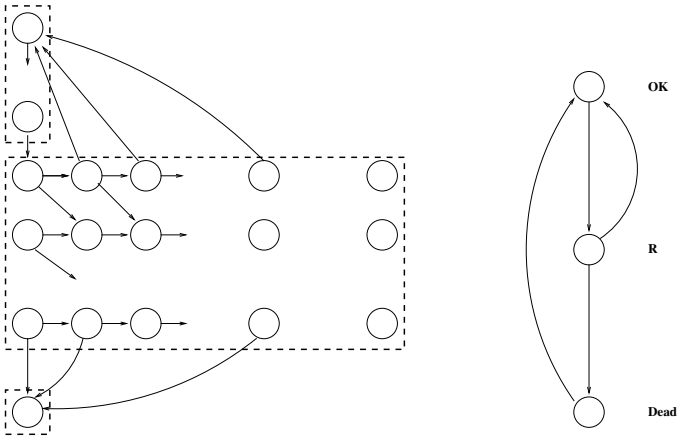


Figure 3.12: Transforming a large (possibly infinite) Markov chain into a semi-markovian process with three states.

The possible transitions are defined by the contraction (there is a transition between two states A and B of the semi-markov process if there exist two states of the original Markov chain, one contracted in A and the other in B, with a transition in the original chain). Note that we did not represent the transition from OK to Dead in Figure 3.12, as it is highly unikely in any practical system. A distribution of time spent in each of these states is associated with each possible transitions. These distributions are computed as follows[3] .

Small preliminary computations. Let us denote by $P_i(k,T)$ the probability to lose k fragments during a time of T time step when you are at state i (i.e., when you have $s+i$ remaining fragments). The probability for a peer to die during T time steps is $\beta(T) = 1-(1-\alpha)^T$. (In the case of *continuous distributions*, just take $\beta(T) = 1-\exp(-\alpha T)$ and replace the following sum over T with integrals.) Hence, we get

$$P_i(k,T) = \binom{s+i}{k}\beta(T)^k(1-\beta(T))^k.$$

Distribution of the transition R $\rightarrow$ Dead: the block takes the transition at time T if

- the reconstruction time is larger than T. It happens with probability $P[RT \geq T]$.
- r_0 fragments are lost between time 0 and time $T-1$. Probability $P_{r_0}(r_0, T-1)$, as we have $s+r_0$ fragments when the reconstruction starts.
- one fragment is lost at time T, happening with probability: $s\alpha$ as we have s remaining fragments before dying.

[3]Note that more direct formulas can be derived at the price of a small approximation/assumption. If we consider that the reconstruction suppose to last a time T lasts a time T even if the block cannot be reconstructed, the distribution corresponding to the transition from R to OK simply is the reconstruction time distribution.

Note that we negletected here the possibility of $k > 1$ multiple fragment losses during one time step, as there are basically of order $\alpha^k(s + r_0)$ and that α is small in the systems we analyze. Nevertheless, the more complete expressions can be easily derived by adding the probabilities to have k failures in time $T - 1$ and $r_0 - k + 1$ ones in the last time step. It gives

$$P_{R,Dead}(T) = P[RT \geq T]P_{r_0}(r_0, T-1)s\alpha.$$

From the distribution, we derive the probability to take the transition R $\rightarrow$ Dead which is only

$$P_{R,Dead} = \sum_T P_{R,Dead}(T).$$

It give the probability of the second transition from state R, as $P_{R,OK} = 1 - P_{R,Dead}$.
Distribution of the transition R $\rightarrow$ OK: the block takes the transition at time T if

- the reconstruction time is T (probability $P[RT = T]$).
- the block did not die before (probability $\sum_{T=1}^{T} P_{r_0}(r_0 + 1, T)$.

We have

$$P_{R,OK}(T) = P[RT = T] \sum_{T=1}^{T} P_{r_0}(r_0 + 1, T).$$

Distribution of the transition OK $\rightarrow$ R: The transition occurs as soon as $r - r_0$ fragments are lost, giving

$$P_{OK,R}(T) = P_{s+r}(r - r_0, T).$$

The semi-markov process is now defined. We then use Theorem 1 to compute its stationary distribution, from which can be extracted the average reconstruction time and the rate of block deaths.

3.7 Conclusion

In this chapter, we analyzed the steady-state of a peer-to-peer storage system based on traditional erasure codes and lazy repair. From a simplified Markov Chain Model we deduced close-form mathematical expressions to estimate the system behavior. The results were focused on the metrics: probability of losing data and bandwidth consumption. We described a methodology to determine the main system parameters, such as the number of initial fragments s, the reconstruction threshold r_0 and the space-overhead defined by $(s + r)/s$. We show that the lazy repair mechanism can be employed to achieve a better utilization of bandwidth for a given reliability, at the cost of additional space usage.

Furthermore, we highlighted that a simple Markov chain can be easily designed to capture the system average behavior, however the results obtained by simulation show an important variation around its mean that is not captured by simple Markov chains. Hence, in the next chapter we propose and analyze a fluid model that harnesses these variations.

CHAPTER 4

Capturing the Variations

In the previous chapter, we presented a Markov chain model that represents the behavior of a single data block. This chain allows us to compute the average behavior of the system accurately. Simulations confirm our analytical results, but also indicate that the deviations around the average behavior (i.e., the standard deviation) are much higher than those given by the Markov model.

These variations are explained by the fact that when a disk failure occurs (or a peer permanently leaves the system) many *data fragments are lost at the same time*. This correlation induces large peaks in the bandwidth consumption. In addition, when the bandwidth is limited, those peaks tend to slow down the repairing process, resulting in data loss. Indeed, when the repairing time is longer, a damaged block is more likely to lose its remaining redundancy fragments to a point where it cannot be repaired. The consequence is that a bandwidth provisioning decision not taking into account these variations would lead to an erroneous design which in turn would introduce a risk of losing a significant amount of data.

In this chapter, we propose and study a new stochastic model based on a fluid approximation that assesses the variations on the bandwidth consumption and the probability of data loss. In addition to its expectation, the fluid approximation gives a correct estimation of its variations. This new model is also validated by simulations.

The results presented in this chapter appeared in [3] (best student paper award) and were published in [4].

Our contribution

In order to take into account the variations around the mean, we propose a new stochastic Fluid Model, that does not represent a single block anymore, but the whole system. We provide a mathematical analysis of this model by giving a method to compute all the moments of its associated stationary distribution. Simulations show that the Fluid Model predicts the system very well (1% margin). Moreover, this model is scalable since its complexity is proportional to the erasure code length and does not depend on the number of peers.

To the best of our knowledge, this work is the first study to propose an analytical model that takes into account the correlations between data block failures. Along with failure correlation, we also point out the impact of disk age heterogeneity on the system, and propose a new shuffling policy and a biased reconstruction policy to reduce this impact.

Organization

The remainder of this chapter is organized as follows: in the following section we discuss the deficiencies of the proposed simple Markov chain model and highlight the importance of capturing the system variations. We then propose a new Fluid Model in Section 4.2 that better captures the system variations, followed with its analysis, validation and some avenues for future research. Finally, our concluding remarks are in Section 4.4.

4.1 Study of Correlation Effects

In this section, we point out the deficiencies of the simple Markov chain Model (MCM) to model the simultaneous loss of fragments when a disk fails. We show in Section 4.1.2 the significant impact of this correlation on the variations of bandwidth usage, even for a large system. In Section 4.1.3, we examine a provisioning scenario and show that, when not taken into account, this variation could lead to a very high loss rate.

4.1.1 The Problem of Correlation

As shown in the previous chapter, the system averages are estimated precisely by the MCM. However, as shown in Tables 3.2 (page 53) and 3.3 (page 54), the standard deviation (after the $\pm$ sign) obtained by simulations can not be captured by the Markov Chain Model (MCM).

Figure 4.1 shows an histogram with the distribution of the bandwidth consumption over time. In the top plot we have the results obtained using the Simulation Model (SM), and in the bottom a system equivalent to the MCM, with independent fragment failures. As previously stated, the average value of both systems are very close (4.92 versus 4.90 Mbits/s). However, the variations around this average are totally different. The standard deviation is 3.22 Mbits/s in the SM, to compare with only 0.10 Mbits/s in the MCM. This difference is explained by the fact that *a disk failure impacts simultaneously all the blocks that have fragments stored on it*. Therefore, when a failure happens, many blocks lose one fragment at the same time. Moreover, an important proportion of these blocks needs to start the reconstruction, which induces high peaks in the bandwidth consumption.

Note that the standard deviation of the independent model can be deduced directly from the MCM. Each block has a probability $p = \sum_{i=0}^{r_0} P(i)$ of being in reconstruction, with P the stationary distribution of the MCM. Hence, the total number of blocks in reconstruction is the sum of independent variables and follows a binomial distribution of parameters B and p. This distribution is very concentrated around its mean Bp and the standard deviation is given by $\sqrt{Bp(1-p)}$.

We conclude that modeling the behavior of a single block using the MCM and extrapolating the results to the whole system do not lead to an accurate representation of the system.

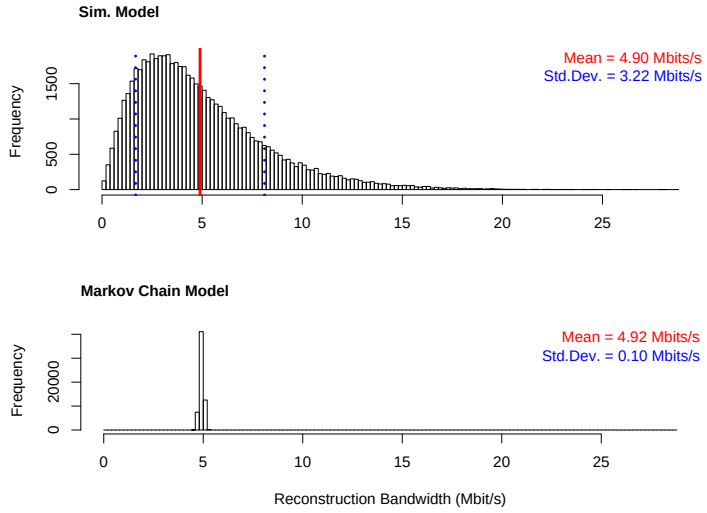


Figure 4.1: Histogram of the bandwidth used by reconstructions. Top: Simulation Model. Bottom: Markov Chain Model. (Notice that y-scales are not the same.)

4.1.2 Correlation and the System Size

The impact of data loss correlation shown above actually depends on the amount of fragments stored on the disk. A somewhat extreme case is when the number of peers is equal to the number of fragments of a block at full redundancy, that is $N = s + r$. In such a system all the blocks lose one fragment whenever a disk crashes and all the blocks follow the same trajectory. Almost at the opposite, when the disk contains few fragments (the extreme being each disk contains at most one fragment), trajectories do get independent and the system does not deviate from its mean. These two extreme examples illustrate the fact that the impact of correlation depends on the ratio between the number of fragments per disk and the number of peers (a peer failure simultaneously impacts about $\frac{B(s+r)}{N}$ fragments). In an extremely large system, the dynamic gets closer to the independent case. The following simulations confirm this intuition.

To illustrate it, we simulate systems with the same amount of data (number of blocks), but with varying number of peers (between 50 and 1 million) and varying number of fragments per disk. Figure 4.2 shows the average bandwidth consumption and standard deviation (represented by the lines) given by SM and MCM. These values can also be seen in Table 3.2 (page 53). The standard deviation of MCM is very far from the SM. This is obvious for small systems: 0.10 vs 19.45 for 100 peers. But this is true even for large systems: the deviation is still 5 times higher for a system with $100,000$ peers. The deviation of the dependent system decreases monotonically with the system size toward the limit obtained for the independent system. In this example, when the number of peers reaches 1 million, both standard deviations are of the same order. As expected, the standard deviation of the MCM is almost constant, as it depends only on the number of blocks which is constant here.

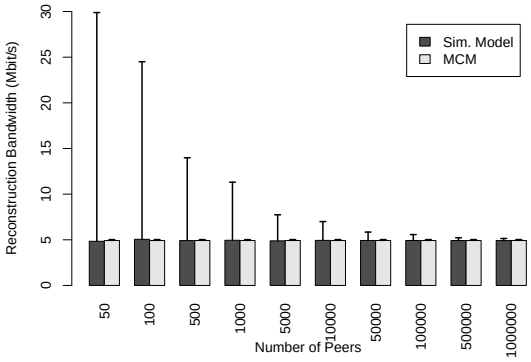


Figure 4.2: Avg. bandwidth usage and std. variation for different system size N. Same amount of data.

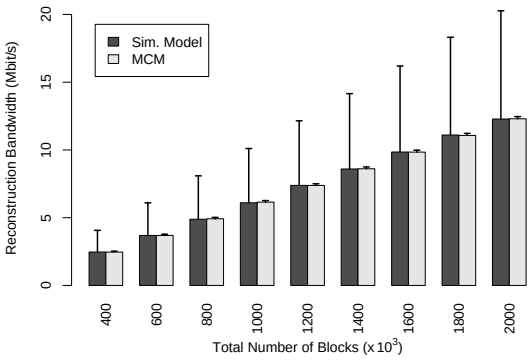


Figure 4.3: Avg. bandwidth usage and std. variation for different number of blocks B. Same number of peers.

Figure 4.3 shows the average bandwidth consumption and standard deviation for a system with fixed number of peers and increasing number of blocks. The results obtained by the SM and MCM grows linearly with B, however the standard deviations can not be estimated by the MCM.

4.1.3 Bandwidth Provisioning and Loss of Data

We show that the data loss correlation has a strong impact on the variations of the bandwidth usage. But do these variations really affect the system reliability? What happens if the amount of bandwidth available, or allowed by the user application, is limited? To answer these questions, we simulate different scenarios with bandwidth limitation. This limit varies from μ to $\mu + 10\sigma$, with μ and σ respectively the expectation and the standard deviation given by the MCM. In these experiments, when the bandwidth is not sufficient to carry out all the reconstruction demands, a queue is used to store the blocks to be rebuilt. The reconstructions then start in FIFO order when bandwidth is available.

Figure 4.4 shows the cumulative number of dead blocks for different limits of bandwidth. We see that limiting the bandwidth has very strong impact. Between μ (4.95 Mbits/s) and $\mu + 5\sigma$ (5.90 Mbits/s), the loss rate per year dropped from 11.2 blocks in the former to 0.12 block in the later.

If we have no limit on the bandwidth, the estimated number of dead blocks per year is $4.5 \cdot 10^{-3}$, which is respectively 2400 and 26 times less than the former cases.

Note that for all these experiments, the available bandwidth is greater than the average bandwidth given by the MCM. Hence, it is only *the fact of delaying some block reconstructions that increases the probability to lose fragments*. As a consequence, provisioning the system based on a model assuming block independence, as the MCM, could lead to disastrous effects. As a matter of fact, in the MCM, the bandwidth usage is very concentrated around its mean. For example, the probability to exceed $\mu + 5\sigma$ is less than

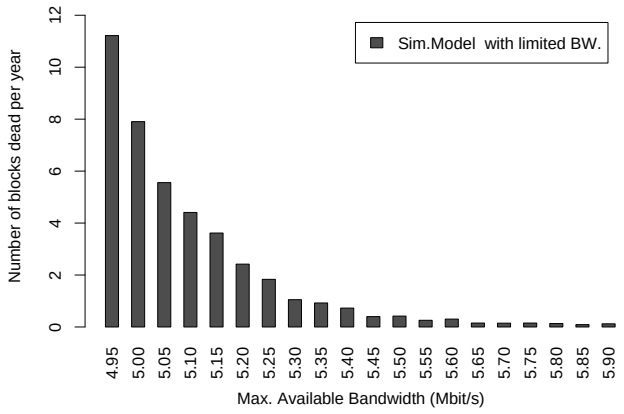


Figure 4.4: Data loss for different provisioning scenarios using the SM.

$5.8 \cdot 10^{-7}$. A provisioning of this amount of bandwidth seems a very safe one. But as we see in Figure 4.4, such a dimensioning would lead to data loss. Therefore, it is very important to have a model that takes data loss correlation into account.

4.2 A New Stochastic Model

The discussion above shows that the system cannot be seen as a set of independent blocks; so we need to model the system globally. For this purpose, we propose in this section a new approximated model based on a fluid approximation. We provide a theoretical analysis in Section 4.2.2, giving its average behavior, the variation from its mean and a way to compute any of its moments. In Section 4.2.3, we show by means of simulations that it models very closely the variations of a realistic system.

4.2.1 The New Model

We need to model the whole system. Block states could be fully described by a vector encoding the location of its fragments. This would lead to a gigantic Markov Chain (with around $N^{(s+r)B}$ states) which is too big to compute its stationary distribution. Therefore, we propose a new Markovian Approximated Model whose purpose is too approximate this gigantic chain.

The Approximated Model. The Approximated Model is derived from the following observation: fragments are spread randomly during the initialization phase and whenever a reconstruction occurs. Hence, we make the following approximation:

(A) At any time the fragments of a block are randomly placed into the system[1].

[1]Assumption (A) is indeed an approximation since the fragments of a block whose last reconstruction occurred at time T_0 can only be located on the disks that where in the system at time T_0 and never got faulty

In such a case, the state of a block is fully described by its level of redundancy and blocks at the same level are equivalent. Hence a Markov Chain that counts how many blocks are at each level can be used. The system is described by a vector $B(t) = (B_0(t), \cdots, B_r(t))$ where $B_i(t)$ is the number of blocks at level i at time t. This discrete chain can be formally described, but it is still too large for practical use (it has $(r+1)^B$ states). However since many blocks are in the same state, we use a fluid approximation for that chain (see [71] for references on fluid models).

Fluid Model Approximation for Large Systems (Fluid Model). The process to distribute the fragments among the disks follows a multinomial distribution during time (Assumption (A)). When the number of blocks B is large compared to N, as in practical systems, the multinomial distribution is very concentrated around its mean: the standard error of the number of fragments per disk is of order $O(\frac{1}{\sqrt{B/N}})$. The fluid approximation consists in neglecting these variations around the mean and considering that, at each time step, the proportion of blocks affected by the reconstructions and peer faults is exactly the average proportion.

We present here this stochastic Fluid Model, with discrete time step τ. The system is described by the state vector $X(t) = (X_r(t), \cdots, X_0(t))$, where $X_i(t)$ counts the fraction of blocks that are in state i at discrete time t (i.e., $X(t) = B(t)/B$). The evolution of the state vector is then modelled as follows. First, we define two matrices: $\mathbf{R}$, which represents the effects of the reconstruction process on the state vector,

$$\mathbf{R} = \begin{pmatrix} 1 & & & \gamma & \cdots & \gamma \\ & \ddots & & & & \\ & & 1 & & & \\ & & & 1-\gamma & & \\ & & & & \ddots & \\ & & & & & 1-\gamma \end{pmatrix}$$

and $\mathbf{F}'$, the effects of a disk failure,

$$\mathbf{F}'(t) = \begin{pmatrix} 1-\mu_r(t) & & & & & \mu_0(t) \\ \mu_r(t) & \ddots & & & & \\ & \ddots & \ddots & & & \\ & & \ddots & \ddots & & \\ & & & \mu_1(t) & 1-\mu_0(t) \end{pmatrix}$$

where $\mu_i(t)$ is the fraction of blocks in state i affected by a failure. We then express a transition of the system as

$$X(t+1) = M(t) \cdot X(t),$$

since. The correct statement is that the fragments of a block with age $T - T_0$ are randomly spread on disks with age at least $T - T_0$. Nevertheless we assume that (A) holds.

with $M(t)$ a random product defined as follows

$$M(t) = \begin{cases} \mathbf{R}\cdot\mathbf{F}' & \text{with prob. } f \text{ (disk failure);} \\ \mathbf{R} & \text{with prob. } 1-f \text{ (recons. only),} \end{cases}$$

where f is the probability to experience a disk failure during a time step. At each time step, if no disk failure occurs, we only account for the effects of reconstructions; otherwise the disk failure effect is added. Henceforth, we note $\mathbf{F} = \mathbf{R}\cdot\mathbf{F}'$ for simplicity.

The model makes the following assumptions:

- At most one disk can fail during a time step.
- The failure rate during a time step is $f = \alpha N$.
- Whenever there is a failure, a block at level i has probability $\mu_i(t)$ to lose a fragment. This is indeed hypothesis (A). A first approach is then to consider that each disk contains a proportion $1/N$ of fragments (i.e., about $B(s+r)/N$), then the probability to lose a fragment at level i (assuming a fault) is $\mu_i(t) = \frac{s+i}{N}$. It corresponds to a first Simple Fluid Model (SFM).

Our first simulation experiments showed that this approximation already gives good results, but we can still refine it further as follows.

Disk age and number of fragments in a disk. When a disk fails, it is replaced by a new *empty* disk. Since disks fill up during the system life, a newly replaced disk is empty, while an old disk contains many fragments. Disk age and disk size distributions can be approximated closely for systems with large number of blocks. When a block is reconstructed, each of the rebuilt fragment is sent on a random peer. Hence, at each time step, the distribution of the rebuilt fragments among the peers follows a multinomial distribution, with parameters the number of rebuilt fragments and $1/N$. As the multinomial distribution is very concentrated around its mean, the *filling up process can be approximated by a affine process of its age*, in which, at each time step, each disk gets in average the number of reconstructed fragments divided by the number of peers. That is, the speed that disks get filled is approximated as:

$$v \approx \alpha \frac{\bar{F}}{N},$$

where $\bar{F}/N$ is the average number of fragments per disk. At age K, a disk has approximately $v{\cdot}K$ fragments. Thus, it represents a factor $\omega \approx v{\cdot}K/(\bar{F}/N) = \alpha K$ of the average disk capacity.

The age of death follows a geometric law of parameter α, as at each time step a disk has a probability α to experience a failure. That is,

$$\Pr[\text{death age} = K] = (1-\alpha)^{K-1}\alpha.$$

Hence, *disks with very heterogeneous number of fragments* are present in the system. This

strong heterogeneity of the number of fragments per disk may have a significant influence on the variations of the system. As a matter of fact, when the system experiences a disk failure, we may lose a lot of fragments if the disk was almost full, but a lot less for a young disk. Therefore, we propose a refinement of the Simple Fluid Model to take these variations into account.

Figure 4.5 shows the cumulative distribution function (CDF) of the number of pieces per disk obtained using the SM. It is compared with the CDF of the geometric law of parameter α normalized by the average disk occupancy $\bar{F}/N = 2000$ fragments (in this example $\alpha = 1/(24 \times 365)$). We see that the distribution of disk occupancy is very close from what expected from the previous discussion.

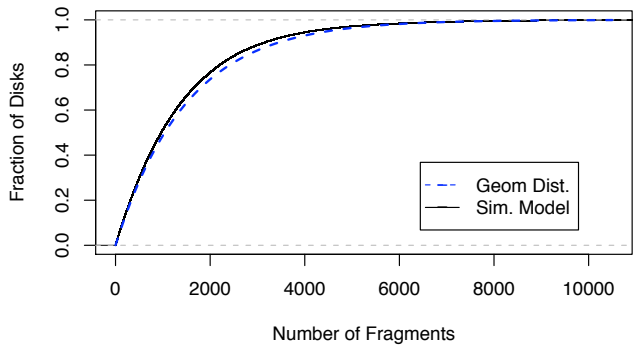


Figure 4.5: CDF of the number of fragments per disk in the system.

Remark. In this modeling, for the sake of exposition, we assume that disks have unlimited capacity (or equivalently, they die before they get filled up). This assumption makes the analysis more tractable. Indeed, in Chapter 5 we do take into account the limited capacity of disks. Which is done by modeling it by a truncated geometric distribution.

Refinement of the Fluid Model. We can take the disk size distribution into account and modify $\mu_i(t)$ accordingly. This can be done by setting

$$\mu_i(t) = \frac{(s+i)\omega(t)}{N},$$

where $\omega(t)$ is the *disk filling ratio*, $\omega(t) = \alpha K$, taken accordingly to the distribution of the number of fragments in a disk that have died with age K:

$$Pr[\omega(t) = \alpha K] = (1-\alpha)^{K-1}\alpha, \text{ for } K \geq 1.$$

where $\omega(t)$ follows a geometric distribution with $\mathbb{E}[\omega(t)] = 1$ (i.e., an average filling ration of 1), which indeed represents the average disk capacity $\bar{F}/N$.

Note that the model is scalable since its size is $s + r$ and the random transition matrix at time t can be computed in time $O((s+r)^2)$. Finally, let us summarize the new

notations that will be used throughout this section:

f	Probability to have a disk failure during a time step ($f = \alpha N$)
$\mathbf{R}$	Matrix that represents the effects of the reconstructions
$\mathbf{F}$	Matrix that represents the effects of a disk failure
μ_i	Probability of a block in state i to be affected by a failure
ω	Filling ratio of a failed disk

4.2.2 Analysis

We present a theoretical analysis that allows us to compute all the moments of the stationary distribution of the Fluid Model. The analysis boils down to the analysis of a random matrix (or matrix distribution), $M(t)$. Note that we do not give a closed formal solution to this difficult problem because there exists no general theory to get the distribution of a random product of two matrices. It is not surprising since, for example, only determining if the infinite product of two matrices is null is an undecidable problem [80].

Expression of the expectation of the Simple Fluid Model. A transition of the system transforms the state vector $X = (X_1, \cdots, X_n)$ according to

$$X(t+1) = M(t)X(t).$$

Hence,

$$\mathbb{E}[X(t+1)] = \mathbb{E}[M(t)]\mathbb{E}[X(t)].$$

The expectation of the transition matrix is given by

$$\mathbb{E}[M(t)] = \mathbb{E}[f\mathbf{F}(t) + (1-f)\mathbf{R}] = f\mathbb{E}[\mathbf{F}(t)] + (1-f)\mathbf{R},$$

with $\mathbb{E}[\mathbf{F}(t)] = \mathbb{E}[\mathbf{R}\mathbf{F}'(t)] = \mathbf{R}\mathbb{E}[\mathbf{F}'(t)]$, as $\mathbf{F}'$ is independent of $\mathbf{R}$. We have $\mathbb{E}[\mathbf{F}'(t)] = \mathbf{F}'$, with $\mathbf{F}'$ corresponding to the failure matrix for an average filling ratio of 1. Therefore, we obtain the same expectation for the Simple Fluid Model and the Fluid Model. To summarize, we get

$$\mathbb{E}[M(t)] = f\mathbf{R}\mathbf{F}' + (1-f)\mathbf{R}.$$

The linear operator $\mathbb{E}[M(t)]$ is a probability matrix and it can be computationally checked that 1 is the only eigenvalue with norm one. Hence we have $\mathbb{E}[X(t)]$ converges to E_0, solution of the equation

$$E_0 = (f\mathbf{R}\mathbf{F}' + (1-f)\mathbf{R})E_0.$$

Note that, since $(f\mathbf{R}\mathbf{F}' + (1-f)\mathbf{R})$ is roughly equivalent to the matrix transition of the MCM, we find that $\mathbb{E}[X(t)]$ converges to the stationary vector of the single block model. This is expected since expectations are linear.

Expression of the standard deviation of the Simple Fluid Model. We want to compute the standard deviation of the state vector X, meaning the standard deviation of each of its coordinates. We recall that each coordinate corresponds to the number of blocks in a given state.

Let start by computing $\mathbb{E}[X^2]$.

$$X(t+1)^2 = (M(t)X(t))^2.$$

That is

$$X_i^2 = \left(\sum_{j_1=1}^{n} m_{ij_1} X_{j_1}\right)\left(\sum_{j_2} m_{ij_2} X_{j_2}\right).$$

We get

$$X_i^2 = \sum_{j_1,j_2} m_{ij_1} m_{ij_2} X_{j_1} X_{j_2}.$$

Note that, as X^2 depends of all the cross-products of X_i and X_j, we have to compute all their expectations.

Expression of the expectations of the cross-products. We have

$$X_i X_j = \left(\sum_{k_1=1}^{n} m_{ik_1} X_{k_1}\right)\left(\sum_{k_2=1}^{n} m_{jk_2} X_{k_2}\right).$$

Hence

$$X_i X_j = \sum_{k_1,k_2} m_{ik_1} m_{jk_2} X_{k_1} X_{k_2}.$$

It gives for the expectations:

$$\mathbb{E}[X_i X_j] = \mathbb{E}[\sum_{k_1,k_2} m_{ik_1} m_{jk_2} X_{k_1} X_{k_2}].$$

By linearity and independence (of m_{ij} and X_i), we obtain

$$\mathbb{E}[X_i X_j] = \sum_{k_1,k_2} \mathbb{E}[m_{ik_1} m_{jk_2}]\mathbb{E}[X_{k_1} X_{k_2}].$$

The method is to write a linear system of equations linking the cross-product expectations at time $t+1$ with the expectations at time t. Let ind be the function $[1,n] \times [1,n] \to [1,n^2]$, $\mathrm{ind}(i,j) = (i-1)n + j$. Let us define the matrix N by

$$N_{i'j'} = \mathbb{E}[m_{i,k_1} m_{j,k_2}],$$

with $i' = \mathrm{ind}(i,j)$ and $j' = \mathrm{ind}(k_1,k_2)$. Note that this matrix is of dimensions $n^2 \times n^2$.

We now need to compute $\mathbb{E}[m_{i,k_1} m_{j,k_2}]$. As the matrix of transition $M(t)$ is stochastic, we have to sum over all possible disk fillings $\omega(t)$ to obtain the expectation. If, we note

$\mathbf{F}^{(k)}$ the matrix $\mathbf{F}(t)$ for a filling ratio equal of k, the definition of $M(t)$ gives

$$\mathbb{E}[m_{ik_1} m_{jk_2}] = \sum_{k=1}^{\infty} \Pr[\omega(t) = \alpha \cdot k] \left(f \mathbf{F}^{(k)}_{ik_1} \mathbf{F}^{(k)}_{jk_2} \right) + (1-f) \mathbf{R}_{ik_1} \mathbf{R}_{jk_2}.$$

$N_{i'j'}$ is then directly derived. Now, if we note Z the vector of the cross-products ($Z_{\text{ind}(i,j)} = X_i X_j$), we have

$$\mathbb{E}[Z(t+1)] = N(t)\mathbb{E}[Z(t)]$$

Again, as the linear operator $\mathbb{E}[Z(t)]$ is a probability matrix and because it can be checked that it has no eigenvalue with norm one other than 1, we have $\mathbb{E}[Z(t)]$ converges to E_0, solution of the equation

$$E_0 = N(t)E_0.$$

When Z is computed (by a resolution of a linear system with n^2 variables and equations), we can extract the coefficients $\mathbb{E}[X_i^2]$ and compute the standard deviations with

$$\sigma(X_i) = \sqrt{\mathbb{E}[X_i^2] - \mathbb{E}[X_i]^2}.$$

Conclusions for the number of reconstructions and the bandwidth. The fraction of blocks in reconstruction ξ is equal to the sum of the fraction of blocks in the states from 0 to r_0. We note $\xi = \sum_{i=0}^{r_0} X_i$. We have

$$\mathbb{E}[\xi] = \sum_{i=0}^{r_0} \mathbb{E}[X_i] \qquad \text{and} \qquad \mathbb{V}[\xi] = \sum_{i=0}^{r_0} \sum_{j=0}^{r_0} \text{cov}[X_i X_j].$$

The covariances can be extracted from the previous computations ($\text{cov}[X_i, X_j] = \mathbb{E}[X_i X_j] - \mathbb{E}[X_i]\mathbb{E}[X_j]$).

Each reconstruction lasts in average $\theta = 1/\gamma$, translated in the model by a probability γ to be reconstructed. Hence the expectation of the bandwidth BW used by the system during one time step is

$$\mathbb{E}[BW] = \gamma(s + r - r_0) L_f B \mathbb{E}[\xi].$$

Recall that L_f is the size of a fragment and $s + r - r_0$ is roughly the number of fragments sent during a reconstruction. We also get directly the variance

$$\mathbb{V}[BW] = (\gamma(s + r - r_0) L_f B)^2 \mathbb{V}[\xi].$$

Remark: Other moments can be computed similarly, albeit with additional complexity, as we need to compute all cross-products ($\mathbb{E}[X_1 \dots X_k]$ for the k-th moment).

4.2.3 Validation of the Model

We run an extensive set of simulations to validate the Fluid Model (FM) for different values of parameters. Figure 4.6 presents an example of a timeseries of the bandwidth usage. The top plot is the Simulation Model and the bottom one the Fluid Model. As

expected, the averages of the two models are almost the same (few tenths of percent). But in addition, we observe that the variations are now very close as well (3.22 Mbits/s vs. 3.14 Mbits/s).

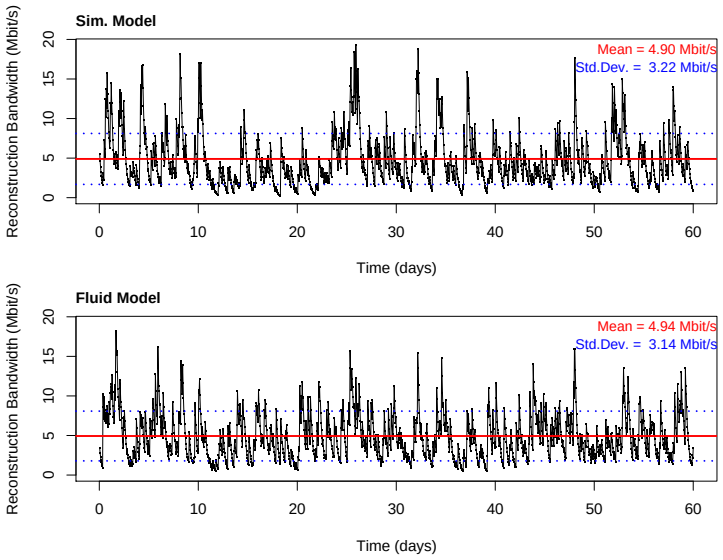


Figure 4.6: Timeseries of the bandwidth usage for SM and FM during 60 days at the steady-state.

Figure 4.7 shows the standard deviation of the bandwidth use in both models for systems with different number of peers and fixed amount of system data. Figure 4.8 shows the standard deviation of the bandwidth for increasing amount of data. We see that the values are very close and differ by only few percents. The average bandwidth use is about the same in all these experiments. Note that the variations of the FM and SFM are of the same order of magnitude, but still differ by around 20 to 40 percent in most cases, showing the impact of the heterogeneity of disk occupancy, and hence the need for the Fluid Model.

A summary of results is given in Table 4.1. We see that the relative standard deviation of the two models differs from less than 5 percent for this set of parameters. We conclude that the system is modeled very closely by the FM.

Influence of the parameters. Note that the standard deviation does not seem to depend of the value of r_0. To give an intuition of the influence of the parameters on the system variations, we provide here a rough estimate of relative standard deviation (Std.Dev/Mean) of the bandwidth usage.

When there is a disk failure, in average, roughly $R_{start} \approx \frac{B(s+r)}{N(r-r_0)}$ block reconstructions start. The average number of reconstructions can then be estimated by

$$\mathbb{E}[R] \approx f R_{start}.$$

Let us now estimate its variance. When f is small, there are two cases: either no

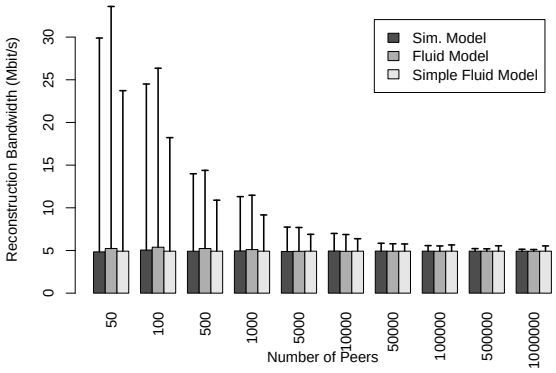


Figure 4.7: Bandwidth consumption vs number of peers N for SM, FM (SFM is also given for comparison).

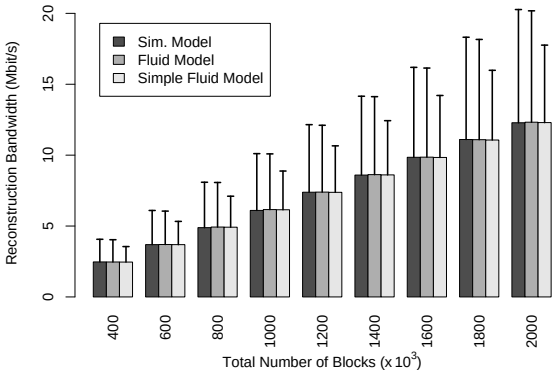


Figure 4.8: Bandwidth consumption vs number of blocks B for SM, FM (SFM is also given for comparison).

Table 4.1: Relative Standard Deviation of bandwidth usage (Std.Dev/Mean) for different values of r_0, B, N, θ and MTTF.

(a) Reconstruction threshold (r_0)

r_0	1	2	3	4	5
SM	0.69	0.67	0.65	0.63	0.62
FM	0.64	0.64	0.64	0.64	0.64

(b) Total amount of data (B x 10^3)

B	400	600	800	1000	1200
SM	0.65	0.65	0.65	0.65	0.65
FM	0.64	0.64	0.64	0.64	0.64

(c) Number of peers (N)

N	100	1000	10000	100000	1000000
SM	5.18	1.85	0.58	0.19	0.06
FM	5.42	1.75	0.57	0.18	0.06

(d) Reconstruction time ($\theta = 1/\gamma$, in hours)

θ	1	6	12	18	24
SM	2.17	0.66	0.45	0.37	0.28
FM	2.10	0.64	0.44	0.36	0.27

(e) Peer failure rates ($MTTF = 1/\alpha$, in years)

MTTF	1	2	3	4	5
SM	0.66	0.89	1.07	1.21	1.31
FM	0.64	0.89	1.08	1.25	1.41

failure occurs with probability $1-f$ and no reconstruction starts, or there is a failure and R_{start} blocks start the reconstruction. The reconstruction lasts θ time steps. Then the system reconstructs R_{start}/θ blocks per time step during a time θ. Hence it gives

$$\mathbb{V}[R] \approx (1-f\theta)E[R]^2 + f\theta\left(\frac{\mathbb{E}[R]}{f\theta} - E[R]\right)^2.$$

That is

$$\mathbb{V}[R] \approx (1-f\theta+f\theta(1/f\theta-1)^2)\mathbb{E}[X]^2.$$

When $f\theta$ is small, we get

$$\text{Rel.Std.Dev.}[R] \approx \frac{1}{\sqrt{\alpha N\theta}}.$$

From this approximation, the system variations should be roughly independent of r_0, but inversely proportional to $\sqrt{N}$, $\sqrt{\theta}$ and proportional to $\sqrt{MTTF}$. These tendencies are seen in Table 4.1 and Figures 4.7 and 4.8.

4.3 Convergence of the Fluid Model

We consider the system under assumption (A) (at any time the fragments of a block are randomly placed into the system). We prove here that the *trajectory of the fluid model remains close to the one of the system*. Close means here that the deviation is proportional to $\sqrt{B\log B}$. It implies that *both trajectories converge* when B grows to infinity. First, we note that we are not in the classic situation analyzed by Kurtz in [71]. As a matter of fact, we apply *two different operators* according to if there is a failure or not. The fluid model then also is no more deterministic, but random. Our proof is inspired by the work of Hennion on the products of positive random matrices [62]. Note that the result presented here can easily be generalized to a system with a larger number k of different events (here $k=2$). The only condition is that the random product of the k matrices should have a positive probability to be strictly positive after some time T. Finally, we observe the deviations and the speed of convergence in practice in Section 4.3.2.

4.3.1 Proof of Convergence

Sketch of the proof. The idea of the proof of convergence is the following.

- **Each transition adds a small noise** (vector with null sum, V). At each transition, the trajectory of the system is different from the one of the fluid model by a small noise of amplitude of order $\sqrt{B\log B}$.

- **This noise decreases with time.** As a matter of fact, there is a positive probability that the random product of matrices is a strictly positive matrice P after some time, with $P_{ij} > \varepsilon$. In this case, the norm $\sum_i |(PV)_i| < \sum_i (1-\varepsilon)|V_i|$ decreases.

- Hence the **difference between trajectories is bounded.**
- We deduce the **convergence** when the amount of data B groes to infinity.
- Let us also note that **the bound we get is very pessimistic.** As a matter of fact, the noises do not add upp. On the contrary, they cancel each other out as they have an expectation of 0 on each coordinate (thus, like a classic random walk the deviation is not linear but in $\sqrt{time}$, see for example [106]).

Proof

- **Each transition adds a small noise**.

  Let S be the system states. We note $W(t)$ the vector of dimension $|S| = r + 2$ coding the number of blocks in each state. B is the total population (total number of blocks). The variable $X(t) = W(t)/B$ represents the average statistical distribution of the population at time t.

  We note E the space of the contexts (here $E = \{\text{failure, no failure}\}$). Let $p(e)$ be the probability of the situation e. We define A_e the matrix associated to the situation e in the following way: the coefficient $A_e(i,j), (i,j) \in S^2$ denotes the probability that an individual in state i goes to state j in context e. The transition made by the individuals are considered independant, when the context e is given.

  In context e, the number of individuals taking transition $i \to j$ is the sum of Bernoulli variables having value 1 with probability $A_e(i,j)$. Its average is $A_e(i,j)W_i(t)$. Recall that the application of the Chernoff bound applied to n i.i.d. random variables Z_i of mean μ gives

  $$P[\sum_{i=1}^{n} Z_i \leq (1-\delta)\mu] \leq \exp(-\frac{\mu\delta^2}{2}).$$

  If we choose $\delta = \frac{\sqrt{2k\log(\mu)}}{\sqrt{\mu}}$, we have that $\sum_{i=1}^{n} Z_i \leq \mu - \sqrt{2k\mu\log(\mu)}$ with a probability smaller than $\frac{1}{\mu^k}$.

  Here, $\mu = A_e(i,j)W_i(t) \leq B$ and $n = X_i(t)B$. Hence, we have a deviation larger than $\sqrt{2kB\log(B)}$ with probability smaller than $\frac{1}{X_i(t)^k B^k}$.

  Hence, with probability $1 - \frac{1}{B}$,

  $$W(t+1) = A_e(t)W(t) + V(t),$$

  where $V(t)$ is a vector with sum zero and of of weak amplitude ($\sqrt{W(t)\log(W(t))}$).

- **This noise decreases with time.**

  There is a positive probability that the random product of matrices is a strictly positive matrice after some time T_0. As a matter of fact, if the system experiences $n = r + 2$ peer failures during period T_0, a block can go from any state to any state.

Fact 4.3.1. *Let be a vector $V = (V_1, \ldots, V_n)$ with null sum ($\sum_i V_i = 0$) and a strictly positive matrix of probability M ($\forall i, j, M_{ij} > 0$) with smallest coefficient $\underline{m}$. If we note $W = MV$, we have W is a vector with null sum and*

$$\sum_i |W_i| \leq (1 - \underline{m}) \sum_i |V_i|,$$

as some mass is transferred from the positive coefficients to the negative coefficients.

Hence, during time a noise vector is multiplied by strictly positive probability matrix and, thus, decreases. We can derive a crude bound of the time of convergence.

Consider now our random product of matrices over T_0 time step. The matrix M is R with probability $1 - \alpha N$ and F with probability αN. Now consider the number of matrices F over T_0 time step. In average, we have $\alpha N T_0$ of them. Using Chernoff bounds, we have that

$$\Pr[\#F < \frac{\alpha T_0}{2}] \leq e^{-\frac{\alpha T_0}{8}}.$$

By choosing $T_0 = 2\frac{n}{\alpha N}$, we get

$$\Pr[\#F < n] \leq e^{-\frac{n}{4}}.$$

Hence, the noise will disappear at the speed larger than $(1 - \varepsilon)^{1/\alpha N n}$.

- **The difference between trajectories is bounded.** We consider the trajectory of the system and the one of the fluid model subject to the same events. The deviation during a time step t is a vector $V(t)$ of norm at most $\sqrt{W(t)\log(W(t))}$. At time $t_1 = t_0 + \Delta t$, we have

$$W(t_1) = NX(t_1) + \sum_{t=1}^{\Delta t} (\prod_{\theta=t}^{\Delta t} A_{e(t_0+\theta)}) \delta(t_0 + t),$$

where $X(t_1) = \prod_{\theta=1}^{\Delta t} A_{e(t_0+\theta)} X(t_0)$ is the trajectory of the fluid model.

We have seen that after a period $T_0 = 2n/\alpha N$, a noise decreases by a factor at least $1 - \varepsilon$ with probability close to 1. This implies that the above sum is less than $\sum_i = 0, 1, \ldots (1\varepsilon)^{\lfloor i/T_0 \rfloor} = \frac{T}{\varepsilon}$ times the norm of a single deviation. So,

$$\begin{aligned} W(t_1) - NX(t_1) \leq \quad & T_0(1-\varepsilon)^{\frac{\delta t}{T_0}} |W(t_0) - NX(t_0)| \\ & + \frac{T_0\sqrt{W(t)\log(W(t))}}{\varepsilon}. \end{aligned}$$

Thus, the two trajectories are linked. Moreover, if $W(t_0) = NX(t_0)$, the two trajectories remain at distance $\frac{T_0\sqrt{W(t)\log(W(t))}}{\varepsilon}$.

- When B groes to infinity, the trajectory of the fluid model converges to the trajectory of the system. As a matter of fact, the componant of the state vector grow linearly with B, when the deviation grows in $\sqrt{B}$.

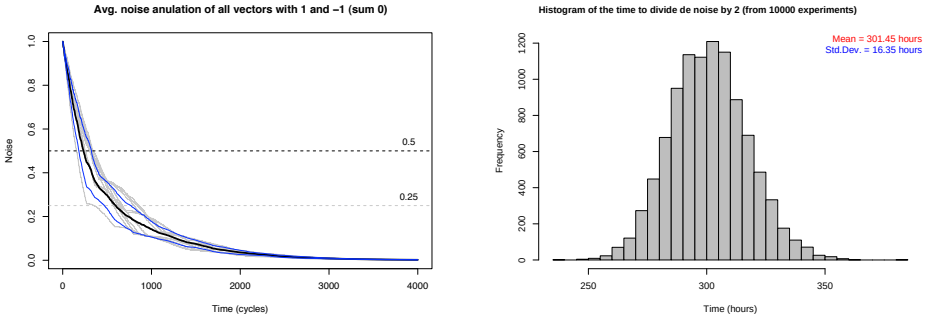


Figure 4.9: Average time to cancel a basis of vector noises.

4.3.2 Convergence in Practice

We study here the convergence of the fluid model towards the system in practice. We do two kinds of experimentations. In the first one, we look at the *speed of cancellation of a noise* by the system. We observe that in practice a noise decreases very quickly (confirming that, if the above analysis proves the convergence, it gives a very pessimistic bound on the speed of convergence). In the second kind of experiments, we are interested in the value of the deviation between the model and the system by considering systems with different amount of data. We observe that, as expected, the *trajectories remain close during time* and that they *converge when the amount of data increases*.

Noise cancellation. We consider here a basis of noise vectors, the family of vectors $\{V_i; v_1 = 1, v_i = -1, v_{j\neq i} = 0\}$. Any noise vector with null sum can be expressed as a linear combination of these vectors. We take each of these vectors and we apply to them the random product of the matrices defining the system. Figure 4.9 (Left) shows the speed of cancellation of these noise vectors. We see that in average the noise is divided by 2 after 300 time steps and by 4 after around 600 time steps. The histogram of the mean time (over 10,000 runs) to divide a noise by two is given in the left plot. We see that this time is concentrated around it mean.

Trajectories remain close. Figure 4.10 shows the two trajectories of the system and of the fluid model for two different values of B, 10^6 and $4 \cdot 10^6$ blocks. We see that

- the *deviation level is very small*;
- the *deviation level remains constant with time*, as explained above by the noise cancellation.

Convergence of the fluid model.

We then consider systems with a constant number of peers ($N = 5000$), but with different amounts of data. Figure 4.11 presents the average relative deviation of the systems for different values of B from one million to one billion blocks. We observe the convergence as B grows.

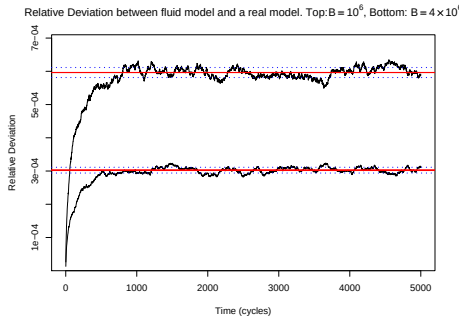


Figure 4.10: Trajectories of the deviation for two different values of the number of blocks B. The trajectories remain close during time.

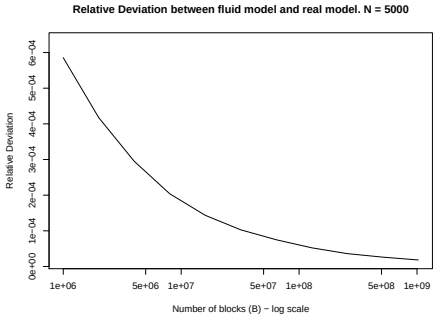


Figure 4.11: Average deviation for systems with different values of B. The fluid model converges towards the system when the amount of data increases.

4.3.3 Model Discussions - Future Directions

We showed that the Fluid Model closely models the behavior of the real system. In fact, we see that the non uniform repartition of the fragments between the different disks increases the standard deviation of the bandwidth use. To lower the impact of disk age and to have more uniform disk fillings, we propose two new policies:

- **Shuffling algorithms.** At each time step, a proportion of the fragments in the system are chosen at random and sent to a random disk. If all fragments are concerned, we obtain an *ideal system* with perfectly uniform repartition of the fragments among the disks. Note that in fact, it corresponds to the Approximated Model of Section 4.2. The advantages of such policy are that it lowers the differences in number of fragments of the disks, but also decreases the correlation between old blocks that were more present on old disks. However, a drawback is the introduction of more network traffic in the system to redistribute the fragments.

- **Biased reconstruction policy.** Another way to obtain more uniform disk fillings is to change the reconstruction policy. During the last phase of the reconstruction, the rebuilt fragments are sent to random peers. We propose to choose these peers not uniformly, but *to select with higher probability disks with less data*. By doing so, the new disks fill up faster. One drawback of this policy, is that it reinforces the correlation between blocks rebuilt at the same time. But it has the advantage of not changing the bandwidth needs.

4.4 Conclusion

In this chapter, we show through simulations and formal analysis that modeling such a system by independent blocks, each following its own Markov Chain is very far from reality: if the expectations are perfectly captured, deviations from the mean are extremely

underestimated. This is due to data loss correlation: a failing disk affects tens of thousands of blocks. We also show by simulation that these variations (e.g., in bandwidth usage) can have a severe impact on the reliability (probability of data loss).

We then introduce an Approximated Fluid Model that captures most of the system dynamic. Simulations show that this model gives very tight results. We believe that the methods proposed here can be applied in other contexts where correlation phenomena occur. We are working at adapting the presented methods to different (non Poissonian) failure models and different reconstruction models, e.g. deterministic reconstruction time. It could also be interesting to study data placement strategies other than random.

This work also raises a more theoretical question. The fluid models have a simple dynamic, since it is defined as a random product of two small dimension matrices. Determining the behavior of such a product is known to be intractable, but in our specific case we succeeded to get exact formulas and compute the moments of the distribution. It would be interesting to find general non trivial conditions (other than commutability) under which the dynamic can be computed.

CHAPTER 5

Repair Time Distribution Under Bandwidth Constraints

To ensure long-term fault tolerance, the storage system must have a self-repairing service that continuously reconstructs the fragments of redundancy that are lost. The duration of this repairing process is crucial to determine the system reliability. That is, repairs that last long increase the probability of losing data exponentially. This speed is mainly determined by how much bandwidth is available by the peers' upload link capacity, which in turn is arguably one of the most scarce resource of such systems (i.e., when compared to the processing capacity or the storage space). In the literature, the durations of the reconstructions are modeled as independent (e.g., poissonian, deterministic, or more generally following any distribution). In practice, however, numerous reconstructions start at the same time (when the system detects that a peer has failed). Consequently, they are correlated to each other because concurrent reconstructions do compete for the same bandwidth. This correlation negatively impacts the efficiency of the bandwidth utilization and henceforth the repair time.

Imagine a scenario where users are connected using a typical home connection via an Asymmetric Digital Subscriber Line (ADSL) with upload capacity of 1Mbps. We expect that only part of this bandwidth is allocated to the storage system, let us say 128kbps. The average amount of data per peer is 100 gigabytes. When a peer fails, if 100 peers participate to the repairing process at an optimal rate of 128kbps, then the system would need theoretically 17 hours to recovery the contents of the failed disk. By our models, if we consider that peers have an expected lifetime of 1 year, this repair time lasts around 22 hours, which gives a probability of data loss per year (PDLPY) of 10^{-8} (we set $s = 7$ and $r = 7$). However, due to several factors, in practice the repair time is in fact much greater than this optimal time. For instance, the *imbalance* on the amount of data per peer negatively impacts the efficiency of the bandwidth utilization. Continuing with the same example, for the same average amount of data per disk of 100 gigabytes, if the system have disks with heterogeneous capacity (limited to 3 times the average amount), then the repair time of a disk reaches 9 days. Which gives a PDLPY of 0.2. Many orders of magnitude more than the previous case! Hence, the importance of having models that estimate precisely the repairing time and henceforth the probability of data loss for limited bandwidth scenarios.

A paper with the results of this chapter has been submitted to the IEEE International Parallel & Distributed Processing Symposium (IPDPS'2011).

Our contribution

In this chapter we propose a *new analytical model that precisely estimates the repair time and the probability of losing data* of storage systems based on erasure codes. This model takes into account the bandwidth constraints of peers when processing reconstructions. Mainly, we introduce queuing models in which reconstructions are served by peers at a rate that depends on the available bandwidth.

We show that *is crucial to take into account the peer imbalance* (i.e., young peers inherently store less data than the old ones) to estimate the system efficiency. Indeed, we show that the traffic load is not well distributed among peers: young peers inherently store less data than the old ones, thus they contribute asymmetrically to the reconstruction process. Hence, we propose to introduce biases in the protocol to correct this imbalance, and show that it improves the efficiency of the system.

We discuss how far the distribution of the reconstruction time given by the model is from the exponential classically used in the literature. We exhibit the different possible shapes of this distribution in function of the system parameters. This distribution impacts the durability of the system.

We address *scheduling and control issues*. Indeed, each peer is involved in many reconstructions, thus we need to schedule their order of execution. We compare several policies, and *propose a simple greedy-like policy* that allows the system to reduce the reconstruction time.

We show a somewhat counterintuitive result that we *can reduce the reconstruction time by using a less bandwidth efficient* Regenerating Code. This is due to the *degree of freedom* given by erasure codes to choose which peers participate to the repair process.

To the best of our knowledge, this is the first detailed model proposed to estimate the distribution of the reconstruction time under limited bandwidth constraints. *We validate our model by an extensive set of simulations and by test-bed experimentation using the* GRID'5000 *platform*.

Related Work

Several works related to P2P storage systems have been done, and a large number of systems have been proposed [28, 23, 20, 69]. But few theoretical studies exist. Most studies are Markov chain models that assume in fact a poissonian reconstruction process (i.e., with independent reconstruction time). Furthermore, in these models, only the average analysis are studied and the impact of congestion is not taken into account.

In [98, 14, 38] the authors use a Markov chain model to derive the lifetime of the system. In these works, the reconstruction time follows an exponential (or geometric) distribution, which is a tunable parameter of the models. However, in practice, a large number of repairs start at the same time when a disk is lost (corresponding to tens or hundreds of GBs of data). Hence, the reconstructions are not independent of each other.

Dandoush et al. in [37] perform a simulation study of the download and the repairing process. They use the NS2 simulator to measure the distribution of the repair time. They state that a hypo-exponential distribution is a good fit for the block reconstruction time.

However, they assume that reconstruction events are independent, which means that they do not take into account their correlation when a disk fails.

Similarly to our study, other works also discuss the impacts of competition for the bandwidth. Ramabhadran et Pasquale in [99] address resource allocation problems in replicated systems. They study different schemes to optimize the average file availability.

Picconi et al. in [90] study the durability of storage systems. Using simulations they characterize a function to express the repair rate of systems based on replication. However, they do not study the distribution of the reconstruction time and the more complex case of erasure coding neither.

Organization

The remainder of this chapter is organized as follows: in the next section we give some details about the studied system, then in Section 5.2 we discuss the impact of load imbalance. The queueing model is presented in the Section 5.3, followed by its mathematical analysis. The estimations are then validated via an extensive set of simulations in Section 5.4. Lastly, in Section 5.5, we compare the results of the simulations to the ones obtained by experimentation.

5.1 Description

In this section we give details about the studied storage system and the modeling assumptions.

Peer bandwidth. In a peer-to-peer system, peers are typically connected to the network via an ADSL (Asymmetric Digital Subscriber Line) link. Thus, we model here asymmetric capacities as they are the configurations most often encountered in practice: each peer has a maximum upload and download bandwidth, resp. BW_{up} and BW_{down}; we set $BW_{down} = 10BW_{up}$ (in real systems, this value is often between 4 and 10). The bottleneck of the system is considered to be the peer links and not the network internal links.

Peer availability and peer failures. Peers can be highly available (as servers that are kept in a controlled environment), or conversely, be barely available with a low presence interval. Since we consider the case of backup storage systems, the peers are expected to stay connected at least few hours per day.

Following the work by Dimakis [41] on network coding, we use similar values of availability and failure rate from the PlanetLab [92] and Microsoft PCs traces [23]. To distinguish from transient failures, a peer is considered as failed if it leaves the network for more than a timeout, which was set to 24 hours. In that case, all data is considered lost. The Mean Time To Failure (MTTF) in the Microsoft PCs and the PlanetLab scenarios are respectively 30 and 60 days. For given values of $s = 7$ and $r = 7$, we achieve a block availability of 5 nines in the Microsoft PCs scenario. The peer failures

are then considered as independent and Poissonian with mean value given by the traces explained above. We consider a discrete time in the following and the probability to fail at any given time step is denoted as $\alpha = 1/MTTF$.

Redundancy Scheme and Repair. In this study we use Regenerating Codes (RC) [41] to introduce redundancy, as they are foreseen as the most efficient codes in terms of bandwidth usage. Similarly to the Reed-Solomon erasure codes, a data block is divided into s fragments, to which are added r fragments of redundancy. Then, the $n = s + r$ fragments are spread into n peers in such a way that the block can be regenerated by retrieving any s fragments. However, when employing the Regenerating Codes, the repairing process can be done by creating a new fragment to replace the missing one, instead of regenerating the whole block as required by Reed-Solomon codes. Hence, a lost fragment can be repaired efficiently by contacting d peers, with $s \leq d < n$ (d is called the repair degree of the block). Each one of the d peers needs to send a small sub-fragment to the *reconstructor* peer, which in turn will store the repaired fragment. This reconstructor peer which is in charge of the repair is chosen uniformly at random. To achieve this repair efficiency, these codes introduce an overhead on the fragment size (that is, how much the original fragment must be increased in size to achieve the regenerating code property). δ_{MBR} is the overhead factor of the *Minimum-Bandwidth Regenerating Codes* [41]. It is defined as follows:

$$\delta_{MBR}(d) = \frac{2d}{2d - s + 1}.$$

The most efficient case is when d is the maximum, $d = n - 1$. Hereafter we note $L_r = \delta_{MBR}(n-1)L_f$, as the amount of information transferred to reconstruct one fragment when $d = n - 1$, where L_f is the original size of the fragment.

Nevertheless, the model presented in this work can be adapted to systems using different codes to introduces redundancy, e.g., Replication, Reed-Solomon, Hierarchical codes [46], or Hybrid coding. Basically, the main change would be to replace the bandwidth efficiency of RC by the bandwidth efficiency of the other code.

Monitoring the data and network size. We consider distributed systems of any size, e.g., thousands of peers. However, for practical reasons and maintainability, the fragments of blocks are often stored on small logical subset of peers that are self-structured. This subset is inherited from the DHT terminology of P2P architectures, where they are often called *leafset* or *neighborhood*. In the following examples, we consider sizes of neighborhood of 100 to 200 peers. Hereafter in this chapter, we use the terms *peer* and *disk* interchangeably.

Table 5.1 shows a summary of the notations used in this chapter.

5.2 Preliminary: Impact of Disk Asymmetry

In this section we start by showing that the efficiency of the system is affected by the imbalanced distribution of data among peers. We then estimate analytically this imbalance and its impact. After this preliminary study, the definition of the queuing model is given in Section 5.3.

Factor of efficiency. When a peer fails, it is replaced by a new peer with an *empty* disk. Since disks fill up during the system life, a recently replaced disk is empty, while an old disk contains many fragments. Hence, at any given time *disks with very heterogeneous number of fragments* are present in the system. This heterogeneity has a strong impact on the reconstruction process: (1) when a disk dies, the number of block reconstructions that start depends on the number of fragments present in this disk. A lot of fragments are lost if the disk was full, but much less for a young disk. (2) during the repair, the peers have to send fragments to the reconstructors that rebuild the missing fragments. A peer storing more fragments has to send a lot more fragments during this phase than a peer with fewer fragments. Hence, such peers become a bottleneck of the system, when on the contrary the less loaded peers stay idle during some part of the time.

To estimate the impact of this imbalance on the system, we introduce a *factor of efficiency* ρ when the system is under load, defined as

$$\rho(load) = \frac{work}{\min(load, bandwidth)}$$

where *load* is the sum over all peers of the number of fragments in their queues at the beginning of the time step; *bandwidth* is the total bandwidth of the system ($BW_{up} \cdot N/\tau$) accounted in time steps of size τ; and *work* is the number of fragments that were effectively uploaded by the peers during the time step. When $\rho = 1$, the system works at its maximum speed, meaning that no peer was idle while another one could not finish its work. Note that ρ greatly depends of the load. If the load is very large compared to the bandwidth of the system, every peer works at almost full capacity and the efficiency is close to one. Similarly, when the load is small, everybody has few fragments to upload and all the work is done. But, between these two cases, the imbalance between the peers causes a range of inefficiencies.

Estimation of the Imbalance

The disk size has in fact a very strong effect on the general imbalance of the system. Figure 5.1 shows an histogram with the number of fragments in failed disks. These results are obtained by simulation of $N = 200$ peers with $MTTF = 60$ days (1440 hours). The amount of data per peer is $14GB$. We set $s = r = 7$, and the fragment size $l_r = 2$ MB. Hence we have a total of $F = 7{\cdot}10^5$ fragments in the system. Then, the average number of fragments per peer is $\bar{D} = 7000$.

We denote the disk capacity of peers as C (number of fragments). Hence, $x = C/\overline{D}$ is the size factor of disks, i.e., how big is the disk when compared to the average amount of fragments per disk in the system. When the size factor $x = 3$ (that is, disk capacity $C = 21,000$ fragments), the imbalance is very large. At the opposite, when $x = 1.1$,

Table 5.1: Summary of the main notations.

N	Total number of peers
s	Number of initial fragments of a block
r	Number of redundancy fragments of a block
n	Number of fragments of a block, $n = s + r$
d	Repair degree of the Regenerating Code, by default $d = n - 1$
δ_{MBR}	Efficiency of the Regenerating Codes
L_f	Size of a fragment, in bytes
L_r	Amount of data to repair a fragment
B	Total number of blocks in the system
F	Total number of fragments in the system
α	Peer failure rate ($\alpha = 1/MTTF$)
N_F	Number of peers with full disks
φ	Ratio of full disks, N_F/N
C	Capacity of a disk (number of fragments)
$\bar{D}$	Average number of fragments per disk
x	Disk size factor, $x = C/\overline{D}$
BW_{up}	Peer upload bandwidth (kbit/s)
v	Rate at which a disk fills up (fragments per cycle)
$T_{\max}$	Number of time steps to fill up a disk, $T_{\max} = C/v$

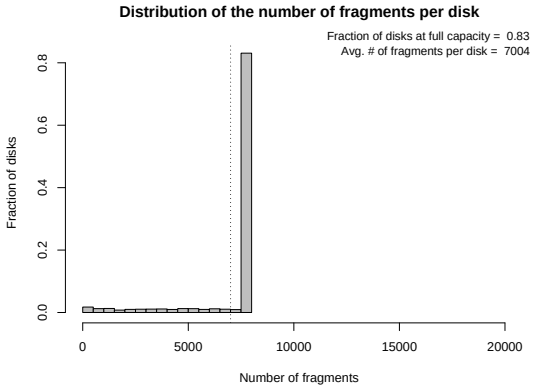
the disk size is close to the average number of pieces per disk in the system. Hence, most of the disk fillings become very concentrated around the average value (83%, in our example). The disks that are not full (17%) have an almost uniform distribution. In the following, we give a method to calculate that imbalance analytically.

Disk age and disk size distributions can be precisely approximated for systems with a large number of blocks. The block fragments are reconstructed by peers that have free space in their disks (i.e., there are $N - N_F$ such peers, where N_F is the number of peers with full disks). Since these peers are chosen at random to reconstruct the blocks, at each time step the distribution of the rebuilt fragments among peers follows a multinomial distribution with parameters: the number of rebuilt fragments and $1/(N - N_F)$. As the multinomial distribution is very concentrated around its mean, the *filling up process can be approximated by an affine process of its age*, in which, at each time step, each disk gets the number of reconstructed fragments divided by the number of non-full peers, roughly

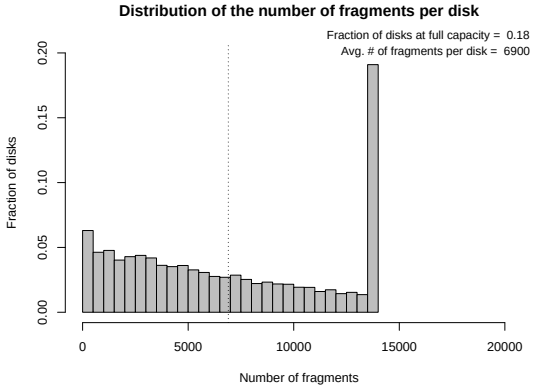
$$v = \frac{\alpha F}{N - N_F}$$

where α is the peer failure rate. This filling process stops when the disk is full. That is after a number of time steps $T_{\max}$ such that $C = \alpha T_{\max} F/(N - N_F)$, where C is the peer disk capacity (maximum number of fragments per disk). The number of fragments of a disk thus depends on the age of the disk.

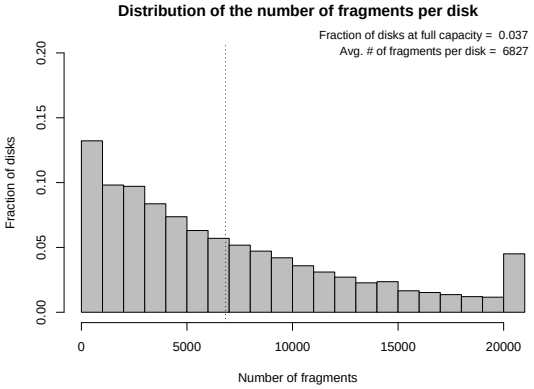
At each time step a disk has a probability α to experience a failure. Hence, the dead



(a) $x = 1.1$



(b) $x = 2.0$



(c) $x = 3.0$

Figure 5.1: Distribution of fragments per failed disk for different disk size factor x of 1.1, 2, and 3. The number of full disks in each scenario is respectively 83%, 18%, and 4%. (y-scales are different)

age of a disk follows a geometric law of parameter α. That is, $\Pr[\text{dead age} = T] = (1-\alpha)^{T-1}\alpha$. Hence the distribution of the number of fragments in a disk follows a truncated geometric distribution, that is, for $1 \leq T < T_{\max}$

$$\begin{aligned} Pr[D = vT] &= (1-\alpha)^{T-1}\alpha, \text{ and} \\ Pr[D = C] &= 1-(1-\alpha)^{T_{\max}}. \end{aligned} \tag{5.1}$$

Note that here v, N_F, and $T_{\max}$ are unknown for the moment. The value of v depends on the number of full disks N_F, and of $T_{\max}$ depends directly of the filling rate v. To find the value of these variables, we use the fact that we know the expectation of the geometric distribution which is just the average number of fragments inside the system. This number is F/N (we neglect here the fragments that are in reconstruction, first order approximation for small α). Hence, we get $\mathbb{E}[D] = \overline{D} := F/N$. By definition, the

expectation is also given by

$$\mathbb{E}[D] = \sum_{i=1}^{T_{\max}-1} vi(1-\alpha)^{i-1}\alpha + C(1-(1-\alpha)^{T_{\max}}).$$

To obtain $T_{\max}$, we now have to solve the equation:

$$\frac{1}{x} = \frac{1-\alpha-(1-\alpha)^{T_{\max}+1}}{\alpha T_{\max}},$$

obtained by identifying the two expressions for the expectation, by dividing by v, and because $C = x\overline{D}$. By solving that equation using the Maple software, we obtain that

$$T_{\max} = \frac{\alpha W(\frac{1}{\alpha}\ln(1-\alpha)x(1-\alpha)^{\frac{x+\alpha-x\alpha}{\alpha}}) - \ln(1-\alpha)x(1-\alpha)}{\ln(1-\alpha)\alpha},$$

where W is the Lambert W function. For example, when $MTTF = 1440$ hours ($\alpha = 1/1440$), the number of full disks and the number of time steps to fill up a disk are, for different disk capacities:

x	N_F (in %)	$T_{\max}(hours)$
1.1	83	278
1.5	42	1257
2	20	2293
3	6	4060

We verify that these values are very close to the ones obtained by simulation (Figure 5.1).

Effects of the Imbalance on the Bandwidth Efficiency Since some peers store less fragments, their load during the reconstruction process is also smaller. Thus, the overall bandwidth of the system is not fully utilized.

In a system using Regenerating Codes encoding, to repair a fragment, $d = n-1$ small sub-fragments have to be sent to the peer in charge of the reconstruction. Simulations show that the speed of the reconstruction is given by the time that the *most loaded peer* takes to send the fragment. This time is in turn given by the number of fragments stored by this peer. We get this number from the distribution of the number of fragments per peer previously derived. For a majority of data blocks, *the most loaded peer storing one of its fragment is in fact a full disk*. This claim is valid for most systems in practice, that is, for the parameters usually found in the literature.

Indeed, recall that N_F denotes the number of full disks (and $\varphi = N_f/N$ the fraction of full disks). We compute the probability for a block that one of its fragment is on a full peer (with $n-1$ available fragments when it is being repaired). Recall also that a full disk stores x times the average number of fragments per disk in the system. Then, the fraction of fragments stored on full disks is φx. The probability of the block to have at least one fragment on a full disk is then

$$P_{full} = 1-(1-x\varphi)^{n-1}.$$

For a system with $n = 14$ (the value of N_f for different values of x is given above), the probability for different disk capacities is

x	1.1	1.5	2	3
φx	0.91	0.63	0.4	0.18
P_{full}	$1 - 10^{-14}$	$1 - 10^{-5}$	$1 - 10^{-3}$	0.92

We see that for most practical systems, each block has a fragment on a full disk. Hence, it is enough to consider the work done by the most loaded peers to obtain the reconstrution times. These peers have a load greater than the average load by a factor of $\frac{1}{x}$.

Factor of efficiency. An other way to phrase it: the factor of efficiency ρ of the system is approximately

$$\rho \approx \frac{1}{x}$$

where x is the fraction between disk capacity and the average number of fragments per disk.

More complex models for large disk capacities. We consider that in practice, for fairness issues, the storage system sets a limit of disk capacity not too far from the average amount of data stored. A factor x between 1.1 and 3 seems reasonable. For systems with a very large disk capacity (for example $x = 10$), ρ has to be estimated in a different way. As a matter of fact, a large number of blocks store no fragments on full disks. It is thus not enough to only consider the load of the full disks. This difficulty can be addressed by using a *multi-queue model*. The peers are partitioned into a number C of classes depending on the number of data they store. The model has one queue per class. When a disk fails, we estimate the number of fragments that each class has to upload, that is how much work they do, and in this way derive the factor of efficiency ρ. The analysis of this model is beyond the scope of our study here.

5.3 The Queueing Model

We introduce here a *Markovian Model* that allows us to estimate the reconstruction time under bandwidth constraints. The model makes an important assumption:

1. *The limiting resource is always the upload bandwidth.*

Assumption 1 is reasonable as download and upload bandwidths are strongly asymmetric in common installations. Using this assumption, we model the storage system with a *queue storing the upload load of the global system*.

5.3.1 Model Definition

We model the storage system with a Markovian queuing model storing the upload needs of the global system. The model has one server, Poissonian batch arrivals and deterministic time service ($M^{\beta}/D/1$, where β is the batch size function). We use a discrete time model. The peers in charge of repairs process blocks in a FIFO order.

Chain States. The state of the chain at a time t is the current number of fragments in reconstruction, denoted by $Q(t)$.

Transitions. At each time step, the system reconstructs blocks as fast as its bandwidth allows it. The upload bandwidth of the system, $BW_{up}N$, is the limiting resource. Then, the *service* provided by the server is

$$\mu = \rho \frac{BW_{up}N}{L_r \tau},$$

which corresponds to the number of fragments that can be reconstructed at each time step τ. The factor ρ is the bandwidth efficiency as calculated in the previous section, and L_r is the number of bytes transferred to repair one fragment. Hence, the number of fragments repaired during a time step t is $\mu(t) = \min(\mu, Q(t))$.

The *arrival process* of the model is caused by peer failures. When a failure occurs, all the fragments stored in that peer are lost. Hence, a large number of block repairs start at the same time. We model this with batch inputs (sometimes also called *bulk arrival* in the literature). The size of an arrival is given by the number of fragments that were stored on the disk. As explained in Section 5.2, it follows a truncated geometric distribution.

We define β as a random variable taking values $\beta \in \{0, v, 2v, \ldots, T_{max}v\}$, which represents the number of fragments inside a failed disk (see Equation (5.1) for the probability distribution function of β). Recall that v is the speed at which empty disks get filled, and that $T_{max} = C/v$ is the elapsed time to fill a disk. Further on, β/v is the elapsed time to have a disk with β fragments.

The arrival process of the model is Poissonian. A batch arrives during a time step with probability f, with $f \approx \alpha N$. For the simplicity of the exposition, we consider here that only one failure can happen during a time step (note that to ensure this, it is sufficient to choose a small enough time step). Formally, the transitions of the chain are, for $\forall i \geq \mu$,

$$\begin{array}{lcll} Q_i & \rightarrow & Q_{i-\mu} & \text{with prob. } 1-f & \text{// no failures} \\ Q_i & \rightarrow & Q_{i-\mu+\beta}, \forall \beta & \text{with prob. } f(1-\alpha)^{\frac{\beta}{v}-1}\alpha & \text{// failures} \\ Q_i & \rightarrow & Q_{i-\mu+C} & \text{with prob. } f(1-(1-\alpha)^{T_{\max}}) & \text{// full disk fail} \end{array}$$

When $0 \leq i < \mu$, the i blocks in the queue at the beginning of the time step are recon-

structed at the end. Hence, we have transitions without the term $i - \mu$:

$$
\begin{array}{lcll}
Q_i & \rightarrow & Q_0 & \text{with prob. } 1-f \quad // \text{ no failures} \\
Q_i & \rightarrow & Q_\beta, \forall \beta & \text{with prob. } f(1-\alpha)^{\frac{\beta}{v}-1}\alpha \quad // \text{ failures} \\
Q_i & \rightarrow & Q_C & \text{with prob. } f(1-(1-\alpha)^{T_{\max}}) \quad // \text{ full disk fail}
\end{array}
$$

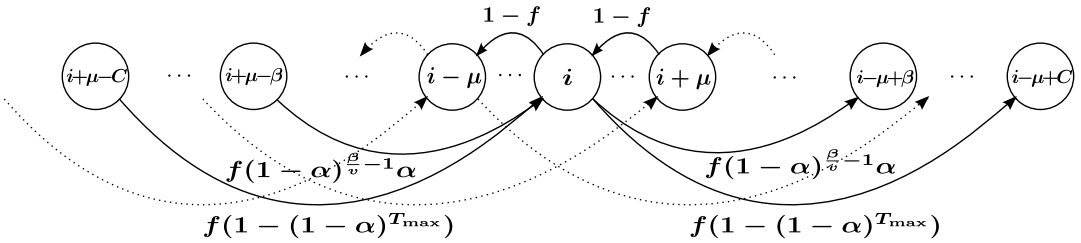


Figure 5.2: Transition around state i of the Markovian queuing model.

Figure 5.2 presents the transitions for a state i. The following table summarizes the notation introduced in this section.

$Q(t)$	Number of fragments to be repaired
f	Batch arrival rate, $f = \alpha N$
β	Number of fragments on a failed disk (i.e., batch size)
ρ	Factor of efficiency, $\rho \approx \frac{1}{x}$
μ	Service rate, $\mu = \rho B W_{up} N / L_r \tau$ (fragments per time step)

5.3.2 Analysis

Here, we give the expressions to estimate the values of two important system metrics: the distribution of the block reconstruction time and the probability of data loss. These expressions are derived from the stationary distribution of the Markovian model, as presented in the following.

A Normalized Model. The queuing model has a service of μ and an input process of average $f\beta$. To simplify the presentation of the analysis, we introduce then a *normalized model* with service of 1, hence an input of mean $\beta' = \beta/\mu$.

5.3.2.1 Stationary Distribution

We analyze here the stationary state of this normalized queuing model. As the chain is irreducible and aperiodic, it exists when the service rate is larger than the load. Let P be the probability generating function of the Markovian model, that is P is defined as:

$$P(z) = \sum_i P_i z^i,$$

where P_i is the probability that the system is in state i, that is, i fragments have to be repaired.

The system reconstructs one block per time step (unless of course, no block is in the queue). It is translated in the generating function language into a division by z. The effect of a peer failure is translated by a multiplication by the probability generating function of the input I, defined as

$$I(z) = \sum_{j=0}^{\infty} I_j z^j,$$

with I_j the probability that the batch is of size j. Hence, we obtain the functional equation

$$\left(\frac{P(z) - P_0}{z} + P_0\right) I(z) = P(z).$$

It gives

$$P(z) = \frac{(z-1)P_0}{\frac{z}{I(z)} - 1}.$$

As $P(1) = 1$, $I(z) - z$ admits 1 as a root and thus can be written as $I(z) - z = (z-1)Q(z)$. We have

$$P(z) = \frac{P_0 I(z)}{Q(z)}. \tag{5.2}$$

As we have seen in Section 5.2, the size of the input follows a truncated geometric distribution of parameter α. A batch is of size vj with probability $(1-\alpha)^{j-1}\alpha$, for $j \in [0, 1, ..., T_{\max}]$. It gives

$$I(z) = (1-f) + f \sum_{j=1}^{T_{\max}-1} (1-\alpha)^{j-1} \alpha z^{vj} + f(1-\alpha)^{T_{\max}-1} z^{vT_{\max}}.$$

It can be rewritten as

$$I(z) = 1 + \frac{f(z^v - 1)(z^{T_{\max}}(1-\alpha)^{T_{\max}} - 1)}{(1-\alpha)z^v - 1}.$$

We factorize $I(z) - z$ by $(z-1)$. We get

$$\begin{aligned} Q(z) &= I(z) - z \\ &= (z-1)(-1 + \tfrac{f(\sum_{j=1}^{v} z^i)(z^{vT_{\max}}(1-\alpha)^{T_{\max}} - 1)}{(1-\alpha)z^v - 1}). \end{aligned}$$

The value of P_0 is obtained by the normalization $\sum_{i=0}^{\infty} P_i = 1$ which implies $P(1) = 1$.

$$P_0 = \frac{Q(1)}{I(1)} = 1 - \frac{1}{\alpha}(fv((1-\alpha)^{T_{\max}} - 1)).$$

We now have an expression of the three terms of Equation 5.2 and we get a close form of

the probability generating function $P(z)$.

5.3.2.2 Distribution of the Waiting Time

The distribution of the block reconstruction time is given by the stationary distribution P of the model calculated above. As we have Markovian (batch) arrivals, the probability for a batch to arrive when there are n blocks in the queue is exactly P_n (for the difference of distribution for an arriving customer and an outside observer, see for example [31]). If there are Q fragments in the queue when a batch of size $\beta' = jv$ arrives, the arriving fragments have waiting times of $Q+1$, $Q+2$, $Q+\beta'$. We define the probability generating function J as

$$J(z) = \sum_{j=1}^{T_{\max}} \left((1-\alpha)^{j-1} \alpha \sum_{i=1}^{jv} z^i \right).$$

The probability generating function W of the waiting times then is just

$$W(z) = P(z)J(z).$$

The distribution of the waiting times can then be directly obtained from the generating function by extracting its coefficients

$$\Pr(W = k) = [z^k]W(z) = \left. \frac{d^k W(z)}{k!(dz)^k} \right|_{z=0}. \tag{5.3}$$

The first coefficients can be computed numerically and then a singularity analysis gives the asymptotic behavior, see for example [49]. Hence, the value of $\Pr(W = k)$ can be computed analytically. However, in the following, we also use another method and calculate them numerically by iterating the queuing model.

5.3.2.3 Number of Dead Blocks

The expected number of dead blocks is indirectly given by the model by computing the waiting time in the queue of a block that has to be reconstructed.

As a matter of fact, a block dies if it loses, before the end of the reconstruction, the $r-1$ fragments of redundancy that it has left when the repair starts, plus an additional fragment. The probability for a peer to still be alive after a period of time of θ time step is $(1-\alpha)^\theta$, where α is the probability for a disk to die during a time step, that is

$$\alpha = \frac{\tau}{MTTF}.$$

Hence a good approximation of the probability $\Pr[die]$ to die during a reconstruction lasting a time θ is given by

$$\Pr[die|W = \theta] = \sum_{i=r}^{s+r} \binom{s+r}{i} (1-(1-\alpha)^\theta)^i ((1-\alpha)^\theta)^{s+r-i}.$$

For practical systems, the ratio $\theta/MTTF$ is small as the probability to of data loss should be very low. Hence $\Pr[die]$ is well approximated by

$$\Pr[die|W=\theta] \approx \binom{s+r}{r}(1-(1-\alpha)^{\theta})^{r}((1-\alpha)^{\theta})^{s-1}.$$

From this and from the distribution of the waiting time, we get the probability to die during a reconstruction, P_D, with

$$P_D = \sum_{i=0}^{\infty} \Pr[\text{die}|W=i]\Pr[W=i].$$

The number of dead blocks during a time T, D_T, is then obtained by the number of reconstructions during T, R_T:

$$D_T = P_D R_T. \tag{5.4}$$

5.3.2.4 Bandwidth Usage

The bandwidth usage is directly given by the distribution of the number of reconstructions being processed by the system, which comes from the stationary distribution of the queuing model.

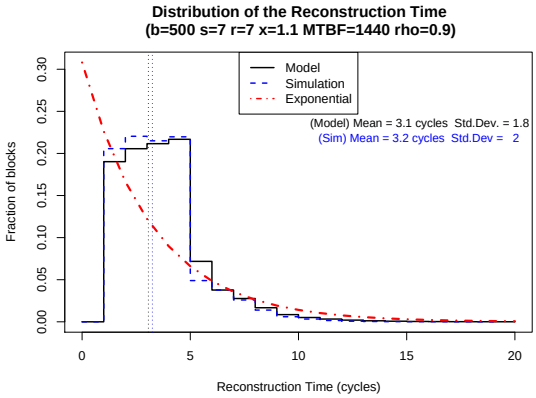
5.4 Results

To validate our model, we compare its results with the ones produced by simulations, and test-bed experimentation. We use a custom cycle-based simulator. The simulator models the evolution of the states of blocks during time (number of available fragments and where they are stored) and the reconstructions being processed. When a disk failure occurs, the simulator updates the state of all blocks that have lost a fragment, and starts the reconstruction if necessary. The bandwidth is implemented as a queue for each peer. The reconstructions are processed in FIFO order.

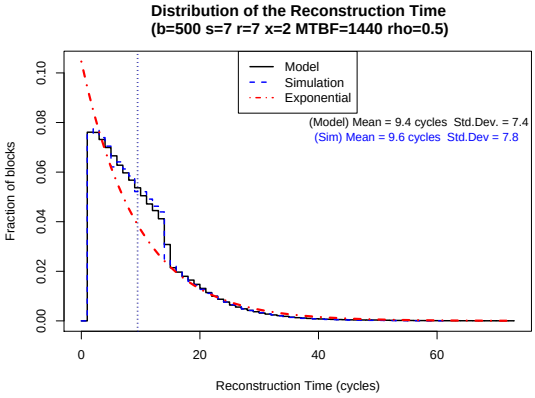
We study the distribution of the reconstruction time and compare it with the exponential distribution which is often used in the literature. We then discuss the cause of the data losses. Finally, we present two important practical implementation points: (1) when choosing the parameters of the Regenerating Code, it is important to give to the peer in charge of the repair a choice between several peers to retrieve the data; (2) we show the strong impact of different scheduling options on the data loss rate.

5.4.1 Distribution of Reconstruction Time

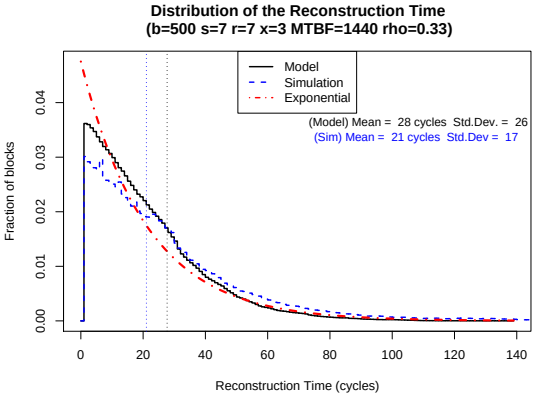
Figure 5.3 shows the distribution of the reconstruction time and the impact of the peer asymmetry on the reconstruction time for the following scenario: N = 100, s = 7, r = 7, L_r=2 MB, B = 50000, MTTF = 60 days, BW_{up} = 128 kpbs. All parameters are kept constant, except the disk size factor x (recall that x is the ratio of the maximum capacity over the average amount of data per peer).



(a) $x = 1.1$



(b) $x = 2.0$



(c) $x = 3.0$

Figure 5.3: Distribution of reconstruction time for different disk capacities x of 1.1, 2, and 3 times the average amount. The average reconstruction times of simulations are respectively 3.2, 9.6, and 21 hours (Note that some axis scales are different).

First, we see that the model (dark solid line) closely matches the simulations (blue dashed line). For example, when $x = 1.1$ (top plot), the curves are almost merged. The average reconstruction times are 3.1 cycles vs 3.2 for the model. We see that there is a small gap when $x = 3$. As a matter of fact, we saw in Section 5.2 that simulating the queue of the full disks is an approximation in this case, as only 92% of the blocks have a fragment on a full disk.

Second, we confirm the strong impact of the disk capacity. We see that for the three values of x considered, the shape of the reconstruction times are very different. When the disk capacity is close to the average number of fragments stored per disk (values of x close to 1), almost all disks store the same number of fragments (83% of full disks). Hence, each time there is a disk failure in the system, the reconstruction times span between 1 and C/μ, explaining the rectangle shape. The tail is explained by multiple failures happening when the queue is not empty. When x is larger, disks also are larger, explaining that it takes a longer time to reconstruct when there is a disk failure (the average reconstruction time raises from 3.2 to 9.6 and 21. when x goes from 1.1 to 2. and

3.). As the number of fragments per disk follows a truncated geometric distribution, we see the rectangle shape is replace by a trapezoidal shape explained by the large range of disk fillings.

Third, we compare the distributions obtained with the exponential distribution that is classically used in the literature. We see that the distributions are far from the exponential when $x = 1.1$ and $x = 2$, but get closer for $x = 3$. Hence, as we will confirm, the exponential distribution is only a good choice for some given sets of parameters. To finish, note that the tails of the distribution are close to exponential.

Figure 5.4 presents the distribution of a distributed storage system experiencing three different rates of failures: MTTF of 90, 180 and 360 days. We clearly see the evolution of the shape of the distribution due to the larger probability to experience failures when the peer queues are still loaded. The average reconstruction time increases from 5 hours when the MTTF is 360 days to 12 hours when the MTTF is 90 days.

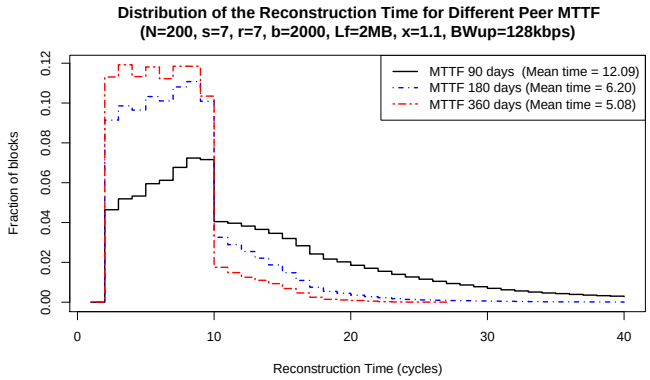


Figure 5.4: Distribution of reconstruction time for different MTTFs.

For the sake of completeness, we ran simulations for different sets of parameters. We present in Table 5.2 a small subset of these experiments. The default values are $s = 7, r = 7, c = 1.1, N = 100, B = 5 \times 10^4, MTTF = 60$ days, $BW_{up} = 128$ kbps.

Table 5.2: Reconstruction time T (in hours) for different parameters.

(a) Amount of data ($B \times 10^2$).

B	500	1000	1500
T_{sim}	3.26	8.65	25.60
T_{model}	3.06	8.15	23.91

(b) Number of peers (N).

N	100	200	300	400
T_{sim}	3.26	1.91	1.51	1.25
T_{model}	3.06	1.80	1.43	1.17

(c) Disk capacity (c).

c	1.1	1.5	2.0	3.0
T_{sim}	3.26	5.50	9.63	21.12
T_{model}	3.06	5.34	9.41	27.7

(d) Peer Lifetime (MTTF).

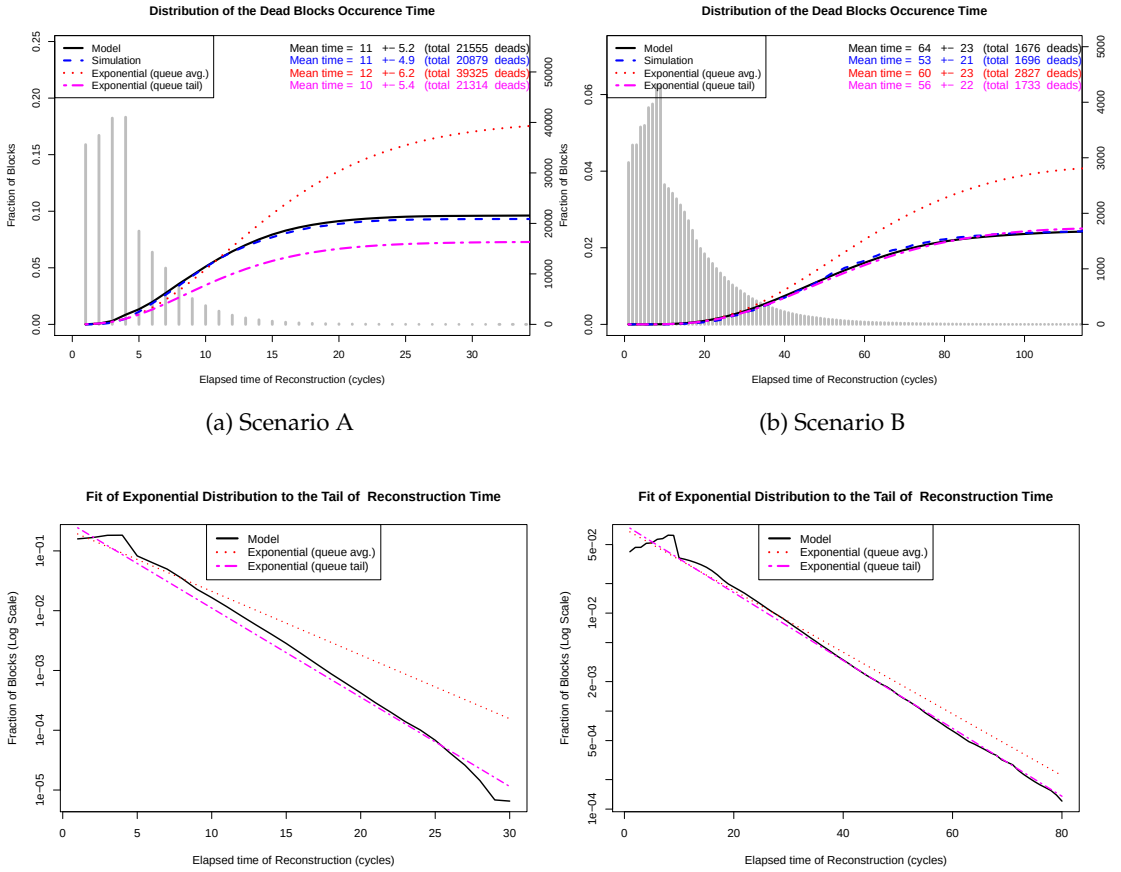
$MTTF$	60	120	180	365
T_{sim}	3.26	2.90	2.75	2.65
T_{model}	3.06	2.68	2.60	2.49

(e) Peer Upload Bandwidth (kbps).

BW_{up}	64	128	256	512
T_{sim}	8.75	3.26	1.69	1.07
T_{model}	8.25	3.06	1.61	1.03

5.4.2 From Where the Deads Come From?

In this section, we discuss in which circumstances the system has more chances to lose some data. First a preliminary remark: backup systems are conceived to experience basically no data loss. Thus, for realistic sets of parameters, it would be necessary to simulate the systems for a prohibitive time to see data losses in our simulations. We hence present here results for scenarios where the redundancy of the data is lowered ($r = 3$ and $r = 5$).



(a) Scenario A

(b) Scenario B

(c) Scenario A (fitting)

(d) Scenario B (fitting)

Figure 5.5: (Top): Distribution of dead blocks reconstruction time for two different scenarios. Scenario A: $N = 200, s = 8, r = 3, b = 1000, MTTF = 60$ days. Scenario B: $N = 200, s = 8, r = 5, b = 2000, MTTF = 90$ days. (Bottom): Fitting of exponential distribution with the tail of queueing model (axis scales are different).

We plot in Figure 5.5 the cumulative number of dead blocks that the system experiences for different reconstruction times. We give this fraction in function of the time the block spent in the system before dying. For the queuing model, we derive the expected number of blocks that died at time T from the distribution of the reconstruction time. A block dies at time T if its reconstruction process lasts a time $\theta \geq T$ and that it loses r

fragments during time T with at least one exactly at time T. This can be expressed as

$$N[\text{die at time } T] = \Pr[\text{die at time } T] \sum_{\theta \geq T} NP[W = \theta]$$

with

$$\Pr[\text{die at time } T] = \binom{s+r-1}{r-1}(1-(1-\alpha)^T)^r((1-\alpha)^T)^{s-1} - \binom{s+r-1}{r-1}(1-(1-\alpha)^{T-1})^r((1-\alpha)^T)^{s-1}.$$

We give the distribution of the reconstruction times as a reference (vertical lines). The model (black solid line) and the simulation results (blue dashed line) are compared for two scenarios with different number of blocks: there is twice more data in Scenario B.

The first observation is that the queueuing models predict well the number of dead experienced in the simulation, for example, in the scenario A the values are 21,555 versus 20,879. The results for an exponential reconstruction time with the same mean value are also plotted (queue avg.). We see that this model is not close to the simulation for both scenarios (almost the double for Scenario A). We also test a second exponential model (queue tail): we choose it so that its tail is as close as possible to the tail than the queuing model (see Figures 5.5a and 5.5b). We see that it gives a perfect estimation of the dead for Scenario B, but not for Scenario A.

In fact, two different phenomena appear in these two scenarios. In Scenario B (higher redundancy), the *lost blocks are mainly coming from long reconstructions*, from 41 to 87 cycles (tail of the gray histogram). Hence, a good exponential model can be found by fitting the parameters to the tail of the queuing model. On the contrary, in Scenario A (lower redundancy), the *data loss comes from the majority of short reconstructions*, from 5.8 to 16.2 cycles (the right side of the rectangular shape). Hence, in Scenario A, having a good estimate of the tail of the distribution is not at all sufficient to be able to predict the failure rate of the system. It is necessary to have a good model of the complete distribution!

5.4.3 Discussing the Implementation of Regenerating Codes

As presented in Section 5.1, when the redundancy is added using regenerating codes, $n = s + r$ peers store a fragment of the block when s are enough to retrieve the block. When a fragment is lost, $s \leq d \leq n - 1$ peers are in charge of repairing the fragments. The larger d is, the smaller is the bandwidth needed for the repair. Figures 5.6 and 5.7 show the reconstruction time for different values of the degree d. We observe an interesting phenomena: at the opposite of the common intuition, the average reconstruction time decreases when the degree decreases: 10 cycles for $d = 13$, and only 6 cycles for $d = 12$. The bandwidth usage increases though (because the δ_{MBR} is higher when d is smaller). The explanation is that the decrease of the degree *introduces a degree of freedom* in the choice of the peers that send a sub-fragment to the peer that will store the repaired fragment. Hence, the system is able to lower the load of the more loaded disks and to *balance more evenly the load between peers*.

In fact, we can estimate for which degree of freedom, the reconstruction time is minimum. It happens when the load of the full disks is the same as the load of the other

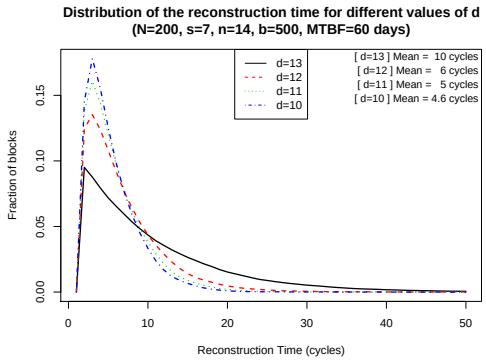


Figure 5.6: Distribution of reconstruction time for different values of degree d.

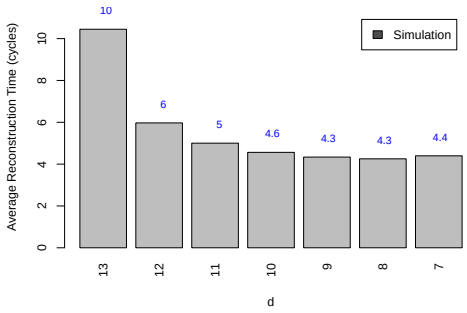


Figure 5.7: Average Reconstruction Time for different values of degree d. Smaller d implies more data transfers, but may mean smaller reconstruction times!

disks. We define $\delta = n - 1 - d$ the allowed degree of freedom for the choice of which peers uploads the sub-fragments. The full disks store a proportion φx of the fragments of the system, with φ the fraction of full disks. We simply look at the how much work we *must* do on the full disks. The probability to have i fragments (among the $n-1$ fragments) on full disks is $\binom{n-1}{i}(\varphi x)^i(1-\varphi x)^{n-1-i}$. Those blocks sends $i - \delta$ units of work the full disks (whenever $i \geq \delta$). So the load of the full disks is

$$\sum_{i=\delta}^{n-1}(i-\delta)\binom{n-1}{i}(\varphi x)^i(1-\varphi x)^{n-1-i}.$$

We presented here a cut argument for only two classes of peers (full disks and non full disks). This argument can be generalized to any number of peer classes.

When the load of the full disks becomes equal to the load of the other disks ($\sum_{i=\delta}^{n-1}(d-i+\delta)\binom{n-1}{i}(\varphi x)^i(1-\varphi x)^{n-1-i}$), it is no more useful to decrease d. We see that the average reconstruction time increases when d is too small, as the increased usage of bandwidth is no more compensated by a better balance of the load.

Note that this phenomena exists for other codes like Reed Solomon where the peer in charge of the reconstruction has to retrieve s fragments among the $s + r - 1$ remaining fragments.

5.4.4 Scheduling

As peers have a large number of repairs to carry out but very limited bandwidth, the question of which repairs to do first is crucial. In this section, we study three different scheduling choices: FIFO, RANDOM, and MOST-DAMAGED data block first.

The FIFO is the default scheduling in the simulator, as discussed in Section 5.1, the blocks are processed in the order of arrival. In the RANDOM scheduling, the simulator processes blocks in a random order (at each time step the list of blocks to be reconstructed

is shuffled). In the MOST-DAMAGED scheduling the blocks are ordered by the level of redundancy (i.e., blocks with less fragments available come first). In case of tied values, then the FIFO order is assumed.

Figure 5.8 presents the reconstruction time of these three schedulings. All strategies give almost the same average reconstruction time, 4.40, 4.43, 4.43 respectively for FIFO, RANDOM and MOST-DAMAGED. We see that their distribution changes slightly. In the RANDOM order the shape has the form of a geometric distribution, with many blocks finishing the reconstruction "early". However, as depicted in Figure 5.8, the differences in the number of dead blocks are enormous. When using the RANDOM scheduling, the dead increases considerably, as expected. The MOST-DAMAGED strategy has a reconstruction time very close to the others but the number of losses is much lower. Hence, this is the strategy of choice when implementing such systems.

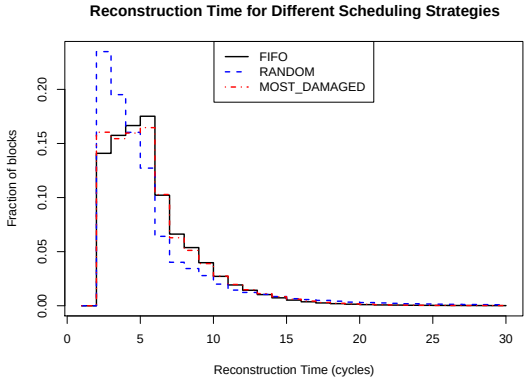


Figure 5.8: Reconstruction time for different scheduling strategies. The average reconstruction time is almost the same (4.4 cycles), but the distribution changes.

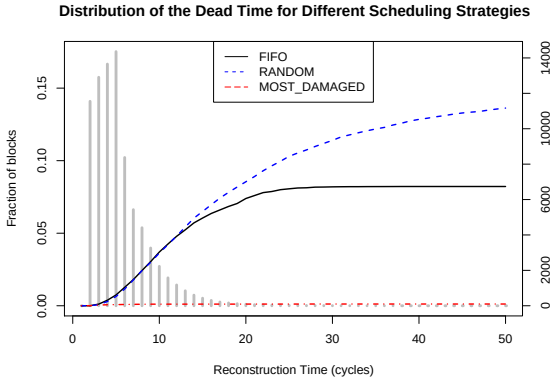


Figure 5.9: Cumulative number of dead blocks for different scheduling strategies. Processing the most damaged first is the best strategy.

5.5 Experimentation

Aiming at validating the simulation and the model results, we performed a batch of real experimentation using the GRID'5000 platform [57]. We used a prototype of storage system implemented by a private company (Ubistorage [119]).

Our goal is to validate the main behavior of the reconstruction time in a real environment with shared and constrained bandwidth, and measure how close they are to our results.

5.5.1 Storage System Description

In few words, the system is made of a storage layer (upper layer) built on top of the DHT layer (lower layer) running Pastry [107]. The lower layer is in charge of managing

the logical topology: finding peers, routing, alerting of peer arrivals or departures. The upper layer is in charge of storing and monitoring the data.

Storing the data. The system uses Reed-Solomon erasure codes [79] to introduce redundancy. Each data block has a peer responsible of monitoring it. This peer keeps a list of the peers storing a fragment of the block. The fragments of the blocks are stored locally on the PASTRY leafset of the peer in charge [72].

Monitoring the system. The storage system uses the information given by the lower level to discover peer failures. In PASTRY, a peer checks periodically if the members of its leafset are still up and running. When the upper layer receives a message that a peer left, the peer in charge updates its block status.

Monitored metrics. The application monitors and keep statistics on the amount of data stored on its disks, the number of performed reconstructions along with their duration, the number of dead blocks that cannot be reconstructed. The upload and download bandwidth of peers can be adjusted.

5.5.2 The GRID'5000 Infrastructure

GRID'5000 is an infrastructure dedicated to the study of large scale parallel and distributed systems. It provides a highly reconfigurable, controllable and monitorable experimental platform to scientists. The platform contains 1582 machines accounting for 3184 processors and 5860 cores. The machines are geographically distributed on 9 different hosting sites in France (two additional sites in Luxemburg and Porto Alegre, Brazil are being added). These site are connected to RENATER Education and Research Network with a 10Gb/s link.

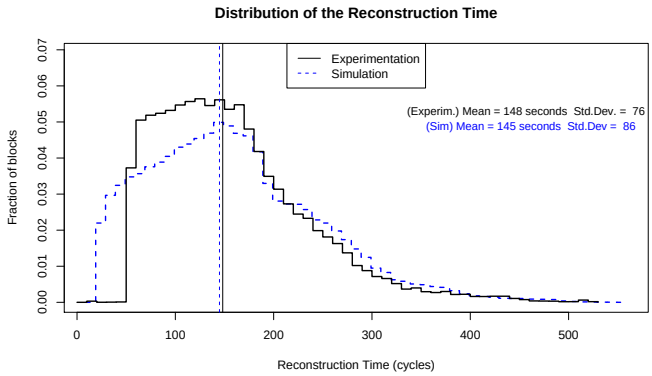


Figure 5.10: Distribution of reconstruction time on a experimentation with 64 nodes during 4 hours compared to simulations.

5.5.3 Experimentation Results

There exist a lot of different storage systems with different parameters and different reconstruction processes. The goal of the work is not to precisely tune a model to a

specific one, but to provide a general analytical framework to be able to predict any storage system behavior. Hence, we are more interested here by the global behavior of the metrics than by their absolute values.

Studied Scenario. By using simulations we can easily evaluate several years of a system, however when doing experimentation this is not the case. We need to plan our experiments to last a few hours. Hence, we define an *acceleration factor*, as the ratio between experiment duration and the time of real system we want to imitate. Our goal is to check the bandwidth congestion in a real environment. Thus, we decided to shrink the disk size (e.g., from 10 gigabytes to 100 megabytes, a reduction of 100×), inducing a much smaller time to repair a failed disk. Then, the peer failure rate is increased (from months to a few hours) to keep the ratio between disk failures and repair time proportional. The bandwidth limit value, however, is kept close to the one of a "real" system. The idea is to avoid inducing strange behaviors due to very small packets being transmitted in the network.

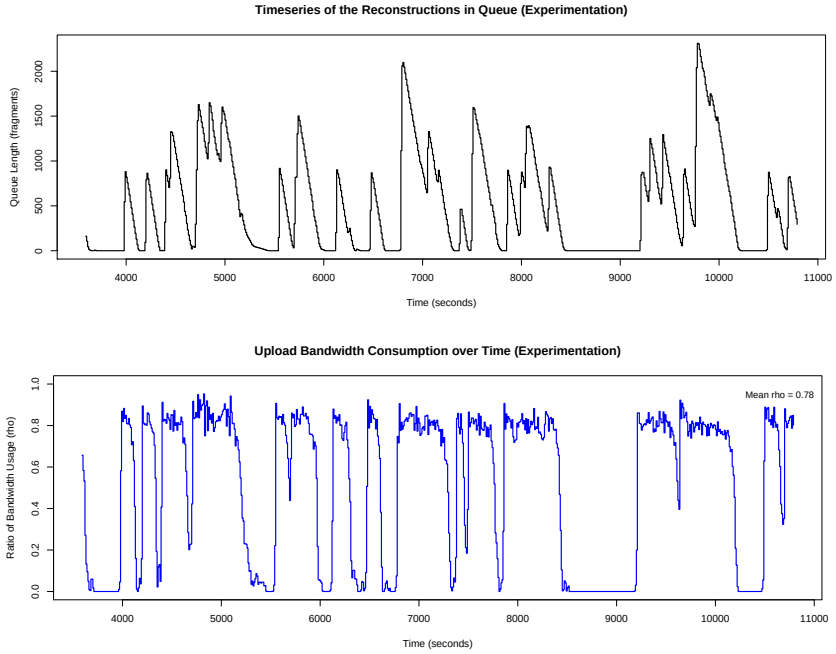


Figure 5.11: Timeseries of the queue size during time (top) and the upload bandwidth ratio (bottom).

Figure 5.10 presents the distribution of the reconstruction times for two different experimentation involving 64 nodes on 2 different sites of GRID'5000. The amount of data per node is 100 MB (disk capacity 120MB), the upload bandwidth 128 KBps, $s = 4$, $r = 4$, $L_F = 128$ KB. We confirm that the simulator gives results very close to the one obtained by experimentation. The average value of reconstruction time differs from some seconds.

Moreover, to have an intuition of the system dynamics over time, in Figure 5.11 we present a timeseries of the number of blocks in the queues (top plot) and the total upload bandwidth consumption (bottom plot). We note that the rate of reconstructions (the

descending lines on the top plot) follows an almost linear shape. Comforting our claim that a determinist processing time of blocks could be assumed. In these experiments the disk size factor is $x = 1.2$, which gives a theoretical efficiency of 0.83. We can observe that in practice, the factor of bandwidth utilization, ρ, is very close to this value (value of $\rho = 0.78$ in the bottom plot).

5.6 Conclusion

In this chapter, we propose and analyze a new Markovian analytical model to model the repair process of distributed storage systems. This model takes into account the correlation between data repairs that compete for the same bandwidth. We bring to light the impact of peer heterogeneity on the system efficiency. The model is validated by simulation and by real experiments on the GRID'5000 PLATFORM.

We show that the exponential distribution classically taken to model the reconstruction time is valid for certain sets of parameters, but that different shapes of distribution appear for other parameters. We show that it is not enough to be able to estimate the tail of the repair time distribution to obtain a good estimate of the system loss rate.

The results provided are for systems using Regenerating Codes that are the best codes known for bandwidth efficiency, but the model is general and can be adapted to other codes. We exhibit an interesting phenomena to keep in mind when choosing the code parameter: it is useful to keep a degree of freedom on the choice of the users participating in the repair process so that loaded or deficient users do not slow down the repair process, even if it means less efficient codes.

In addition, we confirm the strong impact of scheduling on the system loss rate.

CHAPTER 6

Placement Policies

In a distributed storage system using erasure codes, a data block is encoded in many redundancy fragments. These fragments are then sent to distinct peers of the network. In this chapter, we study the impact of different placement policies of these fragments on the performance of storage systems. Several practical factors (easier control, software reuse, latency) tend to favor data placement strategies that preserve some degree of locality. We compare three policies: two of them are *local*, in which the data are stored in logical neighbors, and the other one, *global*, in which the data are spread randomly in the whole system. We focus on the study of the probability to lose a data block and the bandwidth consumption to maintain such redundancy. We use simulations to show that, without resource constraints, the average values are the same no matter which placement policy is used. However, the variations in the use of bandwidth are much more bursty under the *local* policies. When the bandwidth is limited, these bursty variations induce longer maintenance time and henceforth a higher risk of data loss. We then show that a suitable degree of locality could be introduced in order to combine the efficiency of the global policy with the practical advantages of a local placement. Additionally, we propose a new *external reconstruction* strategy that greatly improves the performance of local placement strategies. Finally, we give analytical methods to estimate the mean time to the occurrence of data loss for the three policies.

The results presented in this chapter appeared in [2] and were published in [9, 1].

Our Contribution

As far as we know, we present the first practical study of data placement for systems using Erasure Codes. We show that, even for *local* policies, the erasure codes experience less data loss than replication schemes with the same resources.

We show that, *without bandwidth constraints*, the distribution of the bandwidth usage among peers is much smoother for the Global policy. Despite that all policies have the same average bandwidth consumption and probability of data loss, the mean time between data loss events is longer for the *local* policies. We provide then analytical methods to estimate this metric for the three policies.

When *limiting the maximum available bandwidth* per peer, we exhibit that the Global policy experiences a lot less data loss than the *local* policies for similar resources. In addition, the loss events for *local* policies are more frequent compared to the provisioning scenario (in certain cases even more frequent than for the Global).

We then discuss the size of the leafset in the *local* policies. We show that these policies can be adapted to achieve performances close to the *Global* placement, while keeping

the practical advantages of locality. We propose a new reconstruction scheme, namely *external reconstruction*, which reduces by 40 to 50 percent the number of block losses when using the *local* policies.

Related Work.

The majority of existing or proposed systems, e.g. Intermemory [56], Farsite [23], CFS [35], PAST [44], Glacier [59], use a local placement policy. For example, in PAST, the authors use the Pastry DHT to store replicas of data into logical neighbors. In the opposite way, some systems use a Global policy, as OceanStore [69], pStore [17], Total-Recall [20] or GFS [55]. GFS spreads chunks of data on any server of the system using a pseudo-random placement.

Chun et al. in [28] also discuss the impacts of local placement (namely *small scope*). They state that local placement is easy to maintain but induces higher reconstruction times. Conversely, larger scope (Global policy) has lower reconstruction time and henceforth higher durability. However, they do not address the impact of different bandwidth limits, neither erasure codes redundancy.

Ktari et al., in [68], discuss the impact of data placement. They do a practical study of a large number of placement policies for a system with high churn. They exhibit differences of performance in terms of delay, control overhead, success rate, and overlay route length.

Lian et al., in n [73] study the impact of data placement on the Mean Time to Data Loss (MTTDL) metric, but for a system based of simple replication. They show that the MTTDL is lower for the Global policy (called random placement) when compared to the local policy (called sequence). But they do not discuss other very important metrics: the probability to lose a block and the bandwidth usage.

There are also other studies that evaluate the replica placement, however with focus on the lookup latency and/or throughput performance [122]. Others are focused on Content Delivery Networks [66], which is not our case here.

Organization

The remainder of this chapter is organized as follows: in the next section we present the different placement policies, followed by the details of the simulator in Section 6.2. Then, in Section 6.3 we present our results. Section 6.3.1 we study the behavior of the system without resource constraints, and then under bandwidth constraints in Section 6.3.2. Finally, we propose some improvements of the placement and reconstruction architectures in Section 6.4, followed by analytical methods to estimate the MTTDL metric for the three placements in Section 6.5.

6.1 Description

In distributed storage systems using *erasure codes*, each block of data is divided into s fragments. Then, r fragments of redundancy are added in such a way that any subset of s fragments from the $s + r$ fragments are sufficient to regenerate the data block. The $s + r$ fragments are then sent to different peers of the network. The different placement policies studied in this chapter are detailed in the following and depicted in Figure 6.1.

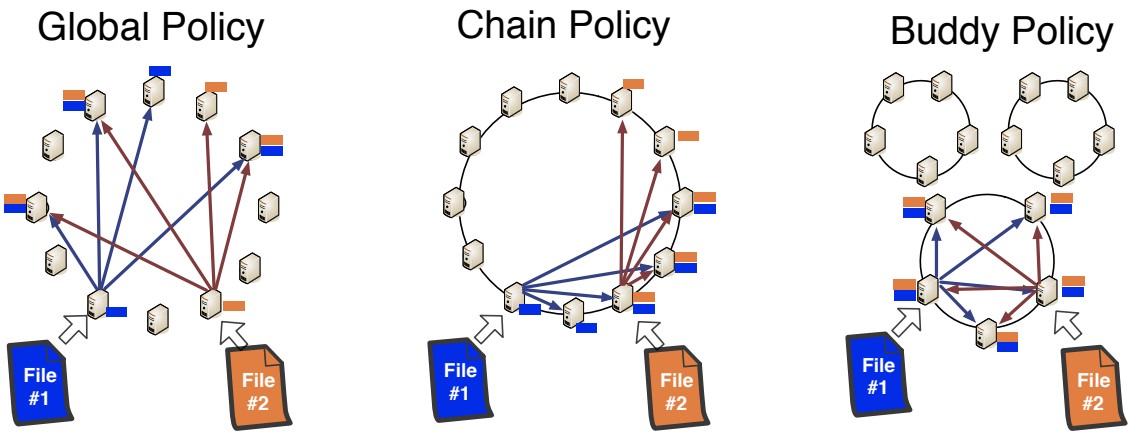


Figure 6.1: Placement of two blocks b_1 and b_2 in the system. Global: $s + r$ fragments are placed at random among all peers; Chain: fragments are placed on $s + r$ neighboring peers; Buddy: many small subsystems of size $s + r$, in this case all peers inside each small group contain the same data.

Global Policy

In the Global policy, the $s + r$ fragments of a block are sent to $s + r$ peers chosen uniformly at random *among all the N peers* present in the system. In this case, the peer in charge of monitoring the state of a block and reconstructing it is also selected among all peers in the system.

Chain (or Neighborhood) Policy

In the Chain, which is a *local* policy, the network is seen as a directed ring of N peers. The fragments are then sent to $s + r$ consecutive peers chosen uniformly at random among the N possible sequences of $s + r$ peers. This policy corresponds to what is done in most distributed systems implementing a DHT. That is, the peer responsible for a data block stores the blocks' fragments on its closest neighbors. Note that these neighbors are also in charge of monitoring and reconstructing these blocks.

Buddy (or RAID) Policy

The Buddy is an extreme case of a *local* policy, in which the peers are divided into C independent clusters of size exactly $s + r$ each. The fragments are then sent to a cluster chosen uniformly at random among the clusters. In this situation, all peers of a cluster store fragments of the same set of blocks. It could be seen as a collection of local RAID like storage.

6.2 Simulations

To evaluate these policies, we use the same cycle-based simulator as the one described in Chapter 3 (Section 3.3). In addition, we implemented the different placements policies as described above. These policies are used in the initialization phase of the simulation, in the choice of the reconstructor peer, and when the missing fragments are redistributed.

Furthermore, in this chapter the reconstruction time is not modeled by a tunable parameter (as the parameter θ in Chapter 3). In fact, this study comprises a more refined modeling of the storage system, in which the repair time is a consequence of the available bandwidth of peers (as modeled in Chapter 5). The peer bandwidth is modeled as follows.

Bandwidth Queue model

We model the bandwidth as a resource constraint per peer, and not as a global shared link constraint as is done in [73, 98]. Each peer has a maximum upload and download bandwidth, resp. BW_{up} and BW_{down}. We assume asymmetric capacities, as often encountered in practice, e.g. ADSL lines (in our experiments $BW_{down} = 5BW_{up}$). So the limiting resource is the upload bandwidth and it is the one presented in our results.

To model bandwidth, we implemented a *non blocking FIFO queue with one server*. When there is a peer failure, the blocks to be reconstructed are put in the queue. The simulator processes these blocks at each cycle (*FIFO order*). For each block it tries to retrieve the fragments, then resends the redundancy fragments to different peers (if the corresponding peers has still some available bandwidth). If some fragments are not available (i.e. peers storing it already reached the maximum uploading capacity), then it tries to download the fragments of the next block in the queue (*non blocking policy*).

Monitored metrics

We focus our analysis on three main metrics:

- *Bandwidth*: Average bandwidth consumption per peer, i.e., estimated from the number of fragments transmitted and received per hour due to the reconstruction process;
- *FDLPY*: Fraction of Data Loss Per Year, which gives the probability of losing a data-block per year;

Table 6.1: Summary of the main notations and their default values in our experiments. See Remarks 1 and 2 for the choice of the parameter values.

N	Number of peers	1,005
B	Total number of blocks in the system	10^5
s	Number of initial fragments of a block	9
r	Number of redundancy fragments	6
r_0	Reconstruction threshold value	2
$MTBF$	Peer mean time to failure	90 days
BW_{up}	Upload bandwidth per peer	6-18 kbit/s and Unlim.
τ	Time step of the model	1 hour

- *MTTDL*: Mean Time To Data Loss, i.e., the period of time between two occurrences of data loss in the system.

Simulation parameters

We did a large number of simulations for different sets of the parameters. The default parameter values used in the simulations are given in Table 6.1, otherwise they are explicitly indicated. The considered size of fragment is $L_f = 400$ KB. With $s = 9$ and $r = 6$, the original data block size $L_b = 3.6$ MB, and the total size with redundancy is 6 MB. Two remarks on the choice of the parameter values:

Remark 1 (Size of the simulated system): In practice, peers have huge disks of tens of Gigabytes, each one containing tens of thousands of blocks. Furthermore a real system with 1000 peers would deal with tens of millions of fragments. As we want to be able to simulate a storage system for several years in a reasonable time, we chose a disk size around 100 times smaller than the one expected in practice. Each peer stores only 1500 fragments in the system, which still corresponds to a total of 1.5 millions of fragments and 571GB of data. Note that the upload bandwidth (BW_{up} spans from 6 to 18 kbit/s in our experiments) directly derives from this choice: disks containing 100 times more data would need a peer bandwidth 100 times larger to maintain the redundancy, that is already in order of Mbits/s and close to the bandwidth limits encountered in practice.

Remark 2 (Measuring block losses): The parameters of real systems are set in such a way that the occurrence of a data loss is a very rare event. As it is impossible to simulate in a reasonable time events of very low probability, for example 10^{-15}, we choose *non realistic values* for some parameters (in particular, the reconstruction saddle $r_0 = 2$ is set very low). In this way, we experience data loss in our simulations. Also for this reason, some of our simulations correspond to 200 simulated years (with time granularity of 1 hour).

Of course, real systems would have a completely different data loss rate than the one reported here for the sake of comparison.

6.3 Results

In this section, we evaluate the three data placement policies for the three following metrics: use of bandwidth, number of dead blocks, and mean time to data loss. First, we study the provisioning scenario (*unlimited bandwidth*) in Section 6.3.1, which is important to measure the required bandwidth to maintain the system. Afterwards, we analyze scenarios with *constrained resources*, in Section 6.3.2.

6.3.1 Without Resource Constraints

Briefly, the results shown here are: (1) the three placement strategies have the same value of average bandwidth demand; (2) however *local* policies exhibit strong variations in resource usage across peers; (3) they have the same data loss rate, (4) but the MTTDLs of the Buddy and the Chain policies are longer.

In our modeling, the unlimited bandwidth scenario means that the system can process all the reconstructions in one time step of the simulator (set to one hour). A summary of the simulation results are presented in Table 6.2. We then discuss: first, the bandwidth consumption; then, the data loss rate; and finally, the mean time do data loss.

Table 6.2: Summary of results (without bandwidth constraints).

Policy	Bandwidth (kbit/s)	FDLPY (% of blocks)	MTTDL (years)
Global	1.99 (± 1.34)	$4.1 \cdot 10^{-2}$ $(\pm 0.6 \cdot 10^{-2})$	0.02 (± 0.02)
Chain	1.99 (± 12.83)	$4.1 \cdot 10^{-2}$ $(\pm 8.6 \cdot 10^{-2})$	4.0 (± 3.0)
Buddy	1.99 (± 15.92)	$4.4 \cdot 10^{-2}$ $(\pm 25.4 \cdot 10^{-2})$	25.8 (± 21.7)

Average Bandwidth Usage

The left column of Table 6.2 shows the average value of upload bandwidth usage across peers during time (i.e., at each time step we measure the average number of fragments transmitted by each peer), along with the experimental standard deviation (in parenthesis).

First, as expected, the average bandwidth use across peers is roughly the same for all policies, 1.99 kbit/s. The reason is that the different placement policies do not change the number of fragments that have to be reconstructed.

However, the variations are not the same, because the policies change the repartition of the pieces among peers. The Chain policy and Buddy policy variations are significantly higher, respectively, 9.5 and 11.8 times more than the Global policy. Figure 6.2 gives an explanation of this behavior. It depicts the bandwidth usage of the 1005 users of the system at a typical instant of time for the three different policies under the same failure scenario. We see that the load is around 2 kbit/s for all the users and all strategies. However, we see that the distributions of the bandwidth are not the same at all.

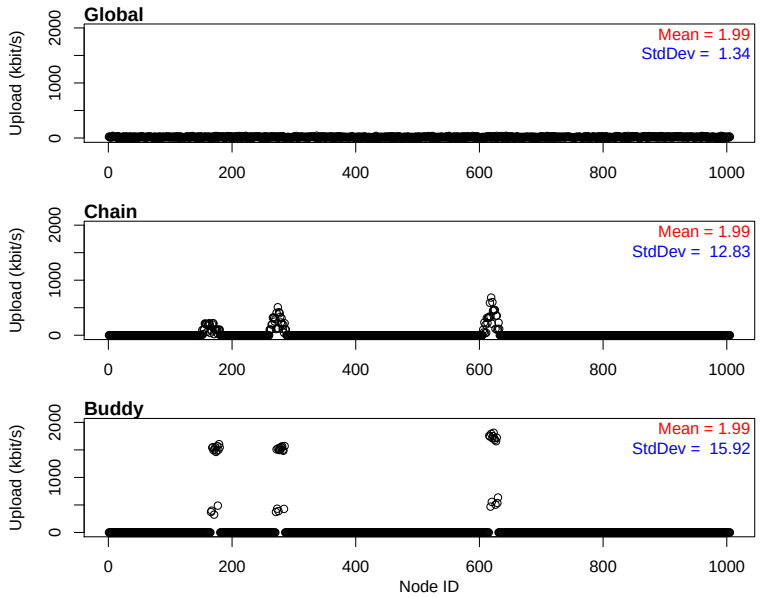


Figure 6.2: Variations of bandwidth usage across users for the three different policies for a typical time step and under the same failure scenario.

In the case of the Global policy (top graph), the fragments of the blocks are placed among all the peers in the system. Consequently the load of the retrieval phase of the reconstruction is uniformly distributed among all peers. Furthermore, the peers in charge of a block reconstruction are also randomly chosen among all peers. So the sending phase of the reconstruction is also evenly distributed. In the example, the standard deviation of the bandwidth of the Global is 1.34 kbit/s.

On the opposite (bottom graph), in the Buddy setting, some groups of $s + r$ users have a very high bandwidth demand, e.g. around 1500 kbit/s. We can identify three groups that correspond to three different sets of peer crashes that triggered reconstructions. In the Buddy policy (similarly to RAID systems), when a failure happens, only the immediate neighbors possess the remaining fragments of the blocks. Moreover, these neighbors are also in charge of the reconstructions, leading to a very high bandwidth load on these peers, while the others peers in the system are spared[1].

The situation for the Chain policy (middle graph) is similar to the Buddy. We also observe three spikes for certain subsets of users. But, differently, these spikes (1) involve more peers and (2) have shapes of pyramids. It is explained as follows. (1) A peer stores fragments of blocks that are managed by peers at distance $s + r$ (chain size). In addition, a block reconstruction affects peers at distance $s + r$. Hence, when a peer crashes, peers at distance $2(s + r) - 1$ contribute to the reconstruction. (2) The spikes correspond to

[1]There are two groups of peers in each spike of the Buddy. A bigger one around 1500 kbit/s, that corresponds to peers doing the retrieval and sending phases of the reconstruction (i.e., $s + r - r_0$ uploads for each block). The smaller one, with an upload bandwidth around 400 kbit/s, correspond to peers that have failed and were replaced with empty disks. As they are empty, they do not send fragments to the reconstructors (no retrieval upload), but they are in charge of some reconstructions, so we see their sending upload (i.e. $r - r0$ fragments for each block).

multiple disk failures. In this scenario, peers close to several failed peers contribute more than peers close to a single failed peer. Hence, the pyramid shaped spikes. To conclude, we see that a peer failure is a quite *big local event* for the two local policies.

Data loss rate

We then compare the probability of losing a block in the three different policies. The results are shown in the middle column of the Table 6.2, normalized as the Fraction of Data Loss Per Year (FDLPY).

When there is no bandwidth limit, the *expected number of dead blocks is the same for the three policies* (roughly 0.04% of blocks lost per year). As a matter of fact, the probability for a block to die does not depend on where its fragments are placed. It can be easily calculated using a Markov Chain Model, as explained in Chapter 3. Note that the *deviations* during time of the number of dead blocks is higher for *local* policies. This is further explained by looking at the MTTDL metric.

Mean Time To Data Loss

The measure of the time between two occurrences of data loss shows that the three policies have very distinct behaviors, as depicted in the right column of Table 6.2. In our simulations, the Global policy loses a data-block every 9 days, the Chain policy every 4 years and Buddy every 25.8 years. However, as we have seen before, the three policies have in average the same number of dead blocks per year. In other words, the average quantity of data loss per year is the same, but the distribution across time of these losses is very different.

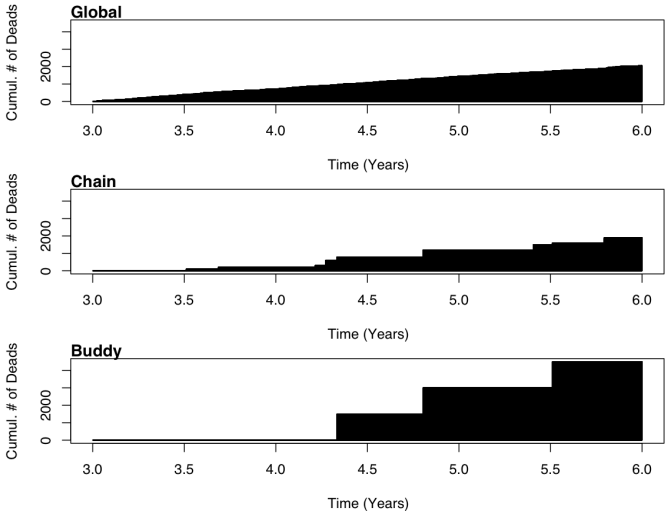


Figure 6.3: Illustrative example of the cumulative number of dead blocks for a period of three years.

The Figure 6.3 illustrates an example of the cumulative number of dead blocks for a period of 3 years (this example uses a different set of parameters with increased failure

rate). The same failure scenario of peers is used in the three placement policies. We see that the loss occurs regularly for the Global policy. Conversely, they occur very rarely for the Buddy placement, but, when they occur, they affect a large batch of data. Basically, all the blocks of a small buddy subsystem of size $s + r$ peers lose all their blocks at the same time. The behavior of the Chain policy is somewhere in the middle of both.

Also, due to its very large variations of behavior, the buddy policy has the drawback of being not very predictable. We see in Figure 6.3 that the Global and the Chain policies experienced around 2,000 block losses after 6 years, when the Buddy policy experienced almost 4,000. Even if they have the same probability of data loss. In a long-run, the number of expected losses are the same.

6.3.2 Results under Resource Constraints

In this section, we study the behavior of the system with bandwidth limitation per peer (meaning that now each peer has a maximum upload and download bandwidth). In this context we show that, using similar available resources, the amount of data loss is no more the same for the three data placement policies. The Global policy behaves considerably better in comparison to the Chain and Buddy policy. Furthermore, the *local* policies now experience more loss events (smaller MTTDL).

Reconstruction Time versus Bandwidth

Figure 6.4 gives the average reconstruction time for different upload bandwidth limits BW_{up}, ranging from 6 kbit/s to 18 kbit/s. The unlimited bandwidth value is given for the sake of comparison. The value of 1 is the time granularity of our simulator (one hour).

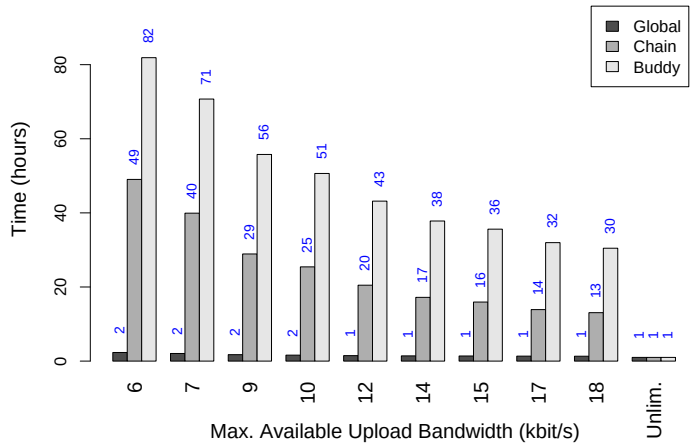


Figure 6.4: *Average reconstruction time* for different bandwidth limits for the three strategies.

We see that the average reconstruction time is a lot longer for the Chain policy and even more for the Buddy policy compared to the Global one. As an example, for a maintenance bandwidth of 6 kbit/s, the reconstruction time is around 49 hours for the Chain policy and 82 hours for the Buddy, but only 2 hours for the Global policy. This bandwidth limit corresponds to three times the average bandwidth usage of the system (as computed without resource constraints). Hence, we see that the imbalance of the reconstruction load among peers has a very strong impact on the reconstruction time, even if each policy has the same average bandwidth demand. For a bandwidth limit of 18 kbit/s, which represents *9 times the average bandwidth* needed, the difference is still very large: 1, 14, and 30 times for the Global, Chain and Buddy, respectively. Thus, under resource constraints, the big local events constituted by peer failures induce longer reconstruction time and henceforth an increase of data loss when using the *local* policies, as shown in the following.

FDLPY versus Bandwidth

A critical performance measure of a P2P storage architecture is the probability of losing a block for a given amount of bandwidth. Figure 6.5 compares the trade-offs of the three policies for different values of BW_{up}. The following table summarizes the values of fraction of data loss per year for the three policies under resource constraints.

FDLPY (%)

Max.BW	6	9	12	15	18	Unlim.
Global	0.05 $\pm$0.01	0.05 $\pm$0.01	0.04 $\pm$0.01	0.04 $\pm$0.01	0.04 $\pm$0.01	0.04 $\pm$0.01
Chain	10.6 $\pm$1.3	3.5 $\pm$0.8	1.8 $\pm$0.5	1.1 $\pm$0.4	0.8 $\pm$0.3	0.04 $\pm$0.09
Buddy	26.0 $\pm$3.5	12.5 $\pm$2.5	7.1 $\pm$1.9	4.6 $\pm$1.6	3.2 $\pm$1.4	0.04 $\pm$0.25

We see that the Global policy behaves a lot better for any bandwidth limit than the Chain policy, which itself is more efficient than the Buddy policy. For example, for a bandwidth limit of 18 kbit/s (which represents 9 times the average bandwidth need of the system), the Global experiences 0.04% of data loss per year, to compare with 0.8% and 3.2% for the Chain and the Buddy, respectively.

MTTDL versus Bandwidth

Opposed to what was showed without bandwidth constraints, the Global policy behaves better than the others with low bandwidth limitations. The following table shows the MTTDL for the three policies under resource constraints.

MTTDL (hours)

Max.BW	6	9	12	15	18	Unlim.
Global	166 $\pm$184	180 $\pm$188	193 $\pm$223	219 $\pm$230	220 $\pm$239	215 $\pm$235
Chain	53 $\pm$69	160 $\pm$221	323 $\pm$422	565 $\pm$600	850 $\pm$900	35,143 $\pm$26,635
Buddy	75 $\pm$74.9	178 $\pm$176	341 $\pm$337	570 $\pm$510	860 $\pm$855	226,000 $\pm$190,424

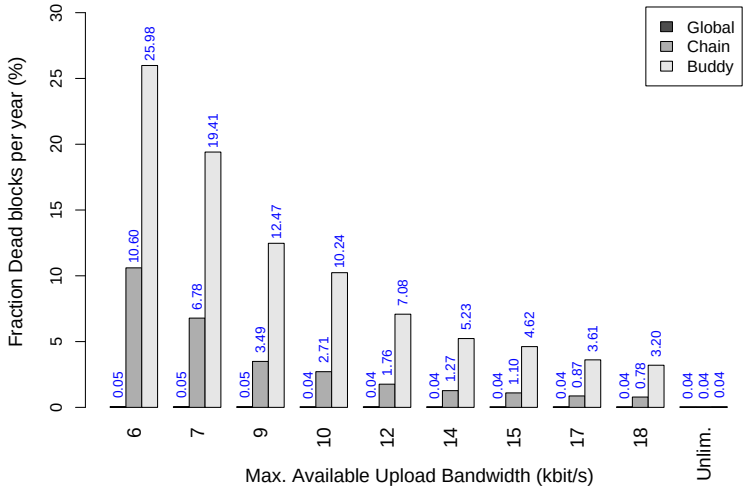


Figure 6.5: *Fraction of block losses per year* (see Remark 2) for different bandwidth limits for the three strategies.

For comparison, without resource constraints, the time between data loss were 0.02, 4.0, and 25.8 years respectively for the Global, Chain and Buddy. Conversely, with an available bandwidth of 6 kbit/s, these values are 166, 53, and 75 (in hours), many orders of magnitude less. These results show that the impact of the bandwidth limits per peer needs to be taken into account when analyzing such systems.

6.4 Proposition for P2P Storage System Architectures

In this section, we suggest some modifications of the architecture of systems implementing *local* policies. We also discuss the best choice of their parameters. First, we propose an *external reconstruction strategy* for the *local* policies, and show that it can lower the duration of the sending phase of reconstructions, and thus reduce the probability of data loss. Second, we show that having a larger neighborhood is sufficient to greatly improve the Chain policy performance. Hence, an architecture with the advantage of locality and performance close to the ones of a Global strategy can be obtained. Finally, we carry out some comparisons between Replication and Erasure Code schemes. We show by simulations that, for the same amount of bandwidth and space overhead, the Erasure Codes are better even for the Chain policy.

6.4.1 External Reconstruction Strategy

We propose here a new reconstruction architecture for the Chain policy, namely *external reconstruction*. The idea is to use peers outside the Chain group to carry out the reconstruction process. In this way, the bandwidth usage is more uniformly spread among

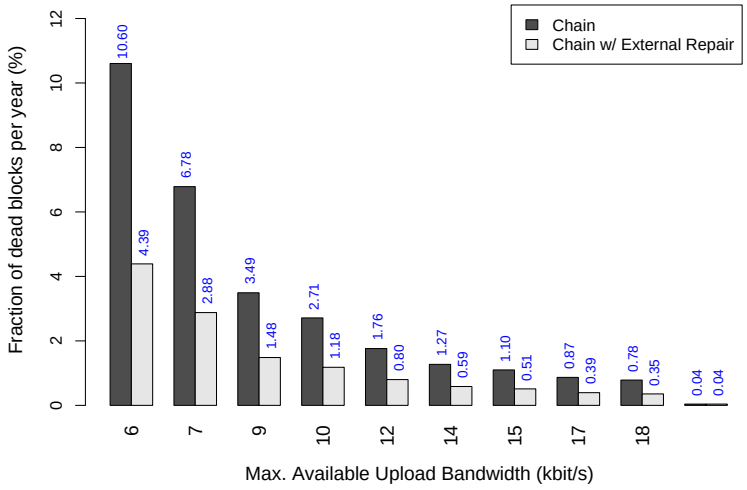


Figure 6.6: Comparison between the Chain policy with internal reconstruction and with external reconstruction. The fraction of block losses (see Remark 2) for different bandwidth limits is presented. There is an improvement of about 50% on the fraction of data losses.

peers. More precisely, only the upload bandwidth of the retrieval phase of the reconstruction is needed locally, while the bandwidth for the sending phase is provided by all the peers of the system. Hence, the *External Reconstruction* has two main advantages:

- a local control for discovering failed peers and updating the data-blocks' states;
- a more uniform distribution of resources among peers, which lowers the reconstruction time.

However, a small cost is paid: in the internal reconstruction, the peer in charge may be chosen in such a way that it possesses a piece of the block to be reconstructed. It reduces by a factor of $(s-1)/s$ the bandwidth needed for the retrieval phase of the reconstruction. Conversely, in the external reconstruction, the reconstructor does not contain any piece.

A rough estimate of the gain in terms of reconstruction time can be given. In the internal reconstruction, the *local peers* have to upload $(s-1)+(r-r_0)$ fragments, noted as $u_{internal}$. However, when using the external reconstruction they only have to upload s fragments, noted as $u_{external}$. As the local peers are the bottleneck of the reconstruction, the gains in terms of bandwidth and hence of reconstruction time are roughly $1-s/(s-1+r-r_0)$ (it comes from the ratio $1-u_{external}/u_{internal}$). With the parameters chosen in our experiments, this factor would be 0.25. Note that the gains in terms of data loss will be significantly higher (see Section 6.4.2).

Figure 6.6 compares the internal and external policies. It gives the trade-off between the average number of dead blocks per year and the available bandwidth. For the same

bandwidth, the fraction of data loss decreases by a factor between 0.5 and 0.6 for this set of parameters. We see that, by choosing to carry out the reconstructions externally, the chain policy behaves substantially better.

6.4.2 What Should Be the Size of the Neighborhood?

We showed above that the Global policy in practice (under tight resource constraints) behaves significantly better than the local policies. Nevertheless, as already stated in the introduction, there exist important practical considerations that explain the choice of *local* placement. Would it be possible to obtain the same practical advantages as the local policies (a small sub-network to monitor) but without paying the high cost of the Chain and Buddy in terms of probability of data loss?

In this section, we study the impact of the *size of the block neighborhood* on the system performance. The block neighborhood is defined as the peers that can receive fragments of this block (of size $s + r$ for Buddy and Chain, and of size N for Global).

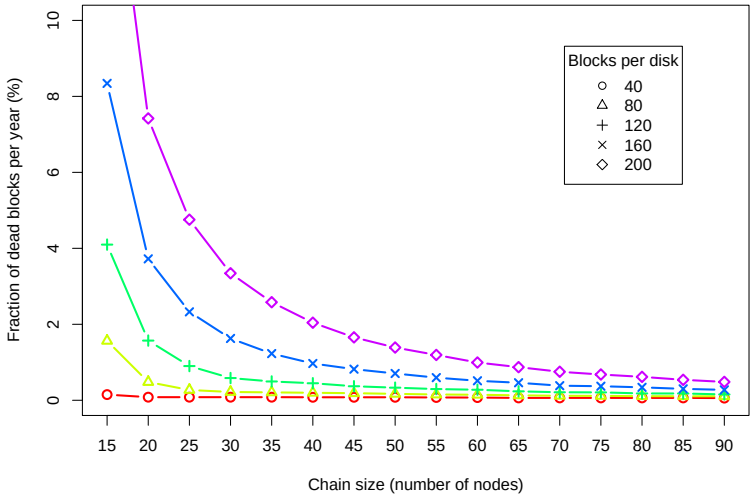


Figure 6.7: Study of the size of the block neighborhood. Fraction of block losses (see Remark 2) per year for different sizes of neighborhood and different number of fragments per disks.

Figure 6.7 shows the average number of dead blocks per year for different sizes of the neighborhood. The sizes range from $s + r = 15$ (corresponding to the size of the neighborhood for the Chain policy) to 90. The experiment was done for different amounts of data per disk (i.e., number of blocks per disk), from 40 to 200, which is, as we will see, an important parameter when choosing the neighborhood size. We see that barely increasing the neighborhood from 15 to 20 has a striking impact on the data loss: with 120 blocks per disk, the fraction of data loss drops from 4.1% to 1.6%, representing a decrease of almost 61 percent. Thus, increasing the size even by few units leads to strong improve-

ments of the system performance. However, the number of dead blocks decreases from 0.17% to 0.16% for neighborhoods of sizes 85 and 90 nodes. The marginal improvement strongly decreases.

The shapes of the curves may be explained as follows. When there is a peer failure, its neighbors are in charge of reconstructing the lost fragments. Hence the reconstruction time (minus the discovery time) depends almost linearly on the neighborhood size. This dependence can be directly translated to the probability data loss:

Exponential relation between the probability to die and the reconstruction time. During a reconstruction, a block dies if it loses $r_0 + 1$ fragments before it finishes. The probability for a peer to be alive after a time T is $\exp(-\lambda T)$, where λ is the peer failure rate. Hence a good approximation of the probability to die during a reconstruction lasting a time T is given by

$$\Pr[die|Rtime = T] = \binom{s + r_0}{r_0 + 1}(1 - e^{-\lambda T})^{r_0+1}(e^{-\lambda T})^{s-1}.$$

Hence we have an exponential relationship between the number of block losses and the neighborhood size.

The neighborhood size should mainly be chosen in function of two parameters: the disk size (or the number of fragments per disk) and the peer bandwidth. Note that a size of $\frac{D}{(r-r_0)BW_{up}}$ allows to reconstruct the blocks in one time step and is sufficient to get the benefits of Global (with D the number of fragments per disk, BW_{up} expressed in blocks/time step and $1/(r - r_0)$ the fraction of blocks of the lost disk that go beyond the saddle value).

Concluding, to implement a local policy, the neighborhood should at least be a little bit larger than $s + r$, as the marginal utility of increasing the block neighborhood is tremendous for very small sizes. In addition, the neighborhood size should be chosen in function of the disk size: The larger the number of fragments per disk, the larger the block neighborhood should be.

6.4.3 Replication versus Erasure Codes

Other experimental studies on data placement analyze the case of replication instead of Erasure Codes, see e.g. [121, 43, 35, 73]. It is shown in [125] that Erasure Codes could be used to achieve a high availability of data storage with low space overhead, but these studies assume a Global and random placement strategy. We show here that, *even for local policies*, the erasure codes scheme is more efficient than replication, meaning that it has less probability of losing data for the same storage and bandwidth usage. Hence, we confirm the pertinence to carry out an analysis of data placement when using erasure codes.

Note first that replication is a special case of erasure codes, but with only one initial fragment ($s = 1$). Hence, we used exactly the same simulator to carry out the experiments. We evaluated the system for different number of replicas k, with values varying

from 1 to 4. To have the same storage overhead factor, we compare the scenarios using k replicas with a system with $r = k{\cdot}s$ erasure coded fragments. The reconstruction threshold value is set to $r_0 = r - s$ ($k_0 = k - 1$ to the case of replication), this way both schemes experience a close number of reconstructions, that is, roughly the same bandwidth usage. We present in Table 6.3 the average bandwidth use and fraction of data loss per year for both techniques. In the case of erasure codes, the number of initial fragments is $s = 4$. For a value of replication $k = 3$ (this means $r = 12$ to the case of erasure codes), the reconstruction starts when the number of available fragments of a block is 2 (and when it is 8 for the erasure codes).

Table 6.3: Comparison of replication and erasure codes when using the Chain placement. Number of dead blocks per year and average bandwidth usage for different values of redundancy k.

Estimated Fraction of Data Loss per Year (%)

$k =$	1	2	3	4
Repl.	$2.2 \cdot 10^{1}$	$3.0 \cdot 10^{-1}$	$3.8 \cdot 10^{-4}$	$1.9 \cdot 10^{-4}$
Erasure	$6.9 \cdot 10^{1}$	$2.5 \cdot 10^{-6}$	$3.2 \cdot 10^{-17}$	$4.3 \cdot 10^{-29}$

Upload Bandwidth usage (kbit/s)

$k =$	1	2	3	4
Repl.	1.08	3.24	4.34	5.40
Erasure	1.45	2.47	3.42	4.37

In the case of erasure codes $s = 4$ *and* $r = s{\cdot}k$

We see that, for $k = 1$, the system with replicas behaves better than erasure codes, while using less bandwidth. But, as soon as $k \geq 2$, systems with erasure codes behave strikingly better: they experience a lot fewer block losses while using a little less bandwidth. In practice, to have a very low probability of data loss, real systems use values of k larger than 4, see e.g. [121]. Thus, systems with erasure codes have less probability of losing data for the same amount of resources and realistic levels of redundancy.

6.5 Analytical Estimations of MTTDL

In the last sections it is shown that fragment placement has a strong impact on the system performance. In the following we describe the analytical methods to compute exact values and approximations of the MTTDL for the three policies.

Here we study the case where the reconstruction starts as soon as one of its fragments is lost, namely *eager* reconstruction strategy. In addition, the blocks are reconstructed in one time step, i.e., there is enough bandwidth to process the reconstruction quickly. After the reconstruction, the regenerated missing fragments are spread among different peers. Hence, after each time step, the system is fully reconstructed.

Data Loss Rate. A data loss occurs when at least one block is lost. A block is considered lost if it loses at least $r + 1$ fragments during one time step, otherwise, recall that all the $s + r$ fragments are fully reconstructed at next time step. The data loss rate for a given

block comes straightforward. This loss rate does not depend on the placement policy (as soon as it is assured that all fragments are stored on different peers). Hence, we have the same expected number of lost blocks for the three placement policies. However, as stated in Section 6.3.1, the measure of the time to the first occurrence of data loss shows that the three policies have very distinct behaviors. It is shown by simulations that the average quantity of data loss per year is the same, but the distribution across time of these losses is very different (see Figure 6.3).

In the next three sections (Section 6.5.1, 6.5.2 and 6.5.3), we present methods to compute exact values and approximations of the MTTDL for the three placement policies. For each policy, we calculate the probabilty $\mathbb{P}_{policy}$ of data loss at any given time step. Then, we deduce $MTTDL_{policy} = 1/\mathbb{P}_{policy}$. We start by presenting the calculations for the Buddy policy as it is the simplest one, then we present the Global and Chain in following.

6.5.1 Buddy Placement Policy

In the Buddy placement policy, the N peers are divided into C clusters of size $s+r$ each. In this strategy, the calculation of the MTTDL_{buddy} is straightforward. Given a cluster, the probability to have a block loss is the probability that the cluster loses at least $r+1$ peers (i.e., fragments), is given by

$$\mathbb{P}_{cluster} = \sum_{j=r+1}^{s+r} \binom{s+r}{j} \alpha^j (1-\alpha)^{s+r-j}. \tag{6.1}$$

In fact, when that happens all the data stored on that cluster is lost. Remember that α is the probability of a given peer to fail at one time step. Since all the C clusters are independent, the probability to have a data loss is given by $\mathbb{P}_{buddy} = 1 - (1 - \mathbb{P}_{cluster})^C$.

If the average number of cluster failures per time step $C \cdot \mathbb{P}_{cluster} \ll 1$, as expected in a real system (i.e., the probabilty of simultaneous cluster failures is small), then we have $\mathbb{P}_{buddy} \approx C \cdot \mathbb{P}_{cluster}$, and so $\text{MTTDL}_{buddy} \approx 1/(C \cdot \mathbb{P}_{cluster})$.

If $(s+r)\alpha \ll 1$, we can approximate even more. In other words, this assumption means that the probability of a peer failure α is small. Since the ratio between two consecutive terms in sum of Equation (6.1) is $\leq (s+r)\alpha$, we can bound its tail by a geometric series and see that it is of $O((s+r)\alpha)$. We obtain $\mathbb{P}_{cluster} \approx \binom{s+r}{r+1}\alpha^{r+1}$. Then we have

$$\boxed{\text{MTTDL}_{buddy} \approx \frac{1}{\frac{N}{s+r} \cdot \binom{s+r}{r+1}\alpha^{r+1}}.} \tag{6.2}$$

6.5.2 Global Placement Policy

In the Global policy, block's fragments are parted between $s+r$ peers chosen uniformly at random. First, we present the exact calculation of the MTTDL_{global}. We then present approximated formulas that give an intuition of the system behavior.

First, we consider i failures happening during one time step. Let F denote the set of the placement classes (i.e., groups of $s+r$ peers) that hold at least $r+1$ of these i failures; we have:

$$\#F = \sum_{j=r+1}^{i} \binom{i}{j}\binom{N-i}{s+r-j} \tag{6.3}$$

Then, suppose we insert a new block in the system: his $s+r$ fragments are dispatched randomly in one of the $\binom{N}{s+r}$ placement classes with uniform probability. Thus, the probability $\mathbb{P}_{block}(i)$ for the chosen class to be in F is:

$$\mathbb{P}_{block}(i) := \mathbf{P}\left[\text{placement in } F\right] = \frac{\sum_{j=r+1}^{i}\binom{i}{j}\binom{N-i}{s+r-j}}{\binom{N}{s+r}}$$

As block insertions are *independent*, if we consider our B blocks one after the other, the probability that none of them falls in F is $(1-\mathbb{P}_{block}(i))^B$. We then come back to the global probability of data loss considering different failure scenarii:

$$\begin{aligned}\mathbb{P}_{global} := \mathbf{P}\left[\text{data loss}\right] &= \mathbf{P}\left[\bigcup_{\{i \text{ failures}\}}[\text{failure kills a block}]\right] \\ &= \sum_{i=r+1}^{N}\binom{N}{i}\alpha^i(1-\alpha)^{N-i}\mathbf{P}\left[i \text{ failures kill a block}\right]\end{aligned}$$

Which gives us the MTTDL of the system using the global policy:

$$\text{MTTDL}_{global}^{-1} \approx \sum_{i=r+1}^{N}\binom{N}{i}\alpha^i(1-\alpha)^{N-i}\left[1-\left(1-\frac{\sum_{j=r+1}^{i}\binom{i}{j}\binom{N-i}{s+r-j}}{\binom{N}{s+r}}\right)^B\right] \tag{6.4}$$

Approximation

We provide here approximations for systems with low peer failure rates, as for example *Brick storage systems* [73]. Each peer is a "brick" dedicated to data storage, that is, a stripped down computer with the fewest possible components: CPU, motherboard, hard drive and network card. In these storage systems, we have either $\alpha N \ll 1$ or $\alpha N \sim 1$, i.e., we have a not too high mean number of peer failures per time step. If we simplify it further, we find (the details are in [1]):

$$\boxed{\text{MTTDL}_{global} \approx \frac{1}{B\binom{s+r}{r+1}\alpha^{r+1}}} \tag{6.5}$$

6.5.3 Chain Placement Policy

For the Chain policy, the computation of MTTDL_{chain} is more difficult than the two previous ones, mainly because the chains are not independent of each other. From the definition of the Chain policy, a data loss occurs only when $r+1$ (or more) peer failures are

located at $s + r$ consecutive peers.

We present in this chapter two approaches to compute or approximate the MTTDL for the Chain policy. We first describe computations using Markov chains techniques, and we then describe an analytical approximation value assuming that α is small enough.

6.5.3.1 Markov Chain Approach

The idea is to survey the N sequences $S_1, S_2, \ldots, S_N$ of $s + r$ consecutive peers. First, we define a binary-vector $(b_i, b_{i+1}, \ldots, b_{i+s+r-1})$ for each S_i, where the elements of this vector represent the state of peers of S_i: $b_j = 1$ if the peer numbered j is failed, $b_j = 0$ otherwise, $i \leq j < i + s + r$. Peer numbered $N + k$ is really the peer numbered k. Remark that the binary-vector of S_{i+1} is $(b_{i+1}, \ldots, b_{i+s+r})$.

As an example, consider a system composed of $N = 10$ peers with the values $s = 3$ and $r = 2$. The first sequence S_1 of peers is associated with the vector $(b_1, \ldots, b_5)$. If $\sum_{i=1}^{5} b_i \geq 3$, then it means that there is a data loss. Otherwise we have for example the vector $(0, 0, 1, 0, 0)$. Thus we now look at the vector $(b_2, \ldots, b_6)$ associated with the second sequence S_2 of peers. To get this new vector, we remove the first bit b_1 of the previous vector and we add the new bit b_6 at the end. We get for example $(0, 1, 0, 0, 1)$ if $b_6 = 1$. Two peer failures appear in the sequence S_2, and so we do not have a data loss. If for example $b_7 = 1$, then the vector associated with S_3 is $(1, 0, 0, 1, 1)$. In that case a data loss is found.

We now want to compute the probability to find at least one "bad" sequence S_i containing at least $r + 1$ bits 1 in its vector. We use a discrete time discrete space Markov chain to represent the transitions between sequences. Indeed, the set of states V of such Markov chain is the set of all possible binary-vectors of size $s + r$ such that the sum of its elements is at most r, plus an absorbing state namely v_{dead} (containing all other binary-vectors of size $s + r$ in which the sum of its elements is greater than r). For a binary-vector $(b_i, b_{i+1}, \ldots, b_{i+s+r-1})$, we have two possible transitions: $(b_{i+1}, \ldots, b_{i+s+r-1}, 1)$ with probability α and $(b_{i+1}, \ldots, b_{i+s+r-1}, 0)$ with probability $1 - \alpha$. One of these vectors (states) could belong to v_{dead}. Remark that we can see this Markov chain as a De Bruijn graph [39].

Consider the previous example with $s = 3$ and $r = 2$. Figure 6.8 describes the two possible transitions from the state $(1, 0, 0, 1, 0)$ (corresponding to the current sequence S_i): the last peer of the next sequence S_{i+1} is failed with probability α, and it is not failed with probability $1 - \alpha$. The two possible states are $(0, 0, 1, 0, 1)$ and $(0, 0, 1, 0, 0)$, respectively. Furthermore from state $(0, 0, 1, 0, 1)$, it is possible to transit to state v_{dead} because with probability α the vector of the next sequence is $(0, 1, 0, 1, 1)$.

First, we assume that the N peers are ordered in a *line* instead of a *ring*. In other words we do not take into consideration such vectors of sequences: $(\ldots, b_N, b_1, \ldots)$. In that case we look at $N - (s + r) + 1$ sequences. We compute the distribution of probability π after N steps as follows: $\pi = v_0 M^N$ where $v_0 = (0, 0, \ldots, 0)$ is the state without peer failures and M is the transition matrix of our Markov chain. In that case $\mathbb{P}_{line}$ is $\pi(v_{dead})$.

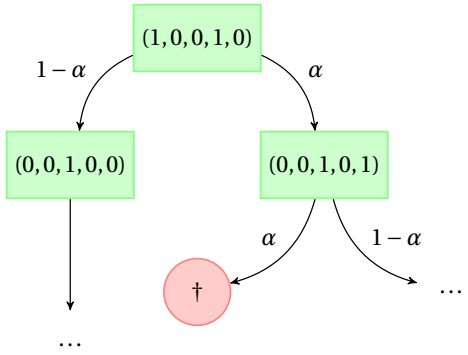


Figure 6.8: Sample part of the Markov chain for $s + r = 5$ and $r + 1 = 3$.

To get the value $\mathbb{P}_{chain}$, we have to carefully take into consideration sequences containing peers on both borders of the network (becoming a ring again). The concerned sequences admit vectors $(\ldots, b_N, b_1, \ldots)$. We get $\pi = \sum_{v \in V} P(v)(v_0 M_{b_{i_1}} \ldots M_{b_{i_{s+r}}} M^{N-(s+r)} M_{b_{i_1}} \ldots M_{b_{i_{s+r-1}}})$ with $P(v)$ the probability to have v as initial state, and M_k, $k \in \{0,1\}$, the transition matrix replacing α by k.

The number of states of the previously described Markov chain is $|V| = 1 + \sum_{i=0}^{r} \binom{s+r}{i}$ states. Lemma 1 proves that we can reduce this number of states showing some properties.

Lemma 1. *There exists a Markov chain having the same $\pi(v_{dead})$ such that:*

$$|V| = 1 + \sum_{i=0}^{r} \binom{s+r}{i} - \sum_{k=1}^{r} \sum_{j=0}^{k-1} \binom{s+k-1}{j} \tag{6.6}$$

Proof. One of the peer failures in the chain is meaningful if and only if it can be present in some following chain containing at least $r + 1$ failures. For example, in the state $(1, 0, \ldots, 0)$, the first dead is not meaningful because, even if we have r dead peers following, it will be too far away to make a chain with $r + 1$ peer failures. In this sense, states $(0, 0, \ldots, 0)$ and $(1, 0, \ldots, 0)$ are equivalent and we can merge them.

More generally suppose we have k peer failures in the current state (current sequence of peers): we miss $r + 1 - k$ peer failures to make a data loss; hence, a peer failure in the current sequence will have incidence if and only if it is one of the last $s + k - 1$ peers of the chain: otherwise, even if the next $r + 1 - k$ peers are dead, they won't fit with our k deads in a frame of size $s + r$.

Thus, among all the states with k peer failures, only those where all failures are in the tail of size $s + k - 1$ are meaningful. As to the others, the first failures do not matter and we can forget them. This merging algorithm leads us to state space size (6.6): in a nutshell, we forget all states with k failures and less than k peer failures in the tail of size $s + k - 1$. □

We presented a method to compute the exact value of $\mathbb{P}_{chain}$ ($\text{MTTDL}_{chain} = 1/\mathbb{P}_{chain}$). We now propose a simple method to approximate the MTTDL using Absorb-

ing Markov chains techniques. We first consider that the number of peers is infinite. In fact peers numbered i, $i+N$, $i+2N$, ..., $i+kN$, ... represent the same peer numbered i but at different time steps. Then the corresponding fundamental matrix gives us the average time t_{abs} to absorption, that is the average number of consecutive sequences of peers to find a data loss. Thus $MTTDL_{chain} \approx \lfloor t_{abs}/N \rfloor$. Indeed let P and Q denote the transition matrices of respectively the complete chain (described before) and the subchain where we removed the absorbing state and all its incident transitions. Then the fundamental matrix $R = (I-Q)^{-1}$ gives us the time to absorption t_{abs} starting from any state (see [106] for details). t_{abs} is not exactly the MTTDL since $N-(s+r)$ steps correspond to one time step (we survey the whole ring). Hence, $\lfloor t_{abs}/N \rfloor$ gives us the expected number of *time* steps before we reach the absorbing state, which is, this time, the MTTDL we are looking for.

6.5.3.2 Analytical Approximation

In the rest of this section, a *syndrome* is a sequence of $s+r$ consecutive peers containing at least $r+1$ peer failures. Under the assumption that α is "small enough" (we will see how much), we can derive an analytical expression of the MTTDL.

$$\boxed{MTTDL_{chain} \approx \frac{1}{N\frac{r+1}{s+r}\binom{s+r}{r+1}\alpha^{r+1}}.} \tag{6.7}$$

Let us begin with two lemmas.

Lemma 2. *The probability to have two distinct syndromes is negligible compared to the probability to have only one and bounded by*

$$\mathbf{P}\left[\exists \textit{ two distinct syndromes} \mid \exists \textit{ a syndrome}\right] < \frac{\alpha N(s+r)\cdot(\alpha(s+r))^{r-1}}{r!} \tag{6.8}$$

Proof. The probability for a syndrome to begin at a given peer (the beginning of a syndrome being considered as his first peer failure) is given by $p = \alpha\sum_{i=r}^{s+r-1}\binom{s+r-1}{i}\alpha^i(1-\alpha)^{s+r-1-i}$. Meanwhile, we have

$$\mathbf{P}\left[\exists \text{ 2 distinct syndromes}\right] = \mathbf{P}\left[\cup_{|i-j|\geq s+r}\exists \text{ 2 syndromes beginning at peers } i \text{ and } j\right],$$

which is $\leq \binom{N}{2}p^2 < (pN)^2$. Normalizing by pN gives us the probability to have two syndromes knowing that there is at least one:

$$\mathbf{P}\left[\exists \text{ two distinct syndromes} \mid \exists \text{ a syndrome}\right] < pN.$$

Hence, we would like to show that pN is negligible. An upper bound on p is easy to figure out: given that $\alpha(s+r) \ll 1$, we have $p \approx \binom{s+r-1}{r}\alpha^r(1-\alpha)^{s-1} \leq (\alpha(s+r))^r/r!$, and so $pN \leq (\alpha N(s+r))(\alpha(s+r))^{r-1}/r!$. Hence, assuming $\alpha N(s+r) \ll 1$ (or otherwise $r \geq \log N$) suffices to conclude. □

Lemma 3. *The probability to have more than $r+1$ dead peers in a given syndrome is negligible and bounded by*

$$\mathbf{P}\left[\exists > r+1 \textit{ dead peers} \mid \exists \geq r+1 \textit{ peers}\right] < \alpha(s+r) \tag{6.9}$$

Proof. Since we are working in a syndrome, the probability we want to bound is, in a given chain:

$$\begin{aligned}\mathbf{P}\left[\exists > \text{r+1 dead peers} \mid \exists \geq \text{r+1 dead peers}\right] &= \frac{\sum_{r+2}^{s+r}\binom{s+r}{i}\alpha^i(1-\alpha)^{s+r-i}}{\sum_{r+1}^{s+r}\binom{s+r}{i}\alpha^i(1-\alpha)^{s+r-i}} \\ &\leq \frac{\sum_{r+2}^{s+r}\binom{s+r}{i}\alpha^i(1-\alpha)^{s+r-i}}{\binom{s+r}{r+1}\alpha^{r+1}(1-\alpha)^{s-1}}\end{aligned}$$

Since the ratio between a term of the binomial series and its predecessor is $\frac{\alpha}{1-\alpha}\cdot\frac{s+r-i}{i+1}$, we can bound the tail of the binomial sum by a geometric series of common ratio $q = \frac{\alpha}{1-\alpha}\cdot\frac{s-1}{s+r} \ll 1$. Thus we have:

$$\mathbf{P}\left[\exists > \text{r+1 dead peers} \mid \exists \geq \text{r+1 dead peers}\right] < \frac{\alpha}{1-\alpha}\cdot\frac{s-1}{r+2}\cdot\frac{1}{1-q} < \alpha(s+r) \ll 1. \quad \square$$

$\square$

Therefore, if we only look for a single syndrome with exactly $r+1$ dead peers, we get a close approximation of the MTTDL.

$$\begin{aligned}\mathbb{P}_{chain} &= \mathbf{P}\left[\exists \text{ one syndrome}\right] \\ &= \mathbf{P}\left[\cup_i \exists \text{ one syndrome beginning at peer } i\right] \\ &= (N-(s+r))p\end{aligned}$$

Indeed, since there is only one syndrome, the events [syndrome begins at peer i] are exclusives. Here p is the probability for the syndrome to begin at a given peer, which we saw in proof of lemma 2. Given lemma 3, we can approximate it by $\binom{s+r-1}{r}\alpha^{r+1}(1-\alpha)^{s-1}$, which leads us too:

$$\text{MTTDL}'_{chain} \approx \frac{1}{N\binom{s+r-1}{r}\alpha^{r+1}} \tag{6.10}$$

One may notice that this is the same formula as (6.2) in the Buddy case with $c = N\frac{r+1}{s+r}$.

Validity of the approximation

Numerical results suggests that the Equation (6.10) is a good approximation for $\alpha < 10^{-3}$, while s have little influence (and r almost none) on the relative variation between simulation and approximation.

6.5.4 Discussion

The approximations given by the Equations (6.2), (6.5), and (6.7) give an interesting insight on the relation between the placement policies. For instance, note that the ratio between MTTDL_{buddy} and MTTDL_{chain} does not depend of N, nor B, nor s. When $B \ll \binom{N}{r+1}$, the ratio between MTTDL_{buddy} and MTTDL_{global} depends on the number of fragments per disk $B(s+r)/N$.

$$\frac{\text{MTTDL}_{buddy}}{\text{MTTDL}_{chain}} \approx r+1, \quad \frac{\text{MTTDL}_{buddy}}{\text{MTTDL}_{global}} \approx \frac{B(s+r)}{N}, \quad \frac{\text{MTTDL}_{chain}}{\text{MTTDL}_{global}} \approx \frac{B(s+r)}{N(r+1)}.$$

We succeeded in quantifying the MTTDL of the three policies. The Buddy policy has the advantage of having a larger MTTDL than the Chain and the Global. However, when a failure occurs a large number of reconstructions start. When the bandwidth available for reconstruction is low, the reconstructions are delayed which may lead to an increased failure rate. This trade-off has still to be investigated.

6.6 Conclusion

In this chapter, we show that placement policies strongly impact the performance of P2P storage systems. We study three different policies, a Global and two *local*, and show that under resource constraints, the Global policy behaves better in terms of probability of data loss and MTTDL than the *local* policies.

We suggest architectural choices to improve the performances of local policies. We show that, by using a new reconstruction strategy, namely *external reconstruction*, and by increasing the size of the neighborhood, local policies can have performances almost equivalent to the ones of the Global, while keeping their practical advantages.

CHAPTER 7

Hybrid Coding

It is known that erasure codes are a very efficient solution in space to obtain fault-tolerance. However, as previously addressed in Chapter 3, the reconstruction process of storage systems based on erasure coding has an important overhead in bandwidth consumption. When a fragment of redundancy is lost, the whole original data has to be retrieved to regenerate the lost fragment. At the opposite, in a system using full replication, a repair is done by simply sending again the lost data. However, adding more redundancy increases considerably the space overhead, and consequently the bandwidth.

In this chapter, we investigate in detail the use of a *Hybrid coding*: mixing erasure codes and simple data replication, i.e., keep one full replica of the data along with the erasure code fragments. The idea is to try to get the best of both worlds: the storage efficiency of erasure codes and the repair efficiency of replication. We try to answer the following question: *for a given erasure coding configuration, is there a Hybrid solution that performs better in terms of bandwidth and loss rate, and that uses the same or less storage space?* We present results for the Reed-Solomon (RS) coding as we argue that these codes are the most widely used in practice. Nonetheless this analysis can applied for different codes.

Our Contribution

We model the Hybrid system and the Reed-Solomon system using a continuous-time Markov chain. These chains are more refined than the one previously studied, as they take into consideration the existence of the reconstructor peer. From these Markov chains we derive closed-form expressions to approximate the bandwidth usage and the system durability. These expressions are validated by simulations. We analyze different scenarios and we characterize when it is interesting to use the Hybrid solution.

To summarize our results:

- when storage is the scarce resource, RS system have a higher durability;
- when bandwidth is the scarce resource, the Hybrid solution is the better choice;
- when the storage overhead is not sufficient to use Hybrid, we propose a simple *Mixed system* that is better than RS in terms of bandwidth.

To the best of our knowledge, this is the first study to compare the efficiency of the Hybrid system with lazy repair Reed-Solomon system. These two solutions are, from our point of view, the most practical methods to obtain an efficient utilization of the bandwidth when employing erasure codes on distributed storage.

Related Work

P2P and large scale distributed storage systems have been analyzed by using Markov chains: for erasure codes in [14, 38, 120] and for replication in [98, 28]. In this work, we model the Hybrid system with a Markovian model. We also introduce a new chain for RS systems that models the failure of the reconstructor during a repair.

Rodrigues and Liskov in [105] compare the Hybrid system versus replication in P2P Distributed Hash Tables (DHTs). However, there is no analysis of the impact of the Hybrid compared to the traditional erasure codes. Dimakis et al. in [41] study the efficiency of bandwidth consumption for different redundancy schemes, among them the Hybrid approach. They state that the Hybrid has a better availability/bandwidth trade-off than the traditional erasure codes. But they do not consider the lazy repair. Both of these works focus on availability and *they do not consider the durability of the data*. They also do not take into account the time to process the reconstructions. Camargo et al. in [40], evaluate by simulations the use of hybrid coding. They compare the bandwidth consumption of the hybrid solution to an ideal erasure code.

By using Markov chains, we exhibit the impact of this parameter on the average system metrics. Furthermore, they only consider RS using an *eager repair* policy, which is highly inefficient for the bandwidth. In [20], the authors propose the *lazy repair* mechanism to improve the reconstruction process. Here, we thus compare Hybrid and an RS system using lazy repair.

Other works introduced other efficient and promising codes, as for example *Regenerating codes* [41] and *Hierarchical codes* [46]. However, as far as we know, they are not used in practice yet. Hence, we focus on RS codes.

Organization

The remainder of this chapter is organized as follows: In the next section we give the details of the Hybrid solution. Then, in Section 7.2 we present the Markov chain models for both systems, along with a compilation of closed-form formulas and the validation by simulation. In Section sec:hybrid-results we show the results and discuss when it is interesting to use the Hybrid solution.

7.1 Description

In distributed storage systems using Reed-Solomon (RS) *erasure codes*, each block of data b is divided into s fragments. Then, r fragments of redundancy are added to b in such a way that any subset of s fragments from the $s + r$ fragments are enough to reconstruct the whole information of b. Observe that, the case $s = 1$ corresponds to the simple *replication*. In the *Hybrid system*, one of the $s + r$ peers stores a full copy of the orginal data, namely *full-replica* (see Figure 7.1).

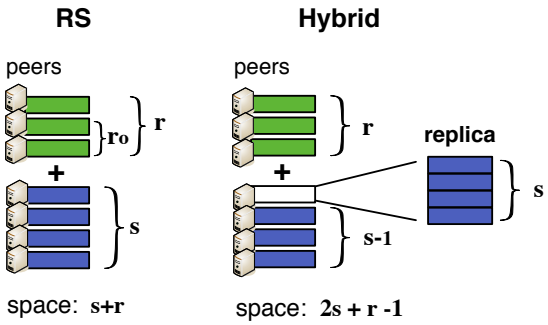


Figure 7.1: Description of RS and Hybrid systems. Both systems store the data fragments in $s + r$ peers. In the Hybrid system, one of the peers keeps a full-replica.

7.1.1 Reconstruction Process

To ensure fault tolerance, the storage systems must have a maintenance layer that keeps enough redundancy fragments available for each block b. We consider here archival systems, where there are only transient churn and permanent peer departures or failures [28]. We describe in the following the reconstruction process in both systems.

Traditional Reed-Solomon Approach (RS). For the traditional Reed-Solomon, we assume a lazy repair strategy [38] which is more efficient in terms of bandwidth. Given a threshold $0 \leq r_0 < r$, the reconstruction process starts only when the number of fragments of b is less than or equal to $s + r_0$. Observe that the case $r_0 = r - 1$ corresponds to the eager reconstruction.

When the reconstruction starts, a peer $p(b)$ is chosen to be the *reconstructor*. It retrieves s remaining fragments of b, rebuilds the missing redundancy, and resends the missing fragments into the system. Note that, after reconstructing the missing redundancy of b, the peer $p(b)$ possesses a full-replica of the block which is discarded afterwards. We refer to this system as the *Traditional Reed-Solomon (RS)*.

Hybrid Approach. In the Hybrid system, we call by $p_c(b)$ the peer that contains a full-replica of the block b. When there is a failure, two cases can happen.

If the peer $p_c(b)$ is still alive, it generates the lost fragments from its full-replica. It then sends the missing fragments to different peers in the network. To be able to do that, the peer only needs to store the initial block or, equivalently, s fragments. As a matter of fact, it can quickly create the other fragments at will. For usual parameters, the coding/decoding is very fast and the bottleneck is the peer upload bandwidth (at least 20 Mbps for $s + r \leq 128$ when using the library ZFEC [94], on a Intel Core-Duo 2.16 Ghz).

When the peer $p_c(b)$ fails, a new peer is chosen to maintain the full-replica. In this case, the whole block needs to be reconstructed. This is accomplished by using the traditional Reed-Solomon process, with the addition that the reconstructor keeps a full-replica of the block at the end of the process. From that we see that a Hybrid system can be easily built in practice from an RS system.

7.1.2 Code Efficiency

As explained in [41], a system in a steady state has to rebuild in average as much data as what is lost. The bandwidth usage induced by the reconstructions thus depends directly on the *space overhead* used to encode redundancy, defined as $k = (s + r)/s$. A second important factor is the efficiency of the reconstruction process. We define the *efficiency factor* ρ of a P2P storage system as the number of bits of data transferred to reconstruct one bit of data. We can then express the average bandwidth usage of a P2P storage system as:

$$\mathbb{E}[BW] = \alpha N \rho k d,$$

where α is the peer failure rate per unit time, N the total number of peers in the system, and d the amount of data per disk.

Efficiency of Replication and RS. In systems using replication, a replica is recovered by just sending a copy to a new peer. Consequently, $\rho_{repl} = 1$, the optimal efficiency.

In an eager RS system, s fragments of redundancy are exchanged to reconstruct one fragment of data: $\rho_{RS} = s$. This is highly inefficient. However, RS systems can be made a lot more efficient by using the lazy repair policy. In this policy, $s + i - 1$ fragments are sent to recover i fragments (s are retrieved by the peer that reconstructs the block, it keeps one fragment, and then sends the $i - 1$ missing fragments). Remember that, when using the lazy repair strategy we start the reconstruction at the redundancy level r_0. Hence, in most of the cases, the number of missing fragments is $i = r - r_0$ when the reconstruction is fast enough. Consequently,

$$\rho_{RS} \approx \frac{s + r - r_0 - 1}{r - r_0}.$$

Replication has a better efficiency factor than RS systems (even if, for large $r - r_0$, the difference is not as significant). However, if we consider the space overhead needed to tolerate n_f failures, RS has an important advantage. One needs to add n_f fragments of size only L_b/s each (with L_b the size of a block) to tolerate n_f failures (thus $k_{RS} = (s + n_f)/s$, in an eager RS system). In replication systems, n_f whole copies of the data are needed ($k_{repl} = n_f + 1$). In a lazy RS system, the trade-off is more complex, as redundant fragments can be added in the systems for two reasons: tolerate more failures or improving the bandwidth efficiency of the code. We will discuss that more thoroughly in the following of the chapter.

Efficiency in Hybrid system. The idea of this system is to try to have the repair efficiency ρ of the replication, while having a storage overhead to tolerate failures close to the one of RS system.

As stated, the reconstruction in the Hybrid consumes one fragment when $p_c(b)$ is alive or s fragments when the full-replica needs to be replaced. In both cases, one bit of bandwidth is used to recover one bit of data, thus $\rho_{Hybrid} = 1$, which is the optimal value. The Hybrid system is able to tolerate one more failure by adding one small fragment of redundancy (as in RS systems). However, it uses an additional copy of the data. Nevertheless, the value of k_{Hybrid} to tolerate n_f failures has to be determined: *the presence of the*

Table 7.1: Summary of the main notations.

s	Number of initial fragments
r	Number of redundancy fragments (RS system)
r_0	Reconstruction threshold
k	Storage space overhead: $(s+r)/s$
α	Peer failure rate
γ	Fragment reconstruction rate
θ	Average time to reconstruct one fragment: $\theta = 1/\gamma$
γ^-	Block reconstruction rate
θ^-	Average time to retrieve the whole block: $\theta^- = 1/\gamma^-$

full-replica allows the system to tolerate more than r failures. In fact, the Hybrid system can tolerate up to $s+r-1$ failures in some cases (when $p_c(b)$ is not affected). Furthermore, the full-replica also greatly improves the speed of the reconstruction as a simple transfer is needed. This affects greatly the system durability (see Figure 7.5).

To summarize, the use of the Hybrid system improves the reconstruction efficiency, but at the cost of additional storage space and/or of a smaller tolerance to failures.

We analyze and compare the *precise trade-off redundancy-durability of the Hybrid system and of lazy RS system* in the following.

7.2 Markov Chain Models

We model the behavior of a block in the Hybrid system and in the traditional RS system with lazy repair by Continuous Time Markov Chains (CTMCs). From the stability equations of these chains, we derive the bandwidth usage and the system durability. We also give explicit closed-form formulas that approximate the system behavior. They also give a good intuition of the influence of the parameters. To model the RS system, we use a slightly different CTMC from the classic ones found in the literature [14, 38]. We take into account the possible death of the peer in charge of the reconstruction during the reconstruction process.

7.2.1 Model of the Hybrid System

In Figure 7.3, it is presented the Markov chain that models the behavior of a block b in the Hybrid system. Recall that, in a Hybrid system, $s+r$ Reed-Solomon fragments and one replica are present inside the system. We make the choice here that the peer storing the full-replica also stores one of the RS fragments. Hence, there are $2s+r-1$ fragments per block.

In brief, the states of the chain are grouped into two columns. The level in a column represents the number of Reed-Solomon fragments present in the system. The column codes the presence of the replica: present for the left states and absent for the right ones.

States. The chain has $s+2r+1$ states: $\{S_i; 1 \leq i \leq s+r\}$, $\{S_i^-; s \leq i \leq s+r-1\}$ and

S_{DEAD}. A state at level i (S_i or S_i^-) represents the case where i peers store a fragment. The states S_i (left) correspond to the case where the full replica is present inside the system. In this case, a block can only die if the replica is lost. Hence, we have $s+r$ such states. The states S_i^- (right) correspond to the cases where the replica was lost. We need at least s RS fragments inside the system in order to reconstruct the block, otherwise the block is lost. Hence, we have r states. Finally, the state S_{DEAD} represents the other situations when the block can no longer be reconstructed.

Transitions. The transitions of the chain correspond to *peer failures* and *reconstructions*. As in most analytical works of the literature [98, 38, 14], the peer failures are considered independent. The peer lifetime follows an exponential distribution of parameter α. A peer is still alive after a time t with probability $\exp(-\alpha t)$ and, in average, the lifetime is $MTTF = 1/\alpha$. When a block has i fragments, it loses one fragment with rate αi. The replica is lost with rate α and one of the other RS fragments with rate $\alpha(i-1)$. Thus, we have the following transitions:

$$\begin{array}{ll}
S_i \xrightarrow{\alpha(i-1)} S_{i-1} & 2 \leq i \leq s+r \\
S_i \xrightarrow{\alpha} S_{i-1}^- & s+1 \leq i \leq s+r \\
S_i \xrightarrow{\alpha} S_{DEAD} & 1 \leq i \leq s \\
S_i^- \xrightarrow{\alpha i} S_{i-1}^- & s+1 \leq i \leq s+r-1 \\
S_s^- \xrightarrow{\alpha s} S_{DEAD} &
\end{array}$$

The reconstruction process of b starts when a fragment is lost. We model a Poisson reconstruction as it is classically done in the literature [14, 98]. We use here processes with two different rates for the two cases: presence or absence of the full-replica. When the replica is present, the peer storing it sends the missing fragments one by one into the system with rate γ. We note $\theta = 1/\gamma$ the average time to send one fragment. When the replica is lost, the system rebuilds it. A peer is chosen as the new storer of the full-replica. It retrieves s fragments. We note γ^- the reconstruction rate of the replica and $\theta^- = 1/\gamma^-$ the average time to reconstruct it. Obviously, we have $\theta^- > \theta$. Given these notations, the transitions that represent the reconstruction process are:

$$\begin{array}{ll}
S_i \xrightarrow{\gamma} S_{i+1} & 1 \leq i \leq s+r-1 \\
S_i^- \xrightarrow{\gamma^-} S_{i+1} & s \leq i \leq s+r-1
\end{array}$$

We consider here a system with a constant amount of data. Hence, new data is reintroduced into the system when we experience a loss. Note that, this assumption has a small influence on the results for practical storage systems in which it is expected to have a very small probability of losing data. The rate of reintroduction of the data is γ^T.

$$S_{DEAD} \xrightarrow{\gamma^T} S_{s+r}$$

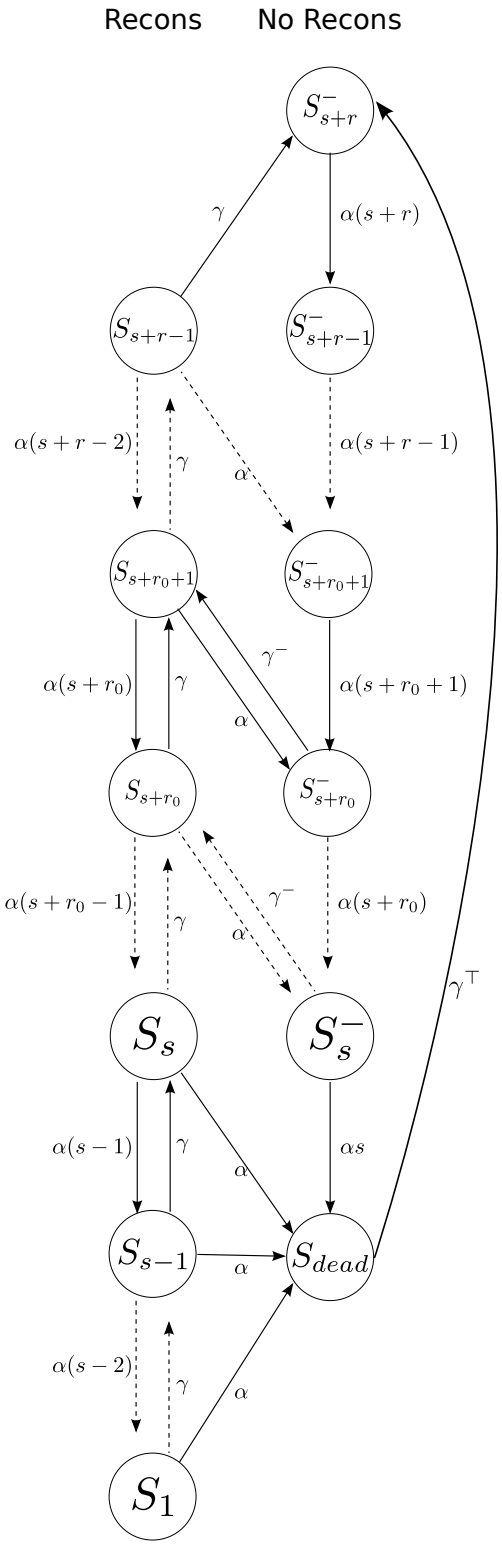


Figure 7.2: Markov chain of the system based on Reed-Solomon.

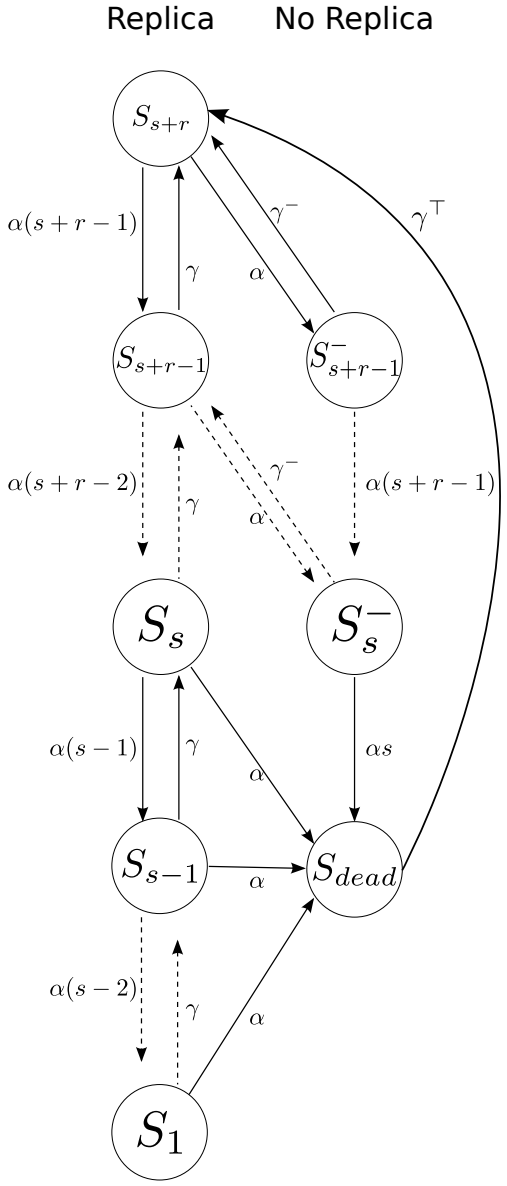


Figure 7.3: Markov chain of the Hybrid system.

7.2.2 Model of the Reed-Solomon System

We also model the behavior of a block b in a lazy RS system by a CTMC, depicted in Figure 7.2. We did not use the chains classically used in the literature [14, 38]. Our chain models the possible loss of the reconstructor $p(b)$ during a reconstruction. We draw our inspiration from the chain representing the Hybrid system. We code here the presence of the peer $p(b)$ in the system, in a way similar to how we code the presence of the full-replica in the Hybrid system.

States. The chain has $s + 2r + 1$ states: $\{S_i; 1 \leq i \leq s + r - 1\}$, $\{S_i^-; s \leq i \leq s + r\}$ and S_{DEAD}. Similarly to the Hybrid chain, the states of the chain are grouped into two columns. A state at level i (S_i or S_i^-) represents the case where i peers store a fragment. The states S_i correspond to the second phase of the reconstruction (the reconstructor has retrieved the whole block). In this phase, a block can only die if the peer $p(b)$ dies. Hence, we have $s + r - 1$ such states. The states S_i^- correspond to the three other cases: no reconstruction at the present time (sufficient number of present fragments), the first phase of the reconstruction ($p(b)$ is retrieving the s fragments), or the reconstructor died during the reconstruction. When $p(b)$ did not retrieve the block, we need at least s fragments inside the system. Hence, we have $r + 1$ states. Finally, the state S_{DEAD} represents the other situations when the block can no longer be reconstructed.

Transitions. The downwards transitions are similar to the ones in the Hybrid chain. When a block has i fragments, it loses one fragment with rate αi. When the block is under reconstruction, it loses the reconstructor (which stores one of the i fragments) with rate α. It gives the following transitions:

$$
\begin{array}{ll}
S_i \xrightarrow{\alpha(i-1)} S_{i-1} & 2 \leq i \leq s + r - 1 \\
S_i \xrightarrow{\alpha} S_{i-1}^- & s + 1 \leq i \leq s + r - 1 \\
S_i \xrightarrow{\alpha} S_{DEAD} & 1 \leq i \leq s \\
S_i^- \xrightarrow{\alpha i} S_{i-1}^- & s + 1 \leq i \leq s + r \\
S_s^- \xrightarrow{\alpha s} S_{DEAD} &
\end{array}
$$

The reconstruction process starts when the number of fragments in the system reaches the level $s + r_0$. We use here Poisson processes with two different rates for the two phases of the reconstruction. In the first phase, s fragments must be retrieved by the reconstructor, which then reconstructs the missing fragments. During the second phase, the reconstructor sends the missing fragments one by one into the system. We note γ^- the reconstruction rate of the first phase. Thus, the average time to retrieve the s fragments and to rebuild the missing fragments is $\theta^- = 1/\gamma^-$. We note γ the rate of the second phase, giving an average time $\theta = 1/\gamma$ to send one fragment into the system. Observe that $\theta^- > \theta$.

$$
\begin{array}{ll}
S_i \xrightarrow{\gamma} S_{i+1} & 1 \leq i \leq s + r - 2 \\
S_{s+r-1} \xrightarrow{\gamma} S_{s+r}^- & \\
S_i^- \xrightarrow{\gamma^-} S_{i+1} & s \leq i \leq s + r_0
\end{array}
$$

At last, to have a constant amount of data inside the system, we have the transition

$$S_{DEAD} \xrightarrow{\gamma^T} S^-_{s+r}$$

7.2.3 Bandwidth Usage and Loss rate

7.2.3.1 Steady-state

From the CTMC, we can derive the bandwidth usage and the loss rate of the systems. The principle is to compute the stationary distribution. The steady-state of the Markov chains presented above can be calculated exactly in polynomial time in $s + r$, by finding the eigenvector of eigenvalue 1 of the transition matrix. Another method is to solve the linear system of stability equations together with the normalization $\sum_i P_i + \sum_i P_i^- + P_{dead} = 1$.

We note P_i (resp. P_i^-) the probability to be at state S_i (resp. S_i^-) in the stationary distribution, for each $i = 1, \ldots, s + r$. We also denote by P_{dead} the probability to be at state S_{DEAD}. We have

$$T_{dead,dead} = \frac{H_{dead}}{P_{dead}},$$

where $T_{dead,dead}$ is the return time to S_{dead} and H_{dead} the holding time in state S_{dead}. By construction, $H_{dead} = \frac{1}{\gamma^T}$. Hence we derive the lifetime of a block:

$$lifetime_{block} = T_{dead,dead} = \frac{1}{\gamma^T P_{dead}}.$$

The loss rate of a system with B blocks then is

$$loss\ rate_{system} = \frac{B}{T_{dead,dead}} = B\gamma^T P_{dead}.$$

The expected bandwidth is given by the product of the rate at which a reconstruction transition is taken times the amount of data used by the reconstruction.
Hybrid system: Two cases. The reconstruction of a fragment transfers only one fragment. The reconstruction of the full-replica transfers s fragments.

$$\mathbb{E}[BW^{Hb}] \approx \left[s\gamma^- \sum_{s}^{s+r-1} P_i^- + \gamma \sum_{i=1}^{s+r-1} P_i \right] L_f B,$$

with L_f the size of a fragment.
RS system: Also two cases. The first phase of the reconstruction sends s fragments into the system. This phase is done when we have at most $s + r_0$ fragments inside the system. In the second phase of the reconstruction (reconstructor has retrieved s fragments), a

reconstruction uses only one fragment.

$$\mathbb{E}[BW^{RS}] \approx \left[s\gamma^- \sum_{s}^{s+r_0} P_i^- + \gamma \sum_{i=1}^{s+r-1} P_i \right] L_f B.$$

These formulas can be computed formally and numerically by using, for example, the MAPLE software. However, they do not give any idea of how the system behaves. Hence, we present approximated closed-form formulas in the next section.

7.2.3.2 Approximations

We present here approximated closed-form formulas that give a good intuition of the system behavior and of the influence of the different parameters. The approximations assume that $\alpha(s+r) \ll \gamma$ and that γ, γ^- and γ^T are of the same order. Observe that these assumptions are reasonable. In a practical system, if we want a very small loss rate, the time to reconstruct a block or a fragment has to be greatly smaller than the time to lose a peer. This is especially true for back-up systems with small churn where most peer losses correspond to disk failures. The following formulas are first order expressions when $\alpha/\gamma \to 0$. The main idea of the proof for the Hybrid system is to prove that a state at level i is of order $\theta(\frac{\alpha}{\gamma})^{s+r-i}$. For the RS system, we similarly find the order of each state. The proofs are omitted due to the lack of space. However, they are validated in Section 7.2.4 by comparing them to the exact numerical values given by the stability equations of the chains.

Approximations for Hybrid systems. We note P_{dead} (resp. P_{s+r}) the probability to be at state S_{DEAD} (resp. S_{s+r}) in the stationary distribution. Under these assumptions, we have

$$P_{dead}^{Hb} \approx \frac{\alpha}{\gamma^-} \frac{(s+r-1)!}{(s-1)!} \left[\left(\frac{\alpha}{\gamma}\right)^r + \sum_{i=1}^{r} \left(\frac{\alpha}{\gamma^-}\right)^i \left(\frac{\alpha}{\gamma}\right)^{r-i} \right] P_{s+r}^{Hb}. \tag{7.1}$$

If $\gamma = \gamma^-$, the formula simplifies to

$$\boxed{P_{dead;\gamma=\gamma^-}^{Hb} \approx \frac{2\alpha}{\gamma^T} \frac{(s+r-1)!}{(s-1)!} \left(\frac{\alpha}{\gamma}\right)^r P_{s+r}^{Hb}.} \tag{7.2}$$

In both cases, P_{s+r}^{Hb} is

$$P_{s+r}^{Hb} = 1 - \frac{\alpha(s+r-1)}{\gamma} - \frac{\alpha}{\gamma^-} + o(\frac{\alpha}{\gamma}).$$

The bandwidth usage in this system is well estimated by

$$\boxed{\mathbb{E}[BW^{Hb}] \approx \alpha(2s+r-1)L_f B.} \tag{7.3}$$

These formulas are validated in Section 7.2.4 and discussed in Section 7.3.

Approximations for lazy RS systems. For RS systems, we have

$$\boxed{P_{dead}^{RS} \approx \frac{\alpha s}{\gamma^{\top}} \frac{(s+r_0)!}{s!} \left(\frac{\alpha}{\gamma^{-}}\right)^{r_0+1} \frac{1}{H_{s+r} - H_{s+r_0}},} \tag{7.4}$$

where $H_n = \sum_{i=1}^{n} 1/i$ is the n-th harmonic number. We also estimate the bandwidth usage of the system in the steady-state:

$$\mathbb{E}[BW^{RS}] \approx \frac{\alpha(s+r-r_0-1)L_f B}{\frac{\alpha}{\gamma}(r-r_0-1) + \frac{\alpha}{\gamma^{-}} + H_{s+r} - H_{s+r_0}}. \tag{7.5}$$

In the case $\gamma = \gamma^{-}$, we obtain a simplified formula:

$$\boxed{\mathbb{E}[BW_{\gamma=\gamma^{-}}^{RS}] \approx \frac{BL_f(s+r-r_0-1)\alpha}{\frac{\alpha}{\gamma}(r-r_0) + H_{s+r} - H_{s+r_0}}.} \tag{7.6}$$

Note that for the RS system, these formulas are similar to the ones approximated from the simplified Markov chain in Section 3.2 (page 41).

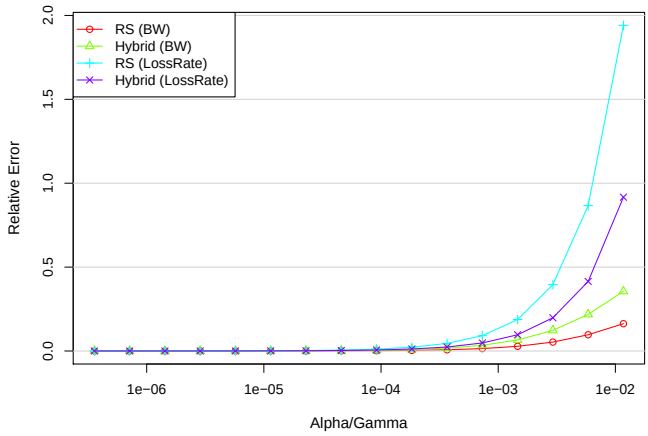


Figure 7.4: Accuracy of the approximations compared to the numerical solutions of the MC stability equations for different values of α/γ.

7.2.4 Validations

7.2.4.1 Validation of the Markov Chains

We validate the models by comparing them with the results of simulations. We implemented a custom cycle-based simulator to evaluate several characteristics of a real system. The simulator models the evolution of the blocks' states during time (number of available fragments and where they are stored) and the reconstructions being processed.

Table 7.2: Average and standard deviation of bandwidth usage (in Kpbs per peer) obtained using the simulations and the MC models.

(a) Total amount of data

D (in GB)	100	200	400	500
RS (MCM)	0.66	1.32	2.65	3.31
RS (Sim)	0.68 +- 0.32	1.35 +- 0.65	2.70 +- 1.31	3.39 +- 1.62
Hybrid (MCM)	0.50	0.99	1.99	2.49
Hybrid (Sim)	0.51 +- 0.22	1.02 +- 0.43	2.04 +- 0.87	2.56 +- 1.08

(b) Number of peers (N), with fixed amount of data

N	500	700	900	1000
RS (MCM)	1.32	1.32	1.32	1.32
RS (Sim)	1.36 +- 0.92	1.35 +- 0.77	1.34 +- 0.69	1.34 +- 0.64
Hybrid (MCM)	0.99	0.99	0.99	0.99
Hybrid (Sim)	1.02 +- 0.61	1.02 +- 0.52	1.02 +- 0.46	1.02 +- 0.43

Table 7.3: Data loss rate per hour obtained using the simulations and the MC models.

(a) Total amount of data

D (in GB)	100	200	400	500
RS (MCM)	$4.9 \cdot 10^{-3}$	$9.9 \cdot 10^{-3}$	$2.0 \cdot 10^{-2}$	$2.5 \cdot 10^{-2}$
RS (Sim)	$4.7 \cdot 10^{-3}$	$8.7 \cdot 10^{-3}$	$1.9 \cdot 10^{-2}$	$2.3 \cdot 10^{-2}$
Hybrid (MCM)	$8.8 \cdot 10^{-4}$	$1.7 \cdot 10^{-3}$	$3.5 \cdot 10^{-3}$	$4.4 \cdot 10^{-3}$
Hybrid (Sim)	$7.6 \cdot 10^{-4}$	$1.4 \cdot 10^{-3}$	$2.9 \cdot 10^{-3}$	$3.9 \cdot 10^{-3}$

(b) Number of peers (N), with fixed amount of data

N	500	700	900	1000
RS (MCM)	$1.7 \cdot 10^{-3}$	$1.7 \cdot 10^{-3}$	$1.7 \cdot 10^{-3}$	$1.8 \cdot 10^{-3}$
RS (Sim)	$1.5 \cdot 10^{-3}$	$1.6 \cdot 10^{-3}$	$1.6 \cdot 10^{-3}$	$1.7 \cdot 10^{-3}$
Hybrid (MCM)	$9.9 \cdot 10^{-3}$	$9.9 \cdot 10^{-3}$	$9.9 \cdot 10^{-3}$	$9.9 \cdot 10^{-3}$
RS (Sim)	$9.4 \cdot 10^{-3}$	$9.6 \cdot 10^{-3}$	$9.8 \cdot 10^{-3}$	$9.7 \cdot 10^{-3}$

When a disk failure occurs, the simulator updates the state of all blocks that have lost a fragment, and starts the reconstruction if necessary.

We ran extensive simulations for different sets of parameters for both systems. A small summary of the results is shown in Tables 7.2 and 7.3[1]. The results obtained from the stationary distribution of the CTMCs are compared with the results given by the simulator. We see that the MC models closely match the simulation results.

[1]In practice, storage systems should have a very low data loss rate, e.g. 10^{-20}. Such a loss rate is impossible to evaluate by simulation. To overcome this difficulty, we choose a non realistic set of parameters to experience losses for the experiments of Tables 7.2 and 7.3 In particular, the reconstruction threshold $r_0 = 2$ and the disk MTTF = 90 days are set very low.

7.2.4.2 Validation of the approximations

We validate the approximated formulas by comparing them with the results of the Markov chains. Recall that they are first order formulas when $\alpha/\gamma \to 0$. Figure 7.4 shows the accuracy of the closed-form approximations versus the results obtained by the Markov Chain models for different ratios of α/γ. The figure confirms that the difference decreases when the ratio decreases. For values of $\alpha/\gamma < 10^{-3}$ the results given by the approximations are very close to the Markov chain models. Note that it corresponds to real scenarios: if we consider a disk Mean Time To Failure (MTTF) of 1 year, and a reconstruction time of 1 hour, this ratio is $1.1 \cdot 10^{-4}$. For a MTTF of 2 years and a reconstruction of 10 minutes, the ratio is $9.5 \cdot 10^{-6}$. For such values of α/γ, several experiments with different sets of parameters confirmed the accuracy of the equations.

7.2.5 Reconstruction Rates γ and γ^-

To finish this section and before studying different system design scenarios, we discuss the impact of reconstruction rates on the system loss rate. The results presented hereafter are calculated numerically from the stability equations of the CTMCs.

If the Hybrid system has a large advantage in terms of bandwidth consumption, it has also a good behavior in terms of loss rate. As a matter of fact, when the full-replica is present, the reconstruction time θ is smaller than the one of the RS system. Only one fragment has to be sent into the system instead of doing a full RS reconstruction. Figure 7.5 (left) shows the loss rate for different ratios between θ and θ^- (in fact, we fixed $\theta^- = 32$ and varied θ from 1 to 32). Note first that, in this case, the loss rate of the RS is not much impacted by the θ, which confirms that it only depends on θ^- (Equation 7.4). Conversely, the Hybrid system is sensible to ratio θ/θ^-. The smaller the ratio, the better the loss rate because of the increased efficiency of the repair using the full-replica. In practice, it is not easy to guess the ratio a priori (we have to compare the upload of one fragment by the peer storing the full-replica and the upload of one fragment by s different peers. The ratio strongly depends on the network behavior and on the peer characteristics). Hence, in the following, we present plots with $\theta = \theta^-$. We are being conservative with the Hybrid solution. The goal is to ensure that when the Hybrid solution is better in an experiment, it is true for any values of θ^-.

As stated by Equations 7.1 and 7.4, the loss rates of both systems are exponentially impacted by the reconstruction rates γ and γ^-. Figure 7.5 (right) depicts this behavior for increasing values of $\theta = 1/\gamma$ (with $\theta = \theta^-$, $s = 16$, $r = 24$, $r_0 = 7$). Obviously, longer reconstruction times mean less durability. The bandwidth consumption remains almost constant when varying γ and γ^- (when $\alpha/\gamma \ll 1$ as in the studied systems), so these figures are omitted.

7.3 Results

In this section, we compare the Hybrid systems with the lazy Reed-Solomon systems. We use three metrics: bandwidth usage, loss rate, and storage overhead. The experiments

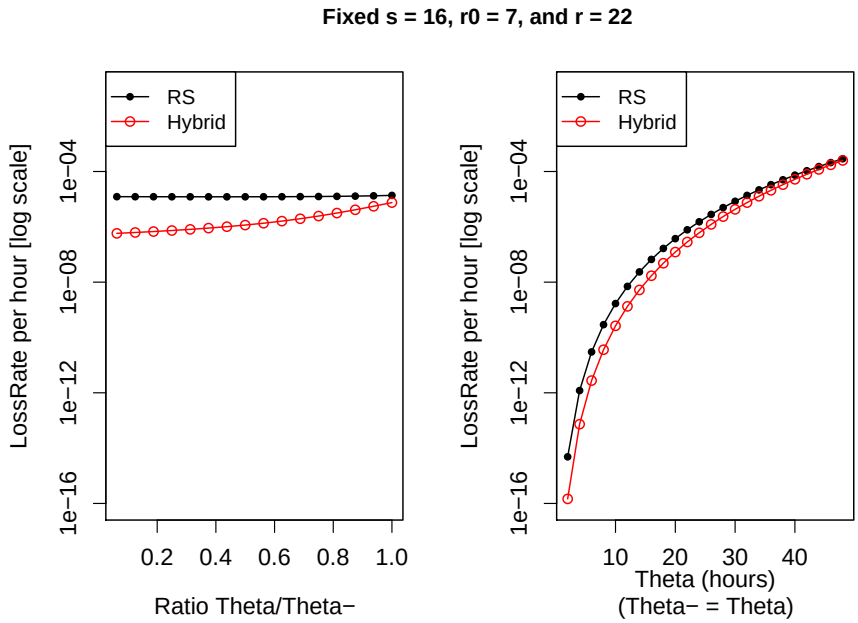


Figure 7.5: Influence of the reconstruction times on the system loss rate. Left: study of the ratio θ/θ^{-}. Right: the loss rate varies exponentially with the reconstruction time.

try to answer the following question: *for a given RS configuration, is there a Hybrid system that performs better in terms of bandwidth and loss rate, and that uses the same or less storage space?*

Three main scenarios are studied. In the first, we have a fixed storage space. In the second, we compare both approaches when they consume the same amount of bandwidth. In the third scenario, they have the same data loss rate.

In the light of these studies, we discuss which system should be used in function of the parameters. In brief, when the bandwidth is not the issue, it is better to use an RS system as it has a much better reliability. However, when bandwidth is the sparse resource, it is better to choose the Hybrid system (if possible), as it achieve a much better durability for the same bandwidth usage. When the storage space is too limited to allow system designer to use the Hybrid approach, we propose a *Mixed approach* which improves the system behavior when compared to RS at the cost of an increased implementation complexity. The results presented hereafter are calculated numerically from the stability equations of the CTMCs.

Value of the parameters for the studies. In the following experiments, we use a set of default parameters for the sake of consistency (except when explicitly stated). We study a system with $N = 10000$ peers. Each of them contributes with $d = 64$ GB of data (total of 640 TB). We choose a system block size of $L_b = 4$ MB, $s = 16$, giving $L_f = 256$ KB. The system wide number of blocks is then $B = 1.6{\cdot}10^8$. The $MTTF$ of peers is set to one year. The disk failure rate follows as $\alpha = 1/MTTF$. The block average reconstruction

time is $\theta = \theta^- = 12$ hours. Observe that we are penalizing the Hybrid solution when choosing $\theta = \theta^-$ (see discussion of Section 7.2.5). The goal is to ensure that, when the Hybrid solution is better in an experiment, it is true for any values of θ^-.

A first remark: Feasibility of a Hybrid Solution. To use a Hybrid system, the amount of space available must be at least two times the original data, as a full-replica is stored. We say that the storage overhead has to be greater than 2: $k \geq 2$ or, equivalently, $r \geq s$. *For the remainder of this chapter, we aim at comparing both systems when using the same storage overhead k. Hence, the redundancy factor of the Hybrid is set to* $r' = r - s + 1$. That is, the space consumed by the full-replica in the Hybrid system is taken out from the redundancy of the RS.

7.3.1 Fixed Space Overhead Scenario

The first scenario of study has a fixed storage overhead $k = 2.5$ for both systems. Then, for the chosen value $s = 16$, the redundancy r in RS is $r = 24$. Consequently, for the Hybrid system the redundancy is only $r' = 24 - 16 + 1 = 9$.

Threshold value of RS. When the above parameters are fixed, the choice of the threshold value r_0 gives the trade-off between the bandwidth and data loss rate for the RS system (see Figure 7.6). As we have seen in Section 7.1.2, when r_0 is close to r, the *RS* system consumes a lot of bandwidth. The extreme case is the eager reconstruction, $r_0 = r - 1$: in our scenario, it consumes 645 kbps per peer (to maintain 64 GB per peer). However, the benefits in loss rate are enormous, the *RS* achieves a loss rate of 10^{-29} block per hour (data lifetime of 10^{24} years). When decreasing the threshold r_0, the improvement of bandwidth is very pronounced when r_0 is close to r. Then the gains become less important when it approaches to 0 (this behavior can be explained by the Harmonic sum factors in Equations 7.5 and 7.6 giving the bandwidth consumption of RS systems). Conversely, the loss rate increases exponentially when r_0 decreases (Equation 7.4).

Comparison with Hybrid. Note that the Hybrid system has no threshold value to tune: its bandwidth and loss rate are constant in this experiment. We remark that, for the eager case ($r_0 = r - 1$), the RS system consumes s times more bandwidth than the Hybrid system. This can be explained from Equations 7.3 and 7.6: $BW_{RS}/BW_{Hybrid} \approx s$, but, at the same time, it has a loss rate smaller by a factor of at least $\alpha^{(s-1)}$.

We confirm that, for a given storage overhead, the RS system is a better option than the Hybrid to achieve low values of loss rate, but this comes at the cost of a very high bandwidth demand.

However, when using the Hybrid solution, one can obtain a reasonable value of loss rate while using much less bandwidth for maintenance. For example, Figure 7.6 shows that, for an RS with threshold $r_0 \leq 7$, the durability of the Hybrid is better. Moreover, the Hybrid consumes much less bandwidth. For instance, when $r_0 = 6$, the durability of Hybrid is two orders of magnitude smaller than RS. It comes with a gain in bandwidth of 34% (the Hybrid consumes 42.5 kbps instead of 64.5 kbps of the RS). In the following, we see that these gains are even much larger when there is more available space.

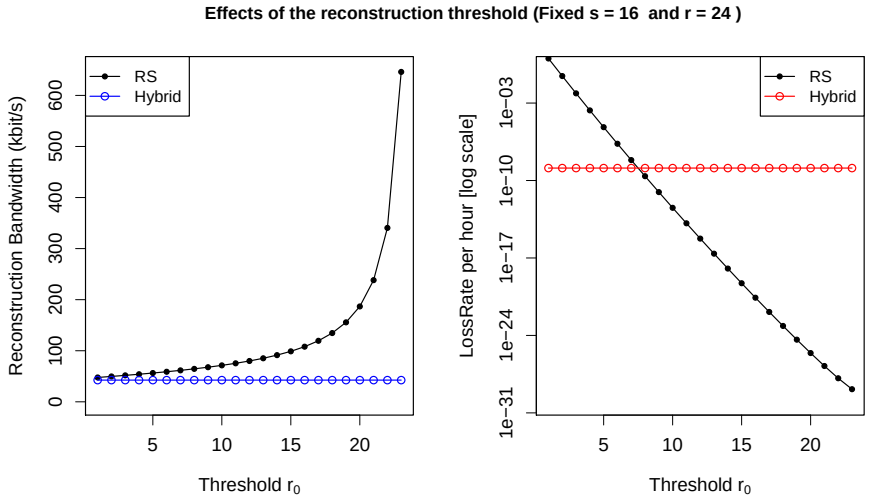


Figure 7.6: Comparison of the bandwidth usage and loss rate of RS ($s = 16$, and $r = 24$) and Hybrid ($r' = 9$) systems and for different values of reconstruction threshold r_0. Both systems consume the same storage space.

7.3.2 Same Bandwidth Usage

In this scenario, we compare systems with the same bandwidth usage and increasing levels of redundancy. That is, we study the question: *for a given bandwidth usage, which system is more durable?* We show that the Hybrid is a better option in this case.

To compare the systems in this scenario, we need to choose the value of the threshold r_0 for which we have approximately the same bandwidth usage. This can be done by equaling the approximations for bandwidth usage in both systems (Equations 7.3 and 7.6). We can then numerically isolate the term r_0 in function of s and r and choose the largest value of r_0 that gives a bandwidth consumption close to the Hybrid solution.

Figure 7.7 shows the experiment with increasing values of the redundancy r, from 16 to 48 (i.e., space overhead between 2 and 4). We see that, while using the same or less bandwidth, the Hybrid achieves a loss rate of many orders of magnitude smalller than the RS.

The explanation is that, starting from the eager RS system which has a very small loss rate (see Figure 7.6), we lower the threshold value r_0 to reduce the bandwidth consumption. We have to lower a lot this value to obtain the same bandwidth than Hybrid and lowering r_0 increases the loss rate exponentially, see Equation 7.4.

Remark that, for the range $16 \leq r \leq 26$, we have straight lines for RS in the plots. This is due to the fact that, for these values of r, we have to lower the threshold to $r_0 = 0$. In other words, there is no value of r_0 that matches the bandwidth of the Hybrid in these cases.

We confirm that, for the same amount of bandwidth usage, the Hybrid gives a much more durable solution (for the same space overhead). We conclude that the Hybrid uses the bandwidth much more efficiently than the RS, since its reconstruction process does not waste

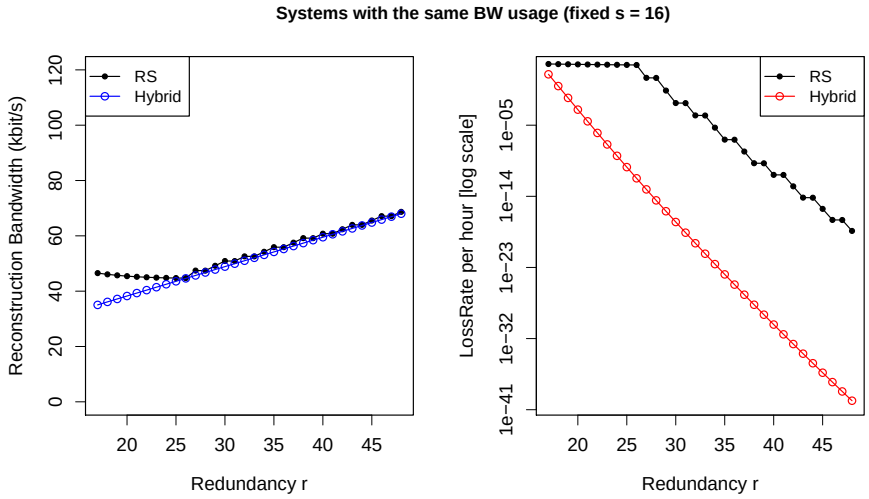


Figure 7.7: Comparing systems ($s = 16$) with the same Bandwidth usage and increasing redundancy (i.e., storage space). For the RS system, choose the r_0 that gives the closest bandwidth usage of the Hybrid. Both systems consume the same storage space.

any transferred information (see discussion of Section 7.1.2).

7.3.3 Same Durability

In this scenario, we compare systems with the same loss rate for increasing values of redundancy. Figure 7.8 shows an experiment with r varying from 16 to 48 (we remind that $r' = r - s + 1$). The threshold value of RS is chosen to give the same loss rate than the Hybrid. That is, pick the smaller r_0 that gives a durability close to the Hybrid. Actually, this value can be calculated using the approximated formulas, by equaling the equations giving P_{dead} (Equations 7.1 and 7.4). We roughly obtain that the sought-after value of r_0 is $r_0 \approx r - s - 1$. Note that it gives $r_0 = r' - 2$.

As an example, when $r = 32$ ($k = 3$), both systems get a loss rate of 10^{-20}, but the Hybrid consumes 32% less bandwidth.

We confirm that, to achieve the same level of durability, the Hybrid system consumes less bandwidth than the RS (for the same space overhead).

In this study, we first consider the Hybrid system corresponding to a fixed storage overhead. We then found an RS system with the same durability. Conversely, for a given storage overhead, if we choose an r_0 for the RS system, there exist a Hybrid system with the same (or better) durability if $r \geq s + r_0 + 1$ (coming from $r_0 = r' - 2$), i.e., the storage overhead is more than $\frac{2s+r_0+1}{s}$. Otherwise, there exists no Hybrid system for the targeted durability.

For system designers, if the most scarce resource is the bandwidth, then it is interesting to pay a little bit in space to be able to use an Hybrid solution. This way, they have a system that consumes less bandwidth and is more reliable.

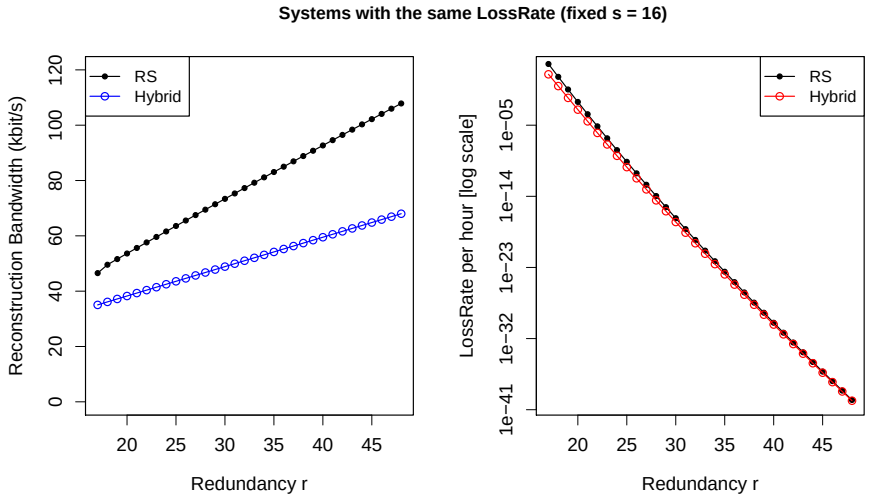


Figure 7.8: Comparing systems with the same Loss Rate and increasing redundancy (i.e., storage space). For the RS, choose the smaller r_0 that gives the same or better loss rate than Hybrid (i.e., $r_0 \approx r - s + 1$). Both systems consume the same storage space.

7.3.4 Mixing Hybrid and RS

In what precedes, we show that, if we target a given durability, the Hybrid is an interesting and a feasible solution when the amount of storage space available is more than twice the original data (storage overhead $k \geq \frac{2s+r_0+1}{s}$). In this section, we study the following problem: *is it possible to perform better than the RS system when the storage overhead is less than this value?*

For system designers that are ready to cope with some additional implementation complexity, we present a new system, namely *Mixed system*. The idea is simply to store a full-replica of the data for some of the blocks of the system, when the others use a simple RS redundancy. We show that this Mixed strategy allows to improve the bandwidth use of systems with a small storage overhead.

Building a Mixed system. When the storage overhead k is fixed, we can use an RS system with $r = (k-1)s$. The threshold r_0 is then chosen according to the targeted system durability. Note that in this system, a space of $r - r_0$ redundant fragments is used to lower the bandwidth usage (Equation 7.5). It is not necessary to obtain the desired loss rate (Equation 7.4). We try in the following to use this extra space to build full-replica for some blocks.

In a Mixed system with a storage overhead of r, some blocks are coded with RS with a redundancy of $R \leq r$ and the others with Hybrid with $r' = r_0 + 1$. The two kinds of blocks have approximately the same durability with this value of r'. The proportion of blocks that can employ a Hybrid system can be calculated by $F = (r - R)/(s + r_0 + 1 - R)$. An RS system with redundancy R uses $s + R$ fragments. Hence, $r - R$ is the space left when we have coded a block with a redundancy of R (instead of r). A Hybrid system

uses $2s + r_0 + 1$ fragments. Hence, we can build a Hybrid block with $s + r_0 + 1 - R$ additional fragments.

Figure 7.9 (left) shows the fraction F of Hybrid blocks versus RS when increasing the redundancy R (from 9 to 20), for a fixed $s = 16$, $r_0 = 8$, and $r = 20$ (remark $r - r_0 < s$).

We now have to find the best R in terms of bandwidth usage. Two contradictory effects appear: when decreasing R, more blocks can be coded in the more efficient Hybrid. However, at the same time, the RS consumes more bandwidth. The extreme case is the inefficient eager RS.

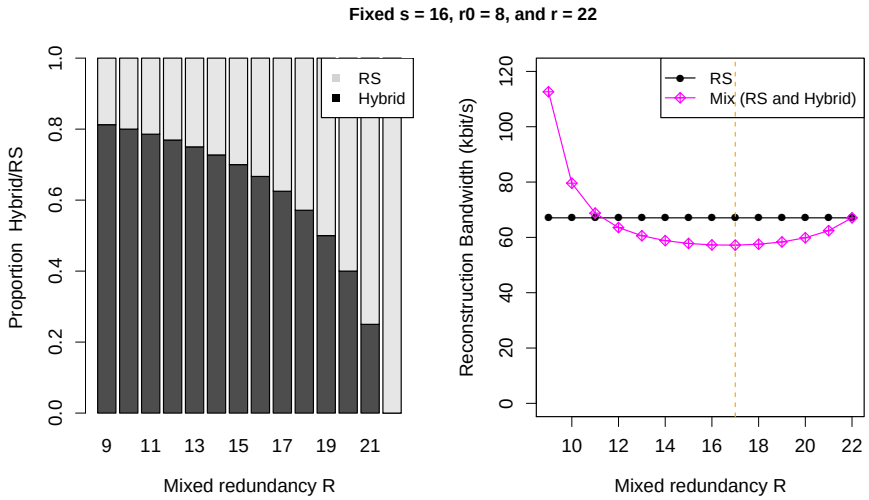


Figure 7.9: Evaluation of a Mixed system with maximum redundancy $r = 22$ for different values of RS redundancy $R \leq r$. Left plot, the proportion of blocks using the Hybrid for different values of R. Right plot, the corresponding bandwidth consumption. The dashed vertical line represents the best Mixed system.

To discover the best proportion of RS and Hybrid for a given r, we experiment all values of R from $r_0 + 1$ to r. Figure 7.9 (left) shows the bandwidth of a Mixed system. The straight line is the traditional RS system with $r = 20$. It is shown as a reference. When $R = 17$, the bandwidth consumption get the optimal value (in this case, the fraction of blocks that have a full-replica is 62.5%). It represents an improvement of 15% in bandwidth. When $R = 9$, the RS is in eager reconstruction. The remaining $r - R = 13$ fragments can be used to keep a proportion of 81.25% of blocks with a Hybrid system, but for this value of R we note that the Mixed system performs poorly compared the RS. In the opposite case, when $R = 20$, there is no extra space to build Hybrid blocks and all blocks employ the RS.

Benefits of the Mixed System for different storage overheads. We now consider different storage overheads: redundancies r from $r_0 + 1$ to $r_0 + s$. Note that the limit redundancy $r_0 + s$ offers a space sufficient to build a complete Hybrid system. For each value of r in the previous interval, we build the best Mixed system in terms of bandwidth by selecting the best value of R.

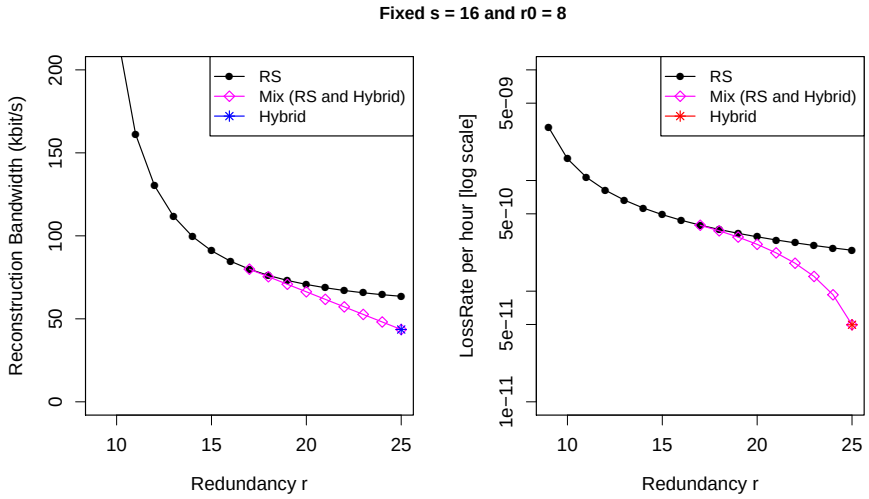


Figure 7.10: Evaluation of the Mixed system for different space overheads (values of redundancy r). The Mixed is an interesting solution when the space overhead does not allow a full Hybrid solution $k < \frac{2s+r_0+1}{s}$, but when k is not too low (more precisely, when $16 \leq r \leq 24$).

The benefits of the Mixed system can be seen in Figure 7.10. This experiment shows the bandwidth and loss rate for different values of the redundancy r from 9 to 25. We see that for very low values of redundancy, there is no advantage of using the Mixed solution. However, for $16 \leq r \leq 25$, the Mixed system is a better option (lower bandwidth and lower loss rate) than the traditional RS. When $r = 25$, there is enough space to employ the Hybrid system, which is the best solution in bandwidth and loss rate.

7.4 Conclusion

In this study, we compare the trade-offs between durability, bandwidth usage and storage overhead of the Hybrid system against a traditional Reed-Solomon erasure code. In practice, since the bandwidth is costly, we confirm that the Hybrid is an interesting solution to obtain a durable system with a minimal bandwidth usage. This advantage comes at the cost of additional storage overhead. On the other hand, when the space is the scarce resource, we show that the traditional RS system is more efficient in space to achieve a desired durability. However, this comes at the cost of high bandwidth usage.

When the storage overhead is not sufficient to use the Hybrid, we propose a simple Mixed system that is better than RS in terms of bandwidth and loss rate. This shows that different strategies to mix RS and replication can be imagined. The conception of an optimal strategy has still to be investigated.

CHAPTER 8

Concluding Remarks and Future Research

The contributions of this thesis belong to two domains of science. On one side we have the analysis of a dynamic system that is complex by nature: a large-scale peer-to-peer data storage system with billions of entities interacting with each other. This magnitude incites the research on novels techniques that harness its dynamics. On the other side, we bring practical results that can be directly applied to the development and implementation of peer-to-peer data storage systems. Our results show how to measure and improve the system reliability while keeping a reduced consumption of resources. A brief summary of these contributions are presented in the next section. Then in Section 8.2 we outline avenues for further research.

8.1 Results and Methodology

Instead of studying the detailed peculiarities of a specific storage system, we tried to move our analysis towards the modeling of its general behavior. This way our contributions are more general as well. Altough, throughout this dissertation, we kept our focus on the same systems metrics: bandwidth consumption, probability of losing data, and space usage. Then, we walked through many different design choices that impact their performances.

Using a Markov Chain model we characterized the average behavior of an erasure coded storage system that employs a lazy repair. Then we derived closed-form mathematical expressions that give the intuition of the system metrics. These expressions bring into light the relation between the main system parameters, and take into account the peers' failure rate and data repair rate. This model and its formulas were validated by simulations (Chapter 3).

The simulations gave us another interesting insight about the dynamic evolution of the system. The variations around the mean behavior are very important for provisioning its resources. Hence, we developed a fluid model that captures that dynamic and that allows us to give all the variations around the mean behavior (Chapter 4). Continuing on the same way, we went through a more detailed study of the repairing processes of the missing redundancy. We developed a queueing model that gives a refined distribution of the reconstruction time for a given limit of peers bandwidth (Chapter 5). This model takes into account the important impact of the peer imbalance on the repair time, and henceforth of the increased probability of data loss.

Additionally to these general studies, we show by simulations the impacts of different data placement strategies on the system metrics. We highlight the needs of a large neighborhood size to cope with the bursty characteristics of bandwidth consumption (Chapter 6). Furthermore, we study in detail the use of Hybrid coding solution that mixes the use of erasure coding with replication. We claim that, in practice, this is the solution of choice when implementing such storage systems (Chapter 7).

In brief, all these studies show that a distributed large-scale data storage solution can be optimized to consume less resources while keeping its reliability. At this moment, there are already some commercial solutions and *open-source* efforts to deploy such a system, which gives us an extra motivation to continue our efforts on building efficient solutions to improve its performance.

Analyzing a dynamic system that evolves over time is a challenging task. From our point of view, the techniques and methodologies that were used to obtain our results are themselves contributions. Indeed, we used several tools to address the "same" problem from different angles. That is, to obtain the results mentioned above, we went through a cyclic process of trying different methodologies to estimate the system metrics and to understand their dynamic behavior. Our methodology was usually to start by creating a simulation model, then deriving the mathematical models to characterize its behavior. During this processes of validation, the analytical models were *solved* by numerical computations and verified using mathematical software packages. Also, we point out the importance of the experimentation using a grid testbed to check if our models were close to a real system.

8.2 Future Research

It is clear that digital data storage is a main concern of the modern society. The use of large-scale data storage solutions is not yet in the mainstream. But we do believe that they will turn into a popular solution in the next years. Indeed, the emergence of cloud computing infrastructures is already pushing forward the demand of online data storage. Further on, the magnitude of these global services turns them into a large-scale storage system by itself, i.e., with the same issues and challenges of a peer-to-peer system but in a controlled environment.

In a short term, there are many issues to be studied that could be done by using the techniques that we used throughout this dissertation:

- Transient-phase. We focused on the steady-state properties of the system, which already gave us a good understanding of its behavior. However, not less important than that is the dynamic behavior of the transient phase of the system, i.e., when the amount of peers and the amount of data keep increasing.

- Time-dependent failure models. Another point is to study different models of peer availabilities and failures that depends on the past history of the peer. For example, failure models in which old disks have more probability to fail.

- Churn during reconstructions. As briefly mentioned, there are many churn models to characterize the dynamics of the network. However, to the best of our knowledge no

study of peer-to-peer storage systems do consider a complex churn model while modeling the reconstruction process. That is, during the reconstruction process the peers can leave and come back. Hence, one should consider the minimum time window that all the related peers of a reconstruction were available to transfer the fragments.

- Scheduling strategies. We pointed out by simulations that the scheduling strategies when processing the reconstruction of data blocks greatly impacts the reconstruction time. Hence, by simulations we show that the use of the *most-damaged* scheduling can reduce the data loss rate. Although, the development of analytical models for this scheduling strategy has yet to be done.

In a long term research, we aim at studying different techniques, as described in the following:

- Simulating large systems. When we want to validate a new model or a new system architecture, we often need to simulate the system. As the goal is to see if such systems can scale, we need to simulate large systems, with sometimes millions of peers with large disks, accounting for billions of chunks of data. It becomes impossible to accomplish on a single machine, leading to questions about efficient distributed simulations.

- Simulating rare events. We want to simulate practical systems with a very small probability of losing data, e.g., of the order of 10^{-20} during one year. It would take a prohibitive simulation time to see occurrence of loss events. A common procedure is to tune the parameters to induce the appearence of such rare events. Although, this can bias the results. Hence, we should consider the use of different techniques to study rare events. For example, with the use of mathematical frameworks as the importance sampling of Monte Carlo simulations.

- System size. For most of the peer-to-peer applications there is a common sense that having larger networks is much better than smaller networks. For example, in file-sharing or content distribution applications the system bahaves better when the network is large (it makes easier to find peers that have the requested content). However, in peer-to-peer storage sytems the gains of having a very large network maybe do not pay the costs of maintain it. Thus, one can imagine to split the big system into small independent sub-systems. This incites the research on the optimum size of a sub-system that keeps the desired performance metrics of the whole system. Further on, the analysis of a smaller system is more tractable.

- Combinatorial problems. We are also interested in more theoretical problems. For instance, the efficiency of the bandwidth utilization by the repairing process of erasure codes can be improved. We aim at developing combinatorial algorithms to calculate the best matching of reconstructor peers and of sending peers. That is, how to choose the best combination of peer connections to improve to efficiency of the network utilization. Another example is the optimal placement strategy of the redundancy fragments on disks, aiming at minimize the variations of bandwidth consumption when a disk fails.

— that's all folks —

APPENDIX A

The Use of Evolving Graphs for Performance Evaluation of Routing Protocols for Dynamic Networks

The assessment of routing protocols for mobile wireless networks is a difficult task, because of the networks' dynamic behavior and the absence of benchmarks. However, some of these networks, such as intermittent wireless sensors networks, periodic or cyclic networks, and some delay tolerant networks (DTNs), have more predictable dynamics, as the temporal variations in the network topology can be considered as deterministic, which may make them easier to study. Recently, a graph theoretic model – the *evolving graphs* – was proposed to help capture the dynamic behavior of such networks, in view of the construction of least cost routing and other algorithms. The algorithms and insights obtained through this model are theoretically very efficient and intriguing. However, there is no study about the use of such theoretical results into practical situations. Therefore, the objective of our work is to analyze the applicability of the evolving graph theory in the construction of efficient routing protocols in realistic scenarios. In this study, we use the *NS2* network simulator to first implement an evolving graph based routing protocol, and then to use it as a benchmark when comparing the four major ad-hoc routing protocols (AODV, DSR, OLSR and DSDV). Interestingly, our experiments show that evolving graphs have the potential to be an effective and powerful tool in the development and analysis of algorithms for dynamic networks, with predictable dynamics at least. In order to make this model widely applicable, however, some practical issues still have to be addressed and incorporated into the model, like adaptive algorithms. We also discuss such issues in this study, as a result of our experience.

The results presented here were published in [5] (best student paper award), and [7, 8].

A.1 Introduction and Motivation

Wireless communication networks have become increasingly popular in the computing industry and are widely available in our every day life. A MANET (Mobile Ad hoc NETwork) is a collection of mobile devices that are dynamically connected in an arbitrary manner, without the aid of any established infrastructure or centralized administration [143, 166]. These mobile devices with wireless transmitters are called *nodes*. When two nodes want to communicate, they may not be within each other's range, but they may

communicate if other nodes between them also participate in the network, acting as routers, forwarding packets to the other end. These are called multi-hop wireless ad hoc networks.

In several environments these nodes are free to move and they may have nonuniform characteristics, driving the emergence of complex ad hoc networks that may have a highly dynamic behavior. Thus, a large number of routing protocols have been developed for MANETs [165, 134]. Besides the mobility, such protocols must deal with the typical limitations of these networks, like energy limitations, low processing capacity, low bandwidth, and high error rates [143].

There are different approaches which try to optimize the cost of a routing path, but, until recently, most of them did not take into account the fact that some MANETs may have known connectivity patterns, as in DTNs [170, 150] such as LEO satellite networks [147, 141] and Wireless Sensor Networks (WSNs), where, due to energy limitations, network nodes can be scheduled to sleep in given periods [167, 163, 134, 152].

In this kind of networks, the topology dynamics at different time intervals can be predicted (see Fig. A.1). Therefore, the performance evaluation of routing protocols should be easier, although a formal tool for benchmarking such protocols has yet to become a standard.

A.1.1 Our contribution

In this study we use *evolving graphs* (EG) – a formal abstraction for dynamic networks [146, 139] –, in order to design and evaluate least cost routing protocols for MANETs with known connectivity patterns. These protocols are then used as benchmarks for establishing fair comparisons between the four MANET routing protocols, namely DSDV [156], DSR [151], AODV [157] and OLSR [149]. This is done through extensive simulations using *NS*2 within different scenarios. It is important to note that the previous protocols were designed to work within connected networks, as they are based in store-and-forward techniques, and not on store-carry-forward as the EG techniques. However, the EG routing protocols can be used as a lower bound reference.

We note that the algorithms and insights previously obtained through the EG model are theoretically very efficient and intriguing. The central objective of our work is thus to assess the usages of these theoretical results in practical situations, where packet dropping, for instance, may pose unexpected challenges to the EG algorithms. As an example, although we do not limit the buffer size on the nodes, we do propose an extensive discussion on the nodes' transmission queues.

The remainder of this work is organized as follows: After presenting the related works, in the next section we describe the concept of *evolving graphs*. The routing protocols to be compared are defined in Section A.4. Section A.5 shows the simulation environment and the decisions made in the implementation. Section A.6 and A.7 present the simulation results and analyses. We close the study with our conclusions and avenues for future research in Section A.8.

A.2 Related Work

It is interesting to note that the theory of Evolving Graphs is contemporary with the formulation of DTNs, although they followed different objectives. The EG theory formalised in a graph theoretical framework the concept of networks with known connectivity patterns. It focussed both on graph theory structural properties and on routing issues related to the optimisation of packet delivery between two nodes, using one of the following criteria: foremost, shortest, or fastest journey [135, 139, 146].

On the other hand, research done in the field of DTNs usually assume only partial, or even no knowledge about future connectivity patterns [144, 170]. Noticeably, however, some research done in DTNs do assume known connectivity patterns, as in [150], where the authors propose a Linear Programming solution when full future connectivity knowledge is available. Some other algorithms which require less knowledge were also presented. For the sake of conciseness, we will refer henceforth to networks with full future connectivity knowledge as deterministic DTNs, or DDTNs for short.

Another difference is on the model, where bounded buffers were considered on the nodes. In our work, we assume unbounded buffers, although we use bounded buffers for message transmission. In [161] the authors also propose a deterministic solution based on a tree construction, which considers in a simplified way the message transmission time.

In [164] the authors studied in detail the single-copy case, that is, for a message transmission there is only one copy of the message on the network, which is the same assumption used in the EG model, as described our study. They present an "Oracle-based" optimal algorithm and several other partial knowledge algorithms. In their paper there were no details on the possible congestion of the "Oracle-based" algorithms, which is one of the main contributions of our work presented here.

Another work which considers predictable behavior is [154] that presents a shortest delay path routing with a complete knowledge of future connections. It also compares this algorithm with Hot potato, Most Frequent Neighbor and Epidemic Routings. However, there is only a very small experiment on the delays with bursty traffic.

Finally, a close approach aiming at benchmarking routing protocols in dynamic networks is the MERIT framework [145]. It uses the notion of competitive analysis [142] on a dynamic setting in order to assess the quality of protocols studied on snapshots describing the history of the network. The results proven include finding a sequence of paths that connect a given pair of nodes throughout the system, such that the global routing plus re-routing costs are minimized.

A.3 Evolving Graph Model

The *evolving graph* (EG) model, proposed in [146] aims to represent a formal abstraction of dynamic networks, through the formalisation of a time domain in graphs.

As an example, consider the four snapshots taken at different time intervals of a MANET, as depicted in Fig. A.1. As one can readily observe, nodes D and G are never

connected on a single time interval. Notwithstanding, D can indeed send messages to G, using the *path over time* composed of D, C, E, F, G. Surprisingly, this otherwise trivial fact cannot be directly modeled by usual graphs.

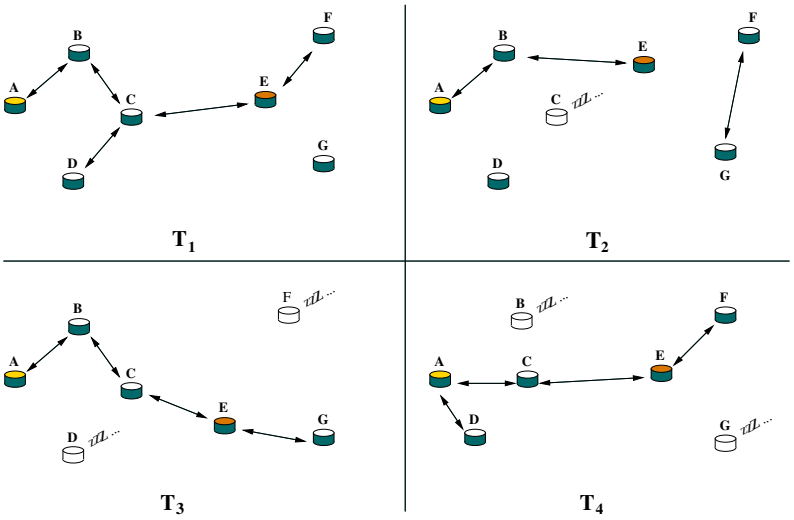


Figure A.1: The evolution of a MANET over time. The indices correspond to successive snapshots in time. "Zzz" indicates a sleeping node.

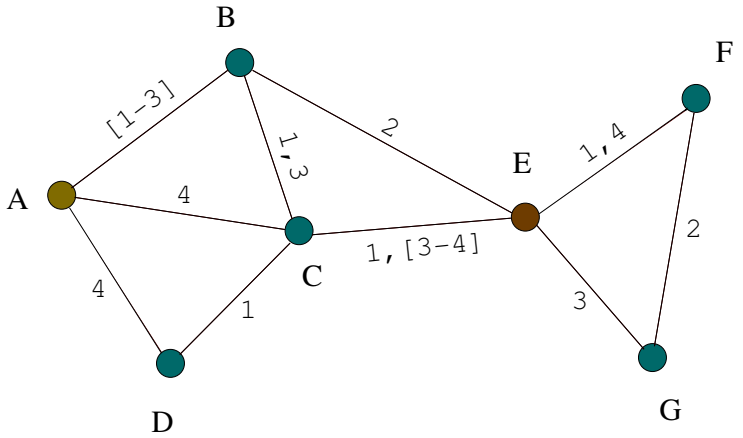


Figure A.2: Evolving graph corresponding to the MANET in Fig. A.1. Edges are labeled with corresponding presence time intervals. Observe that *{E,G,F}* is not a valid journey, since the edge *{G,F}* exists only in the past with respect to *{E,G}*.

Concisely, an evolving graph is an indexed sequence of τ subgraphs of a given graph, where the subgraph at a given index corresponds to the network connectivity at the time interval indicated by the index number, as shown in Fig. A.2.

The time domain is further incorporated into the model by restricting *journeys* (i.e., the equivalent of *paths over time*) to never move into edges which only existed in past subgraphs. A journey in an evolving graph is thus a path in the underlying graph whose edge time-labels are in a non-decreasing order. Now, it is easy to see in Fig. A.2 that

D, C, E, F, G is a journey, as mentioned above. Further, note that D, C, E, G is also a journey, with less hops, but delivering the message later (in time interval 3 instead of 2), giving raise to different objective functions that may be optimized.

A.3.1 Journey Metrics

In the pursue of an optimal journey in networks with known connectivity patterns, three metrics have been formalized until now for EG [139]. They are the *foremost*, *shortest*, and *fastest journey*, which find, respectively, the earliest arrival date, the minimum number of hops, and the minimum delay (time span) route. These three parameters can be individually optimized in polynomial time [139].

We use in this work the *Foremost Journey* algorithm, which computes from a source node s the journeys that arrive the earliest as possible on all other nodes. The algorithm to compute such journeys can be seen as an adaptation of *Dijkstra*'s shortest paths algorithm [142], and is detailed below.

A.3.2 Foremost Journey Algorithm

Remind that, in order to compute shortest paths, the usual Dijkstra's algorithm proceeds by building a set C of *closed* vertices, for which the shortest paths have already been computed, then choosing a vertex u not in C whose shortest path estimate, $d(u)$, is minimum, and adding u to C, i.e., closing u. At this point, all arcs from u to $V - C$ are *opened*, i.e., they are examined and the respective shortest path estimate, d, is updated for all endpoints. In order to have quick access to the best shortest path estimate, the algorithm keeps a min-heap priority queue Q with all vertices in $V - C$, with key d. Note that d is initialized with ∞ for all vertices but for s, which has $d = 0$ (in terms of routing protocols, d needs to be initialized with the current time t).

The main observation in Dijkstra's method is that prefix paths of shortest paths are shortest paths themselves. Unfortunately, the prefix journey of a foremost journey is not necessarily a foremost journey (e.g., considering the EG in Fig. A.2, a message sent at time interval 1 from A to G can use the journey A, B, E, G, with the packet reaching G at time interval 3. The prefix journey A, B, E will reach E at time interval 2, although this is not a foremost journey from A to E, which is in fact A, B, C, E, arriving at moment 1). On the other hand, it was proven that there exists at least one foremost journey with such a property in an evolving graph [146, 139].

To compute the *Foremost journey* starting at time t from s to all other nodes, we use a direct adaptation of *Dijkstra*, sketched below, as detailed in [139]:

1. Set $d(s) = t$, and $d(u) = \infty$ for all other nodes.
2. Initialize *min-heap* Q, sorted by d, with only s in the root.
3. While Q $\neq \emptyset$ do
   (a) $x \leftarrow$ root of heap Q.

(b) Delete the root of heap Q.

(c) For each open neighbor v of x do

i. Compute first valid edge schedule time greater or equal to current time step

ii. Insert v in the heap Q if it was not there already.

iii. If needed, update $d(v)$ and its key.

(d) Update the heap Q.

(e) Close x. Insert it in the foremost journeys tree.

At the end, we have a tree with the *Foremost journey* traversal paths from s (starting at time t) to all other nodes.

Note that the computation of the first valid edge schedule done at the inner loop may take into account the traversal time for the edge, i.e., the duration of the transmission, if needed. This is the case of *timed* evolving graph [146].

A foremost journey from a source node s to all other nodes can thus be computed in $O(M(log\delta_E + logN))$ time, where N is the number of vertices, M is the number of edges and δ_E is the maximum number of presence time intervals over all edges. The term $log\delta_E$ stems from the lookups into the schedule list of intervals, which is required to decide the earliest time interval in which to cross each visited edge.

The routing protocol originated by this algorithm will be henceforth referred to as $EG_{Foremost}$ and is detailed in next section.

A.4 Routing Protocols for MANETs

A great deal of work has been produced comparing the performance of the four main MANET routing protocols, namely DSDV, DSR, AODV and OLSR, that were designed to provide routes in connected networks [137, 138, 143, 158, 149].

The first of these routing protocols, Destination-Sequenced Distance Vector (DSDV) described in [156], is a *proactive* table-driven protocol based on the distributed Bellman-Ford algorithm, with loop-freedom improvement. Each node has a routing table for all reachable nodes, which stores for each destination the next-hop, the number of hops, and a sequence number. DSDV requires periodical flooding to update the routing table.

Dynamic Source Routing (DSR) [151] is a *reactive* protocol, allowing nodes to dynamically discover a route to destination, on demand. Such routes are stored in a route cache to enhance the performance. Source routing means that each packet carries in its header the complete ordered list of nodes (the path) to the destination, so that the forwarding nodes do not need to have the routing information. There is a clear compromise between the size of routing tables and packet size.

The Ad-hoc On-Demand Distance Vector Routing (AODV) [157] is based on DSDV and DSR. AODV is also a *reactive* protocol, which requests a route when needed, and maintain a traditional routing table to destinations in use. A routing table entry is *expired* if not used recently.

Finally, Optimized Link State Routing (OLSR) [149], is a *proactive* table-driven protocol and inherits the use of link state algorithm, using shortest path first forwarding. It periodically exchanges the topology information with neighbors, and every node maintains the topology of the whole network. To minimize flooding, OLSR uses nodes that act as Multi Point Relays (MPR). Only these special nodes are responsible for forwarding control traffic. As DSDV, this is a proactive protocol, so the routing paths are available immediately when needed.

There are many other routing protocols [166, 134] with specialized characteristics. We are not going to evaluate them here, mainly because this experiment aims to compare $EG_{Foremost}$ with massively tested and analyzed protocols, as are the four above.

The $EG_{Foremost}$ protocol

One of the objectives of this work is to investigate the behavior of the *EG* foremost algorithm as a theoretical optimal routing protocol. In this respect, it is important to mention that its implementation as a distributed routing protocol, i.e., with control messages to distribute the *EG*, keeping it up to date and fail safe mechanisms, is out of the scope of this study. Therefore, we assume that all nodes have the knowledge of the *EG*, which makes straightforward the implementation of the protocol. This is still true if the assumption holds for transmitting nodes only.

Let *edge schedules* be a set of time intervals representing the existence of link-connectivity between two nodes. An *edge* exists when two nodes are in the range of each other. The evolving graph of a dynamic network can be represented by a list of *edge schedules* for each pair of nodes. Thus, each node has a list of its neighbors at a given time (as detailed in Section A.5.1, the *EG* is calculated from the mobility scenario and a well known transmission range).

When a packet arrives at the routing layer of node u at time t_{now}, the node computes the *foremost journey* (as shown in Section A.3.2) from the packet source to its destination.

Suppose that the journey next hop is the node v at time t_v. If $t_v = t_{now}$, then both nodes are in the range of each other (i.e, there is an edge in *edge schedules* of (u, v) at time t_{now}), and the packet is readily forwarded to v. Otherwise, if $t_v > t_{now}$, the nodes are not reachable, and the node u must schedule the transmission of the packet to the time t_v ahead. This is the earliest feasible edge present in *edge schedules* of (u, v) with time greater than t_{now}.

In Table A.1 we show the corresponding *edge schedules* as shown in the example of Fig. A.2. Note that the *edge schedules* are the presence time intervals at the labels.

As an example, the foremost journey for a message sent at time index 1 from D to G will be D, C, E, F, G. The packet will reach F at the same time index 1, then afterwards F will schedule to send the packet to G at the earliest edge presence in the *edge schedules* of $F - G$. Hence the packet will be sent by F at time index 2.

If two routes have the same time length when computing the *foremost journey*, the one with less number of hops will be chosen for routing, and if they even have the same number of hops, the route with the smaller node ID will be chosen. This ensures a total

Table A.1: Edge schedules for the EG in Fig. A.2.

Node pair	Edge schedules
A – B	1, 2, 3
A – C	4
A – D	4
B – C	1, 3
B – E	2
C – D	1
C – E	1, 3, 4
E – F	1, 4
E – G	3
F – G	2

order when choosing nodes (i.e. sorting nodes in the heap).

In the examples above, the edges traversal times are not taken into account, for the sake of a simpler illustration. In the implementation and in all simulations the edge traversal time was indeed estimated (we include a discussion about it in Section A.7.1)

A.5 Simulation Environment

We have conducted our performance analyses using the *NS2* [155] simulator version 2.31, with the mobile extensions by CMU Monarch which provides IEEE 802.11 Medium Access Control (MAC) protocol [148], and realistic radio and physical layer with the Two Ray Ground propagation model. The radio model uses characteristics similar to the 802.11g standards, modeled as a shared-media radio with a nominal bit rate of 54 MB/s and an omni-directional antenna with nominal propagation range of 50m. The RTS/CTS radio scheme was turned on in our experiments.

In the simulations, *60 nodes* are randomly placed in terrains with size varying from 300m x 300m, 300m x 200m and a larger one with 1000m x 1000m area, these parameters lead to different density of nodes and are discussed ahead. The simulation time is *3000 seconds*, and the first 1000 seconds of each simulation are not considered, for the sake of the stability of the movements [169]. A number of *10 constant bit rate (CBR) UDP traffic flows* are chosen between node pairs (nodes 1 to 10 are the transmitters and nodes 40 to 49 are the receivers). The average *traffic rate is one packet/sec*, with 256 bytes long packets. Each flow starts to generate packets at random at instant 1000 seconds of simulation and remains transmitting until the end. This low data rate is chosen to address a sensor network-like environment, where dedicated sensor nodes are constantly collecting data. The interface queue length at link layer (IFQ) was doubled from the default 50 packets to 100 packets at each node. We do not use TCP for the simulations, as we did not want to investigate TCP particularities, which uses flow control, retransmit features and so on. We are solely interested in the behaviour of the routing protocols.

In these experiments (in contrast with [5]) we decided to disable the ARP (Address

Resolution Protocol) of mobile nodes, agreeing with the arguments in [140] by Carter, Yi and Kravets, in that MANETs need to have their own ways of neighbour discovering. This is because, in *NS2*, the address resolution is an approximation of the BSD Unix ARP, so the resolution is performed on-demand as packets arrive from the application layer, and the buffer size to each neighbor is only one single packet. When an address resolution is in process, all incoming packets to the same destination will be dropped. This leads to a great amount of dropped packets in ARP mode.

Therefore, in our simulations we assume that each node has the IP and MAC addresses of its neighbours.

A.5.1 Mobility Models

We divided our experiments into two separate kinds of scenarios. One uses the popular *Random Waypoint* (RWP) model [151], while the other uses a new mobility model, called *Intermittent Model*, which is more suitable to the case of WSNs and is explained below.

Random Waypoint Mobility Model

In the *Random Waypoint* (RWP) mobility model, a mobile node moves to a randomly chosen location, with speed randomly chosen from 1 to 3 m/s according to a uniform distribution, and pauses for a uniformly chosen time between 0 and PAUSETIME. The simulation was run with values of PAUSETIME varying from 0 (continuous motion) to 1500 seconds (very low mobility in the network). To avoid known problems of the RWP model, as shown in [169], we are using only non-zero values of minimum speed. The use of this classical scenario, yet with its known limitations, is important to compare the results with other performance studies. The program *BonnMontion* from the University of Bonn [136], was used to generate these scenarios.

Intermittent Mobility Model

This mobility model is based on fixed nodes whose positions are chosen randomly with uniform distribution. The nodes remain uninterruptedly turning themselves *on* and *off* (awake and sleep) in given periods (see Fig. A.3). Here, the parameters we change are the SLEEPPROB (ranging from 0 to 50%), and the HOLDTIME (ranging from 15s to 180s). In the beginning of the simulation each node is awake, and for the entire simulation it has a SLEEPPROB probability to go to sleep (turning itself off). If a node goes to sleep, it remains off for a uniformly randomly chosen HOLDTIME. Once this time expires, the node is turned on and stays awake for another period based on HOLDTIME. If a node does not go to sleep (probability of $1 -$ SLEEPPROB), it will stay awake for HOLDTIME before trying again. It is important to point out that HOLDTIME is recomputed at each state change.

This new model aims to capture the behavior of networks with functioning schedules, like wireless sensor networks. Chapter 7 of [152] presents several related node models.

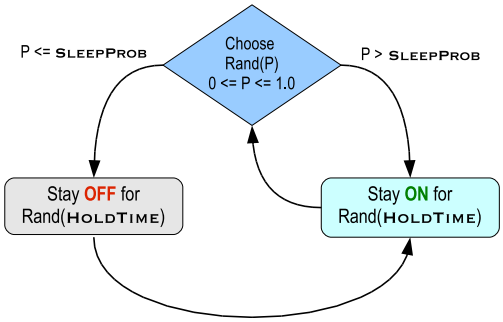


Figure A.3: Lifecycle of a node in the Intermittent Mobility Model.

With the aim of pursuing a constant traffic rate in the network, the nodes which generate the traffic flows never go to sleep during the whole simulation.

In both models, for each evaluated parameter we created 20 random scenarios with different random seeds. Therefore, we ran the simulation 320 times for the first model, i.e., 20 times for each value of PAUSETIME: $0, 100, 200, 300, ..., 1500$ seconds, and 440 times for the *Intermittent Model*, i.e., 20 times for each combination of SLEEPPROB: $0, 5, 10, ..., 50\%$ and HOLDTIME: 15, 180 seconds. Note that in the second scenario we change two parameters, and the value of 0% of SLEEPPROB means that no one goes to sleep, hence the network remains static from the beginning to the end of the simulation, which is relevant as a reference. All five routing protocols (AODV, DSDV, DSR, OLSR, $EG_{Foremost}$) were run on the same 760 scenarios. Thus, identical mobility and traffic scenarios (as they have the same seeds) were used across protocols.

NS2 Implementation Details

Each simulation in *NS2* generates a trace file, containing all communications that have been done between nodes, including the MAC layer. These files were analyzed to consolidate the results, which are shown in Section A.6.

The implementations used to evaluate DSDV, AODV and DSR protocols are the ones provided in the *NS2* package. All of them were implemented by the CMU Monarch group [159]. Concerning OLSR, we used the implementation by Francisco J. Ros [160] (UM-OLSR) version 0.8.8. In all of them we used the default parameters and constants. The AODV implementation, as of *NS2* version 2.31, contains some further optimization codes from [158].

In *NS2*, the node states "sleeping" and "awaken" used in the *intermittent model* scenarios are implemented using the commands *on* and *off* already implemented in the mobile agent. But the last version of *NS2* did not work correctly with this command. To ensure correction, we removed the line that do a `reset-state` in the command `off` in the `mobilenode.cc` file, which is only used by direct diffusion agent.

In order to generate the *evolving graph* that will be used as input to our protocol, we use the following strategy. Mobility in *NS2* is usually represented by a script in Tcl language containing scheduling commands, e.g. *setdest*, *on*, or *off*. In general it is

saved in a separate file used by the simulation. Therefore, we wrote a program (*calceg*) that reads a mobility file used by *NS2* and captures the node movements to generate a corresponding *EG*. This *EG* is afterwards used as input for the foremost journey *EG* based protocol ($EG_{Foremost}$), as shown in section A.3.2. Note that, to calculate the *EG*, the transmission and reception range of each node must be taken into account; in our case is fixed at exactly 50 meters . We build our EGs from the mobility models after a post mortem analysis in a continuous way. It does not use any discretization technique. We calculate analytically the exact moments where the nodes enter in the communication range of each other, and also when they are not able to communicate anymore.

Before the simulation begins, this *EG* of the network is distributed among all nodes. Each node in the simulated network knows the connectivity pattern of the network during the simulation. This is important for benchmarking purposes, since the EG may be generated and used as a reference when developing or tuning routing protocols and mobility models. Furthermore, there are many practical situations, like those shown in [153, 141, 168, 147, 134], in which an *EG* can be built before the routing phase.

From a theoretical perspective, the EG-based protocol can be considered as a distributed protocol, since there is no central controller. The fact that every node possesses the full description of the EG is part of the set of our hypothesis. If each node has local knowledge about future connectivity, then global knowledge might be obtained by a dissemination mechanism like the one deployed by OLSR. If this is the case, then a careful study of the impact of this routing overhead should be conducted.

A.6 Simulation Results

In this section we show the results obtained by simulation of a DDTN composed of wireless mobile nodes that move around, go to sleep for a while, and communicate with each other.

As in the case of the mobility models, the results shown here are separated in two parts, one using the RWP mobility model, and another using the *Intermittent Model*.

We focused our analysis in four main metrics:

- **Average throughput**: The average number of packets received per amount of time (from the first packet sent to the last packet received);
- **Average end-to-end delay**: The average time between sending and successfully receiving a packet;
- **Ratio of dropped packets by no route (NRTE)**: Fraction of dropped packets by *no available route* per total number of sent packets;
- **Ratio of dropped packets by Interface Queue overflow (IFQ)**: Fraction of dropped packets by *link queue overflow* per total number of sent packets. This queue is at the

[0]The EG implementation and related software can be found at http://www.ime.usp.br/ jm/mobidyn/-software.

link layer, i.e., it is used when the routing layer wants to effectively send a packet to be delivered.

Error bars on the figures indicate a 95 percent confidence interval.

A.6.1 Random Waypoint Mobility Model

As mentioned earlier, in the RWP scenario the control parameter is the PAUSETIME. Low values of PAUSETIME mean *high* mobility and high values of PAUSETIME mean *low* mobility.

As shown in Fig. A.4, the $EG_{Foremost}$ performance has the lowest values of *dropped ratio* for all pause times, followed closely by reactive protocols. With low mobility (high pause times), the number of dropped packets raise to 5.7% of the transmitted packets. DSR has good values compared to others and it is surprising that AODV does not perform well (i.e. better than DSR) at high mobility values, in contrast to what was as shown by Perkins, Royer, Das, and Marina in [158]. This behaviour may be explained by the very low network load of our simulation (1 packet/s with 10 traffic sources).

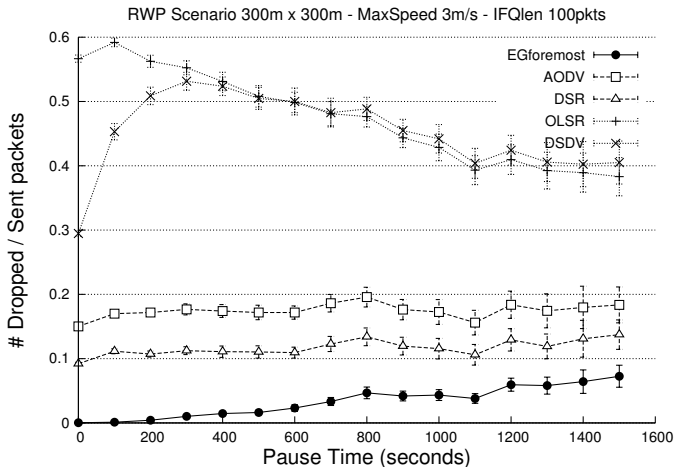


Figure A.4: Drop ratio as a function of PAUSETIME (mobility).

The pro-active, table-driven protocols do not perform well in this first scenario, since the periodically updated tables do not get updated as fast as needed. The drop-ratio for DSDV and OLSR are the worst, ranging from 60% in high mobility to 39% in low mobility simulations, showing that they are very sensitive to mobility. They perform better when the mobility decreases. Finally, $EG_{Foremost}$ produced the best results in this metric in all RWP simulations scenarios. The theoretical throughput of the network is $1 * 256 * 8 = 2048$ bits/s and the $EG_{Foremost}$ results are very close to it.

As stated before, the $EG_{Foremost}$, as expected, performed well in the above scenarios, comforting our claim that it may serve as a benchmark when evaluating the performance of other protocols on similar mobility models.

Scenarios with low density of nodes

In the above scenarios, a terrain with 300m x 300m area was used. The coverage ratio, i.e. the ratio between the sum of nodes transmission ranges divided by the field area was 5.2. To simulate a scattered scenario we raised the terrain size to 1000m x 1000m, which leads to a coverage ratio of 0.47. Using this scenario we can measure how routing protocols behave when nodes remain disconnected for a long time, which is generally the case of DTNs.

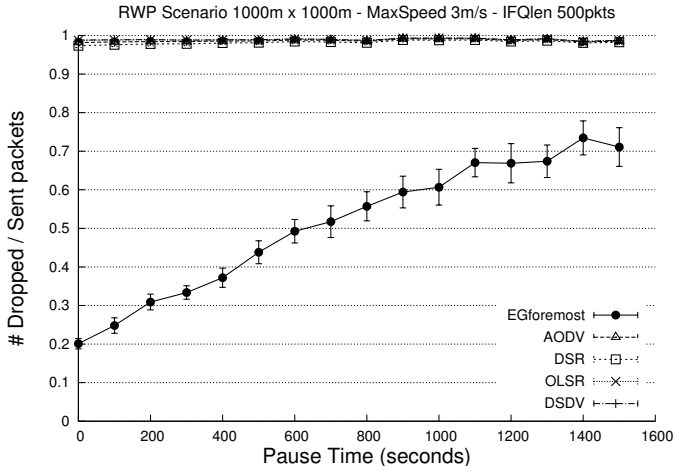


Figure A.5: Drop ratio as a function of mobility using a low density of nodes scenario.

In Fig. A.5 one can see that regular ad hoc protocols are not adapted to a disconnected environment. However, the $EG_{Foremost}$ protocol shows that a high amount of packets could still be delivered. In the high mobility scenarios the drop ratio was close to 20% and it increases to 70% of drops in the low mobility scenarios. It should be noted that the delivered packets have a high average end-to-end delay, ranging from 285 seconds with PAUSETIME 0 to 606 seconds with PAUSETIME 1600s.

A.6.2 Intermittent Model

The intermittent scenario is well adapted to DDTNs, since the nodes' on/off dynamics are easily predicted or even pre-programmed.

We changed the values of SLEEPPROB from 0 to 50%. High probability to sleep means that the network has a low connectivity, i.e., a large quantity of nodes are disconnected from each other because many of them are in sleep state.

The values of HOLDTIME control how slow the nodes change their states (on/off) or, in other words, *how often connections among nodes change*. Low values of HOLDTIME means high dynamics and vice versa. Fig. A.6 illustrates this behavior of the Intermittent Model scenario. In the low connectivity scenarios (SLEEPPROB at 50%) the number of topology changes for a HOLDTIME of 15s is 12 times the number of those with HOLDTIME 180s.

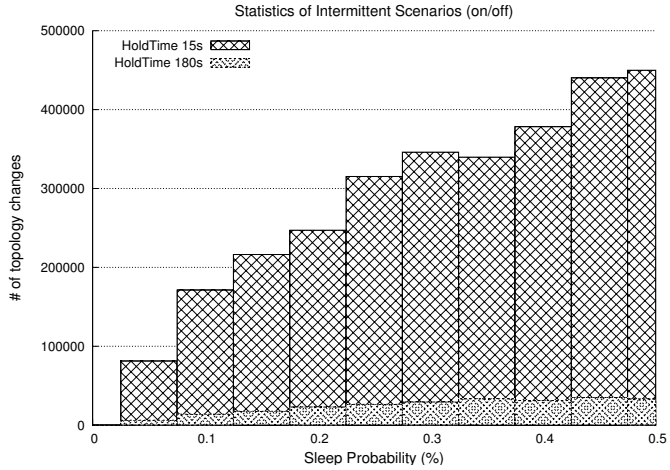
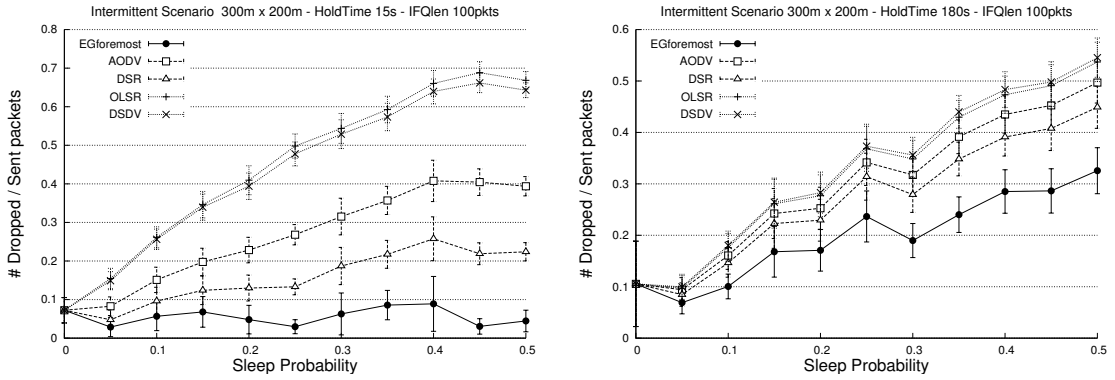


Figure A.6: Number of changes in the network topology in the Intermittent Model scenario for different values of SLEEPPROB.



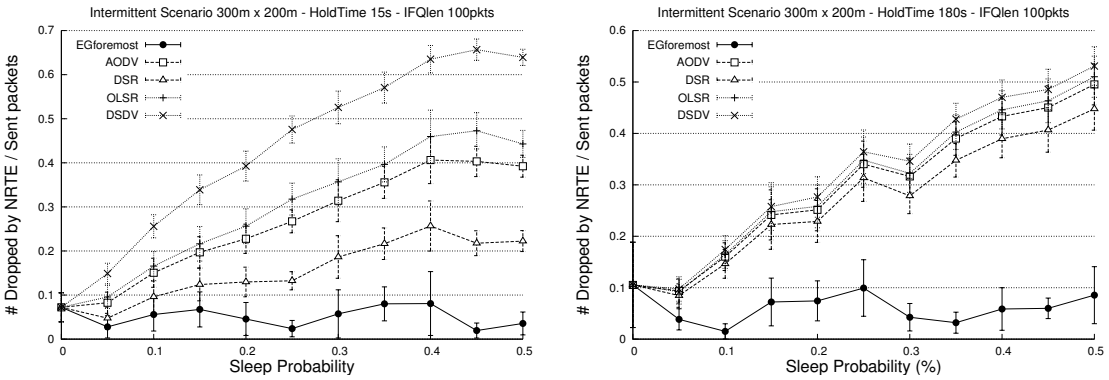
(a) HOLDTIME 15s (b) HOLDTIME 180s

Figure A.7: Total drop rate as a function of SLEEPPROB (connectivity).

As we will see in the following, this model harness some characteristics of connectivity dynamics that are not captured with the former RWP model.

After some experimentation of the simulation parameters, we reduced the size of the simulation field to a rectangular 300x200m, so the coverage ratio is raised to 7.85. Otherwise, the number of dropped packets was very high, as the nodes in these intermittent scenarios do not move.

In Fig. A.7, we show again that the $EG_{Foremost}$ has better values of *drop rate*. Observing the results, in the case of high dynamics (HOLDTIME equal 15s), the drop rate of $EG_{Foremost}$ is close to zero in all connectivity scenarios. On the other hand, in the case of low dynamics (Fig. A.7b), the values of drop rate for EG decrease, with the other protocols, as the connectivity decreases (from 0 to 50% SLEEPPROB). This $EG_{Foremost}$ loss of performance (reaching 32% less if compared to the rate of flow) is not related to non-existing routes, because, as shown in the graphics of Fig. A.8, the number of packets



(a) HOLDTIME 15s (b) HOLDTIME 180s

Figure A.8: Drop packets by NRTE ratio as a function of SLEEPPROB (connectivity).

dropped by NRTE is very low (an average of 5% on the both scenarios).

The values of $EG_{Foremost}$ in Figs. A.8a and A.8b show that the number of *dropped packets by NRTE* is a lower bound in this metric, i.e., when $EG_{Foremost}$ drops a packet by NRTE, it means that the requested path does not exist in any moment of time. Therefore, $EG_{Foremost}$ may again be used as a benchmark to measure how good the other protocols are performing.

The increase of the total drop rate on low connectivity scenarios is not due to inexistent routes, but to other reasons analysed below.

A.7 Further analyses and improvements

The goal of the foremost journey algorithm is to calculate journeys that reach the destination as soon as possible. However, in this process some packets may wait for a long time for a connection to be established, and this waiting time is computed in the end-to-end delay metric. In contrast, in the simulation of the other protocols some of these "late" delivered packets are just being dropped and do not contribute to the end-to-end delay count. In other words, when using the Foremost Journey EG based routing algorithm, the packets end-to-end average delay is usually larger, even though it was proven in [139] that $EG_{Foremost}$ ensures that the packets will reach the destination as soon as possible if a journey exists in the network.

This gives the opportunity to add a new parameter *MaxDelay* in the EG algorithms, which is the maximum delay time that a packet could wait to be delivered in the DDTN. If the calculated delay time is greater than *MaxDelay*, the packet could be dropped instead of overflowing the network.

Now, remind that almost all packets are delivered by the $EG_{Foremost}$ protocol (see Fig. A.4). Hence, to be fair with the foremost algorithm and better perceive the performance of the $EG_{Foremost}$ protocol, we calculated the average end-to-end delay taking into account *only the packets that have been successfully delivered* at the destination in all protocols

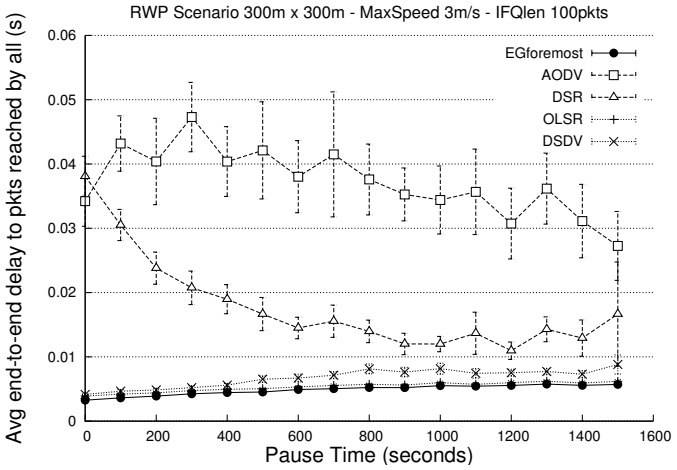


Figure A.9: Average end-to-end delay of packets successfully delivered in all protocols.

(i.e. the intersection of received packets). The results in the Fig. A.9 shows $EG_{Foremost}$ as a lower bound in the *end-to-end delay* in this scenario. The reactive protocols, AODV and DSR, do not have a smooth curve, due to the induced delays from the route discovering process. The variance of their values is very high too. It is important to point out the good performance of the OLSR protocol, very close to that of the $EG_{Foremost}$.

A.7.1 Bottlenecks and Congestion

The intrinsic behavior of $EG_{Foremost}$, namely to schedule packets to be sent when some connections are established, yields the problem of *bottlenecks* [162], since a large quantity of packets are scheduled to be sent at the same moment, and the link interface queue (IFQ) cannot hold that incoming traffic (its size is 100 packets). Furthermore, the *EG* algorithm do not have any mechanism to balance the flows and many flows with different sources could potentially use the same path, even when some other ones are available at the same cost (i.e. arriving at the same time).

We note that the simulation done using the *random waypoint model* does not suffer from this effect. Due to the high mobility of such a scenario, the *bottlenecks* do not show expressive values. This characteristic appeared in the low connectivity and low dynamics scenarios of the *Intermittent Model*, in which the nodes in the evolving graph remain disconnected for long time periods, and a large quantity of packets are then scheduled to the moment when these nodes wake-up.

In Fig. A.10 we see the high values of *dropped packets by IFQ overflow* on a low connectivity scenario (24% of packets are dropped by IFQ at SLEEPPROB 50%).

The histogram in Fig. A.11 shows the number of *dropped packets over time* for one single simulation, namely with SLEEPPROB at 50% and HOLDTIME at 180s. It shows that in $EG_{Foremost}$ the packets are dropped in a burst, again because important nodes go to sleep for a long time and when they wake-up, a large quantity of packets are waiting to be sent, overflowing the queues that then drop the packets.

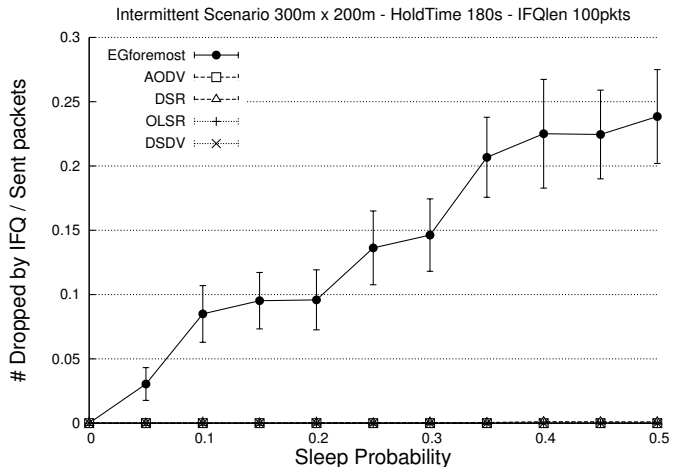


Figure A.10: Number of dropped packets by IFQ overflow on a HOLDTIME 180s scenario.

A.7.2 Congestion with varying flows over time

For the sake of clarity and to allow for comparisons, we show below the results of some experiments using a variable offered load on the network, since the previous experiments have very low traffic demand.

The same parameters from the intermittent model scenarios were used, but the traffic flow rate was changed during time according to a non negative senoidal curve, with discretized values of CBR chosen from the following rates: 0.5, 1, 2, 4, 6, 8, 10, 12, ..., 20 packets/second. The gray shaded area in Fig. A.12 represents the rate of traffic generated in the network by the 10 traffic sources.

Again, the number of dropped packets by $EG_{Foremost}$ is the one with high peaks, and with the increase of traffic rate its behaviour is even worse. Note that, even at time 2000 seconds, when the traffic rate is very low (one packet every 2 seconds) the number of drops is high. This is due to many packets that are scheduled to be delivered at that same point in time. Almost all packets dropped by the EG protocol resulted from IFQ overflow.

A.7.3 Reducing congestion in $EG_{Foremost}$

The discussion above shows the importance of managing the flows of data during time, even when using EG protocols as a reference. Unfortunately, balancing flows in evolving graphs is still an open problem in Graph Theory. Therefore, we tried three different empirical approaches to address the packet dropping problem caused by bottlenecks in $EG_{Foremost}$ (see Table A.2, below):

1. **Jitter**: Add an enforced jitter (a random value uniformly chosen between 0 to T seconds) at sending time to each packet. We experimented with values 0.1 and 0.5 of T;

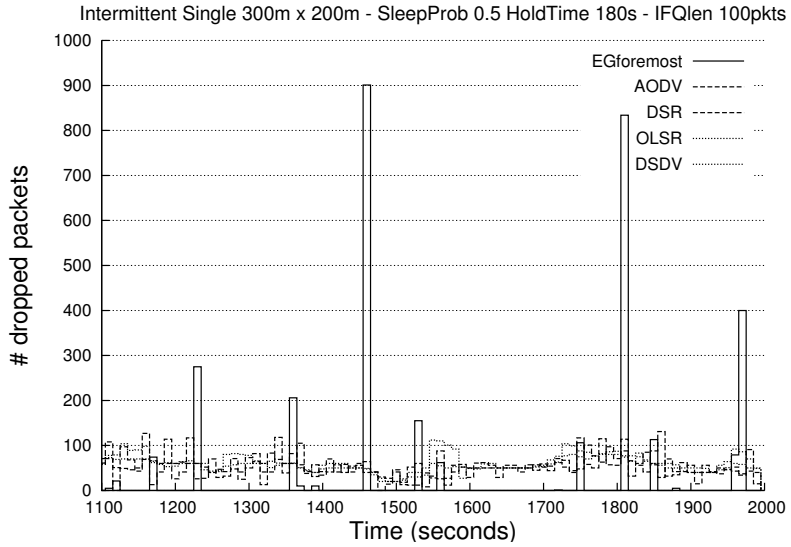


Figure A.11: Number of dropped packets over time for one single Intermittent Scenario (SleepProb. 50% and HOLDTIME 180s).

Table A.2: Approaches used to minimize the bottlenecks (Fig. A.13).

Packet	IFQlen 500	Jitter 0.5s	SmartJitter
p_1	t	t + rnd(0.5)	t
p_2	t	t + rnd(0.5)	t
...	t	t + rnd(0.5)	t
p_{50}	t	t + rnd(0.5)	t
p_{51}	t	t + rnd(0.5)	$t + \delta$
p_{52}	t	t + rnd(0.5)	$t + 2 * \delta$
p_n	t	t + rnd(0.5)	$t + (n - IFQlen) * \delta$

2. **SmartJitter**: Add a fixed size jitter only when some connection is overflown. The size of this jitter is the same as the average edge traversal time;

3. **Increase the IFQ length**: Raise buffer size of the interface queue from 100 to 500 packets. We also showed, as a reference, the values with IFQ length of 50 packets.

The results of the experiments with these three approaches can be seen in Fig. A.13. IfqLen 100pkts is the reference curve, as seen in Fig. A.10. The error bars were removed from this figure to increase readability, their values were between 0.01 and 0.04, and were similar among the experiments.

The first approach is the enforced random chosen jitter when sending each packet at each node, ranging from 0 to one tenth of a second (0.1s) and half of a second (0.5s). In the average, the number of dropped packets decreased 32% and 75% respectively. One drawback of this approach is the high values of end-to-end delay, due to the extra time added at each scheduled packet.

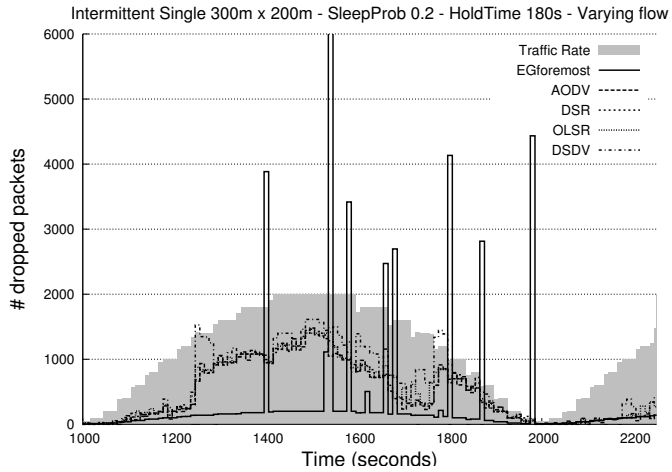


Figure A.12: Number of dropped packets over time for one single Intermittent Scenario with varying flow. The SLEEPPROB value is 20% and HOLDTIME is 180 seconds.

The second approach, the SmartJitter, is an improvement of the former. Here, when the queues (one for each pair of neighbours) are not full, the nodes can send packets to other nodes without any jitter on sending time. However, when some node fills the IFQ, then a fixed size jitter is added to each subsequent packet to be sent. The calculation of the SmartJitter time at node u sending a packet to v at time t_v is shown in the following schema:

if $npkts[t_v] \leq IFQlen$ **then**
  $smartjitter = 0$
else
  $smartjitter = \delta * (npkts[t_v] - IFQlen)$
end if
$npkts[t_v] = npkts[t_v] + 1$

At the end, the sent time will be $t_v = t_v + smartjitter$ and $npkts[t_v]$ is the number of packets already scheduled to use the edge from u to v at time t_v. $IFQlen$ is the interface queue length, and δ is the average traversal time of one-hop transmission. We used the value 3.6 ms in all simulations, which was estimated from the average value of one-hop transmission with packet size of 256 bytes. The value of the traversal time is linearly dependent on the size of the packet. This value was obtained through simulations using similar network loads. The traversal time used in the EG foremost algorithm is the same estimated δ value.

With the SmartJitter, the number of dropped packets also decreased 75% compared to the original $EG_{Foremost}$. The values reached by the SmartJitter are similar to ones with the enforced jitter of 0.5s, thus, in the former case the average end-to-end delay is 10% less than the enforced jitter.

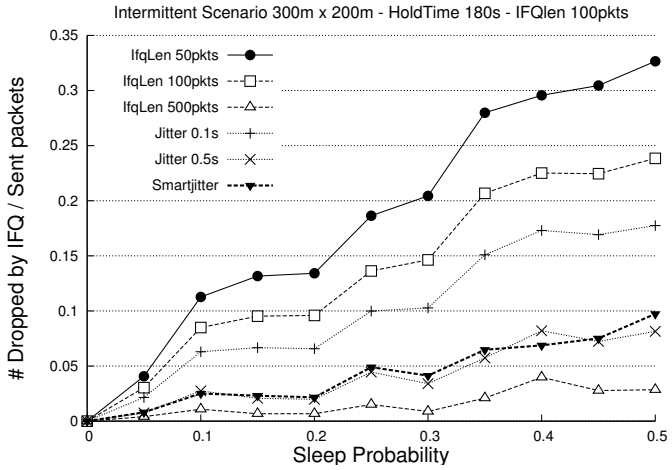


Figure A.13: Number of dropped packets by IFQ overflow with different solutions to minimize the drop rate (enforced jitter, smart jitter and raise de IFQ length to 500pkts).

We note that the best solution is to increase the default length of the IFQ from 100 to 500 packets. In this case, the values of dropped packets decrease 92%. This shows that the SmartJitter is a good solution when the size of IFQ buffer cannot be changed.

Finally, we tested the introduction of the SmartJitter at the first scheduled packet (instead of waiting for the queue to become full). However, the end-to-end delay increased, while the number of packets dropped remained roughly the same. This is the reason why we do not report these results here. As a matter of fact, if the number of $npkts$ is very high compared to the queue size, a packet could be scheduled to be sent a long time after its original time t_v, at which point the edge could not be there anymore.

A.8 Conclusion

Our contribution in this study is to show that an *EG* based routing protocol is well suited for networks with known connectivity patterns, like DDTNs, and that the model as a whole may be a powerful tool for the development of DDTN routing protocols, even in practical scenarios.

The *EG* based protocol has been formalized to provide optimal routing according to its metrics. We implemented the *Foremost Journey* and performed extensive simulations using *NS2*. We compared the performance of the new $EG_{Foremost}$ with four major MANET protocols: DSDV, DSR, AODV and OLSR. The results showed that the drop ratio by no available route (NRTE) using EG was the lowest compared to all protocols in all metrics. Consequently, EG values can be used as a benchmark in many cases.

This first implementation of an *EG* based protocol opens some avenues for further detailed research. For instance, a routing *bottleneck* appears when most used nodes become unavailable for a long period of time, causing overhead when they reappear in the network, leading to packets being dropped due to collision and queue overflows. The use of high values of enforced jitter time when sending packets can minimize the drop

rate, but is not feasible in regular protocols. We introduced the SmartJitter as an option to minimize the congestion and achieved good results. The development of a good *EG* adaptive algorithm could possibly manage this problem, anticipating congested nodes in order to find out alternative routing paths. To date, however, there are no theoretical results on flows over *EG*, which is in itself a very interesting open problem.

It should be noted that the high values of average end-to-end delay is an inherent characteristic of the communication network dynamics. In the case of *EG* based protocols, on which the *foremost journey* metric is studied, the end-to-end delay is in any case the minimum arrival date for a packet. If long delays need to be managed, then one policy could be to drop packets that are aged in the network, or – even better – use a *fastest delay* approach, described in[139], instead of the *foremost journey* as done here.

Future work includes implementation of other *EG* based protocols with different metrics, like *shortest path* and *fastest delay*. A natural extension to this work is related to the deviations in the predicted network dynamics, on which the actual *EG* used by the nodes is not accurate anymore. This engenders the utilization of a model with stochastic predictable behaviour to better address such variations.

Another open question relates to the case where global knowledge does not exist and it cannot be easily computed locally. In such cases, as suggested by a reviewer, it would be important to address the trade-off which exists between the amount and accuracy of knowledge of network topology and the routing performance. In other words, EG-foremost is a benchmark but its implementation may present a high overhead that is required to achieve complete knowledge of the topology and its evolution; therefore the other routing schemes will be compared as achieving different performances with different routing overheads.

Finally, it would be worth providing a framework that unifies the theoretical power expressed by evolving graphs with the engineering aspects captured in DTNs with knowledge oracles. Major advances in the formalisation of dynamic networks are thus to be expected.

APPENDIX B

Corral - Linux Stackable Copy-on-Write File Versioning Device

A very promising feature of modern file systems is the ability to maintain multiple versions of the stored data. A possible way to keep all this information is to perform periodic snapshots of the system. The copy-on-write (CoW) method tends to be an efficient way to manage and maintain these snapshots during time.

Our contribution. Based on these techniques we propose the CORRAL system: a virtual block-device with transparent versioning, which uses the Linux device-mapper[1] snapshot as a lower layer. The snapshot method uses a CoW technique to avoid duplicating unchanged blocks between successive versions. As long as the data remains the same, no duplication is done. The CORRAL system can be used with any file system and can be implemented without the need of a kernel or module recompile. A fully functional prototype is written in a Perl script with less than 500 lines and some experiments were done to measure the effectiveness of the proposed system[2].

Related Work. Similar to Plan 9 Fossil [96], Write Anywhere File Layout (WAFL) [63] and Peabody [82], the CORRAL provides periodically snapshots of the entire file system. However, these solutions are implemented at the file system layer. Our proposal is a device that can work with any kind of file system. The Clotho Transparent Data Versioning [50] also share some of its features with our solution, although it has been implemented as a separated kernel module.

The results presented here appeared in [12].

Device Mapper

A Logical Volume Manager (LVM)[3] provides a higher-level view of the storage system than the traditional view of disks and partitions. Advanced virtualization systems, like Linux LVM2 or EVMS are built on top of the device-mapper kernel module, which creates a mapping from the physical sectors of a disk into a logical volume (i.e., block-device). Every I/O request made to the logical volume is thus routed to the right sector on one or more physical disks. There are many possible mappings, named *targets*.

[1] http://sourceware.org/dm/
[2] The source code can be downloaded from http://gforge.inria.fr/projects/corral
[3] http://sourceware.org/lvm2/

In our case, the studied target is the *snapshot* one, which uses the CoW method to consistently freeze the state of a logical block-device at a determined moment. This target creates an image of the device, on which all the subsequent changes are written on a different partition, keeping the original data intact.

Note that the *snapshot* feature of the Linux LVM2 implementation, that are mainly used to consistently backup a file system, does not behave exactly as the snapshot device-mapper described above; more than doing a snapshot the LVM2 uses a target called *snapshot-origin*, which creates another layer of virtualization, inverting the roles of the CoW process.

B.1 CORRAL Design and Architecture

The main goal of the CORRAL is to provide a block level versioning device on top of device-mapper volumes. The expected characteristics of CORRAL are:

- to work with any file system;
- retention of multiple versions;
- deletion of intermediary versions;
- keep current and versioned data on the same disk;
- easy identification of modified blocks;
- operate on a mounted file system;
- use of standard user level tools.

Initialization: The idea of the CORRAL system is to use all the physical disks or volumes as a container of chunks of blocks. Each chunk has the same size, fixed between 4 and 512KB. Some of these physical chunks are marked as *free*, and the others are part of a version on the CORRAL system. The free physical chunks will be mapped to a device named COW, and the combination of the physical chunks representing the last version of the system is mapped to a device named PREV. Not yet modified chunks are mapped using the target *zero* of the the device-mapper, which behaves similar to the `/dev/zero` character-device. See Figure B.1a.

With these two devices we can create a third one, using the device-mapper *snapshot* target already described. This device will be henceforth named CURRENT and will be the one visible to the user, e.g., can be used to install a file system. Any write request to CURRENT will be redirected to the COW, which is the copy-on-write device of the snapshot target. Reads from the CURRENT will come from the PREV for unchanged data, and from the COW if the block was already modified.

Versioning: At any moment it is possible to freeze the CURRENT_i device to create a new image of the system. The steps to create a version $i + 1$ are the following:

1) suspend the snapshot device;

2) read the COW$_i$ metadata and mark the corresponding physical chunks as used;
3) create a new COW$_{i+1}$ device using the remaining free physical chunks;
4) clone the mapping of CURRENT$_i$ into the PREV$_{i+1}$ device;
5) create a new CURRENT$_{i+1}$ snapshot device based on COW$_{i+1}$ and PREV$_{i+1}$;
6) replace CURRENT$_i$ with CURRENT$_{i+1}$ in the snapshot device;
7) resume the snapshot device (now in version $i + 1$).

These operations can be done even on a mounted filesystem. All the pending operations will be blocked until the device is resumed.

Figure B.1 shows an example of usage of the CORRAL system. After the initialization, two blocks are written: 17 and 32 (as shown in Figure B.1b). Then, a version freeze command is executed. Figure B.1c shows the chunks disposition after the version freeze, followed by a write operation to the logical chunk 32 (Figure B.1d). PREV$_i$ shows that the logical chunks 17 and 32 were written in the past and are mapped to the physical chunks 3 and 4, respectively. The write operation allocates the next free chunk in the COW$_i$ to store the new data. This allocation results from a CoW event from the PREV$_i$ and a merge with the new content of the write operation. A mapping from the logical chunk 32 to the physical chunk 5 is created in the kernel's memory and stored on the exception table metadata. All the new data is stored sequentially into the COW$_i$ device, which leads to a shuffle effect on the physical disk because the logical chunks are not stored in the same order on disk.

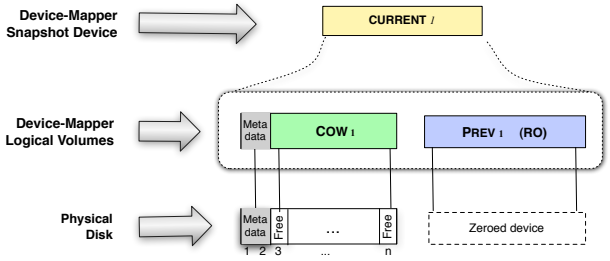
Following the scenario of the Figure B.1d, the logical chunk 32 has now two versions on the disk: the version i is stored at position 4, and version $i + 1$ is stored at position 5. Subsequent writings to the logical chunk 32 during this snapshot will be redirected to the same physical chunk 5. Keeping this mapping information as metadata allows an easy recovery of ancient versions of the system.

To recover an archived version, it is sufficient to read the metadata to create a logical device with a linear mapping from the physical chunks that belongs to the archived version. Removal of old versions can be done with simple operations on the metadata, freeing physical chunks occupied by the corresponding version when their reference count reach zero.

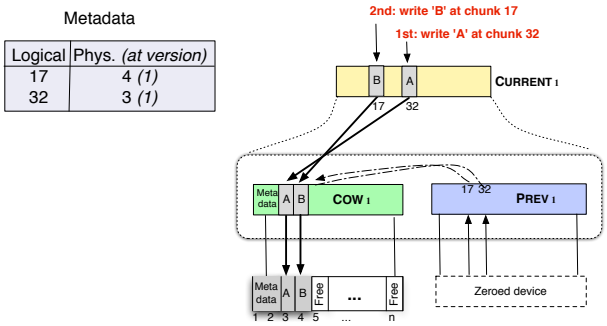
B.2 Experimentation

Our goal is to quantify how file systems are performing when using the CORRAL versioning device with different chunk sizes. We used the microbenchmark Bonnie++, the Postmark transaction benchmark and a Andrew-like benchmark to evaluate the system [118]. All the benchmarks were run with and without CORRAL on three different file systems: Ext2, Ext3 and ReiserFS. A summary of the results are shown in Figures B.2,B.3, and B.4.

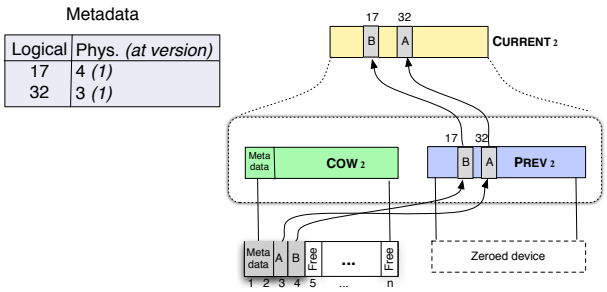
Preliminary results show that the first time a chunk is written, triggering a CoW event causes a significant degradation on the write throughput . Subsequent writes and reads, however, have similar performance compared to the unversioned file system (see Figure B.4). Nevertheless, this system provides an efficient way to maintain a consistent



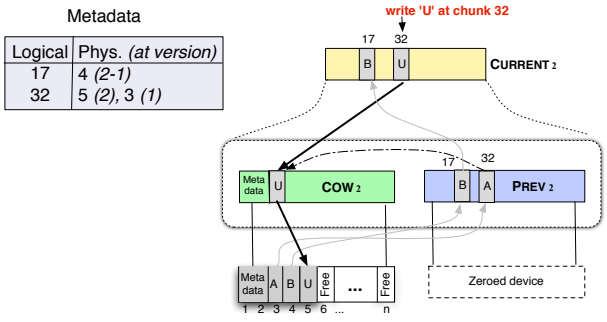
(a) **Version 1** - Initialize



(b) **Version 1** - write into chunks 32 and 17



(c) **Version 2** - freeze



(d) **Version 2** - write again at chunks 32

Figure B.1: Example of use of the CORRAL system over time.

list of versioned chunks of file system, which can be easily used by a distributed backup system.

Another important metric to consider is the amount of disk space used. Our preliminary results showed that the allocated disk space is not correlated to the chunk size, but rather to the file system used: the Ext2 and Ext3 file system use almost twice as much space as with the ReiserFS (see Figure B.4). Hence, the performance of CORRAL depends on the chosen file system, and further investigation and comparison with other file systems need to be conducted in order to decide which one is best suited.

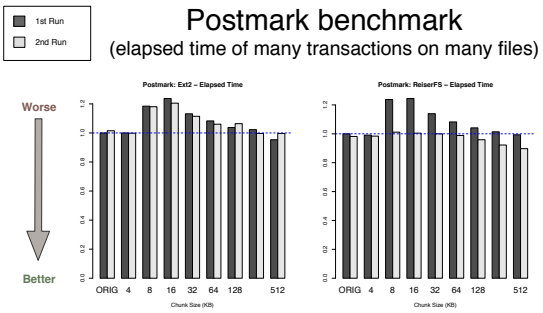


Figure B.2: Elapsed time of many transactions on many files *Postmark benchmark*. The performance is better with bigger chunk size.

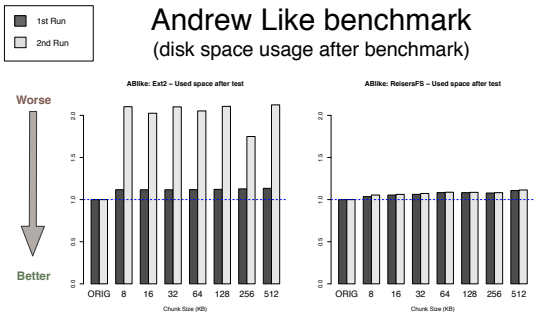


Figure B.3: Disk space usage after the first run (*Andrew like benchmark*). The Ext2 FS use different block allocations between runs, hence it consumes more space.

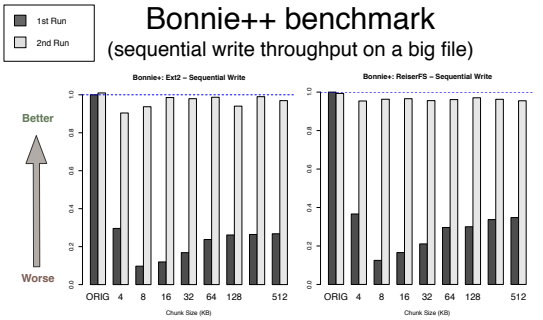


Figure B.4: Sequential write throughput on a big file. The Cow device has an overhead on the first write.

Personal Publications

[1] S. Caron, F. Giroire, D. Mazauric, J. Monteiro, and S. Pérennes. Data life time for different placement policies in p2p storage systems. In *Proceedings of the 3rd Intl. Conference on Data Management in Grid and P2P Systems (Globe)*, pages 75–88, Bilbao, Spain, Sept. 2010.

[2] S. Caron, F. Giroire, D. Mazauric, J. Monteiro, and S. Pérennes. P2P storage systems: Data life time for different placement policies. In *12es Rencontres Francophones sur les Aspects Algorithmiques de Télécommunications (Algotel)*, page 4p, Belle Dune, France, June 2010.

[3] O. Dalle, F. Giroire, J. Monteiro, and S. Pérennes. Analyse des corrélations entre pannes dans les systèmes de stockage pair-à-pair. In *11es rencontres francophones sur les Aspects Algorithmiques des Télécommunications (Algotel)*, page 4p, Carry-Le-Rouet, France, June 2009. (Best Student Paper Award).

[4] O. Dalle, F. Giroire, J. Monteiro, and S. Pérennes. Analysis of failure correlation impact on peer-to-peer storage systems. In *Proceedings of the 9th IEEE Intl. Conference on Peer-to-Peer Computing (P2P)*, pages 184–193, Seatlle, USA, Sept. 2009.

[5] A. Ferreira, A. Goldman, and J. Monteiro. Performance evaluation of dynamic networks using an evolving graph combinatorial model. In *Proceedings of the 2nd IEEE Intl. Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pages 173–180, Montreal, CA, June 2006. (Best Student Paper Award).

[6] A. Ferreira, A. Goldman, and J. Monteiro. On the evaluation of shortest journeys in dynamic networks. In *Proceedings of the 6th IEEE Intl. Symposium on Network Computing and Applications (NCA)*, pages 3–10, Cambridge, MA, USA, July 2007. Invited Paper.

[7] A. Ferreira, A. Goldman, and J. Monteiro. Using evolving graphs foremost journey to evaluate ad-hoc routing protocols. In *Proceedings of the 25th Brazilian Symposium on Computer Networks (SBRC)*, pages 17–30, Belem, Brazil, June 2007.

[8] A. Ferreira, A. Goldman, and J. Monteiro. Performance evaluation of routing protocols for manets with known connectivity patterns using evolving graphs. *Wireless Networks*, 16(3):627–640, 2010.

[9] F. Giroire, J. Monteiro, and S. Pérennes. P2P storage systems: How much locality can they tolerate? In *Proceedings of the 34th IEEE Conference on Local Computer Networks (LCN)*, pages 320–323, Zurich, Switzerland, Oct. 2009.

[10] F. Giroire, J. Monteiro, and S. Pérennes. Peer-to-peer storage systems: a practical guideline to be lazy. In *Proceedings of the IEEE Global Communications Conference (GLOBECOM)*, Miami, USA, Dec. 2010. To appear.

[11] J. Monteiro. The use of evolving graph combinatorial model in routing protocols for dynamic networks. In *Proceedings of the XV Concurso Latinoamericano de Tesis de Maestría (CLEI)*, pages 41–57, Santa Fe, Argentina, Sept. 2008. Third prize in the CLEI'08 Master's Thesis Contests.

[12] J. Monteiro and O. Dalle. CORRAL: Stackable Copy-on-Write Versioning Device using Linux Device-Mapper. In *USENIX Annual Technical Conference (USENIX'08)*, Boston, USA, June 2008. Poster.

[13] J. Monteiro and S. Pérennes. Systèmes de stockage P2P : un guide pratique. In *11es Journées Doctorales en Informatique et Réseaux (JDIR 2010)*, pages 15–20, Sophia Antipolis France, Mar. 2010.

Bibliography

[14] S. Alouf, A. Dandoush, and P. Nain. Performance analysis of peer-to-peer storage systems. *Proceedings of the 20th International Teletraffic Congress (ITC)*, LNCS 4516:642–653, June 2007.

[15] R. J. Anderson. The eternity service. In *Proceedings of Pragocrypt*, pages 242–252, 1996.

[16] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the clouds: A berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Feb 2009.

[17] C. Batten, K. Barr, A. Saraf, and S. Trepetin. pStore: A secure peer-to-peer backup system. Technical Memo MIT-LCS-TM-632, Massachusetts Institute of Technology Laboratory for Computer Science, October 2002.

[18] R. Bhagwan, D. Moore, S. Savage, and G. M. Voelker. Replication strategies for highly available peer-to-peer storage. In *Future Directions in Distributed Computing: research and position papers*, volume LNCS 2584, pages 153–158, 2003.

[19] R. Bhagwan, S. Savage, and G. Voelker. Understanding availability. In *Intl. Workshop on Peer-to-Peer Systems (IPTPS), Peer-to-Peer Systems II*, volume LNCS 2735, pages 256–267, 2003.

[20] R. Bhagwan, K. Tati, Y. chung Cheng, S. Savage, and G. M. Voelker. Total recall: System support for automated availability management. In *Proceedings of the 1st USENIX symposium on Networked Systems Design and Implementation (NSDI)*, pages 337–350, 2004.

[21] A. Binzenhöfer, T. Hoßfeld, G. Kunzmann, and K. Eger. Efficient simulation of large-scale p2p networks: Compact data structures. In *Proceedings of 15th EUROMICRO International Conference on Parallel, Distributed and Network-Based Processing*, pages 467–474, 2007.

[22] G. Bolch, S. Greiner, H. d. Meer, and K. S. Trivedi. *Queueing Networks and Markov Chains*. Wiley-Interscience, 2005.

[23] W. J. Bolosky, J. R. Douceur, D. Ely, and M. Theimer. Feasibility of a serverless distributed file system deployed on an existing set of desktop PCs. *ACM SIGMETRICS Performance Evaluation Review*, 28(1):34–43, June 2000.

[24] J.-Y. L. Boudec, D. McDonald, and J. Mundinger. A generic mean field convergence result for systems of interacting objects. In *Proceedings of the Fourth International Conference on Quantitative Evaluation of Systems (QEST'07)*, pages 3–18, Washington, DC, USA, 2007.

[25] J.-M. Busca, F. Picconi, and P. Sens. Pastis: A highly-scalable multi-user peer-to-peer file system. In *Proceedings of the 11th European Conference on Parallel Processing (Euro-par)*, volume LNCS 3648, pages 1173–1182, Lisbon, Portugal, Aug. 2005.

[26] J. W. Byers, M. Luby, M. Mitzenmacher, and A. Rege. A digital fountain approach to reliable distribution of bulk data. In *Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*, pages 56–67, New York, NY, USA, 1998. ACM.

[27] H. Casanova. Simgrid: a toolkit for the simulation of application scheduling. In *IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2001*, pages 430–437, 2001.

[28] B.-G. Chun, F. Dabek, A. Haeberlen, E. Sit, H. Weatherspoon, M. F. Kaashoek, J. Kubiatowicz, and R. Morris. Efficient replica maintenance for distributed storage systems. In *Proceedings of USENIX symposium on Networked Systems Design and Implementation (NSDI)*, pages 45–58, Berkeley, CA, USA, 2006.

[29] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, pages 46–66, July 2001.

[30] B. Cohen. Incentives build robustness in BitTorrent. In 1st Workshop on Economics of Peer-to-Peer Systems, 2003.

[31] R. B. Cooper. *Introduction to Queuing Theory*. North Holland New York, 1981.

[32] R. M. Corless, G. H. Gonnet, D. E. G. Hare, D. J. Jeffrey, and D. E. Knuth. On the lambert w function. *Advances in Computational Mathematics*, 5:329–359, 1996.

[33] L. P. Cox, C. D. Murray, and B. D. Noble. Pastiche: making backup cheap and easy. In *Proceedings of the 5th symposium on Operating systems design and implementation (OSDI 02)*, pages 285–298, New York, NY, USA, 2002. ACM.

[34] A. Crespo and H. Garcia-Molina. Modeling archival repositories for digital libraries (extended version). Technical Report 1999-24, Stanford InfoLab, 1999.

[35] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with cfs. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, pages 202–215, Canada, October 2001.

[36] A. Dandoush. *L'Analyse et l'Optimisation des Systèmes de Stockage de Données dans les Réseaux Pair-à-Pair*. PhD thesis, Université de Nice Sophia-Antipolis, Mar. 2010.

[37] A. Dandoush, S. Alouf, and P. Nain. Simulation analysis of download and recovery processes in P2P storage systems. In *Proceedings of the International Teletraffic Congress (ITC)*, Paris, France, September 2009.

[38] A. Datta and K. Aberer. Internet-scale storage systems under churn – a study of the steady-state using markov models. In *Procedings of the IEEE Intl. Conference on Peer-to-Peer Computing (P2P)*, pages 133–144, 2006.

[39] N. G. De Bruijn. A combinatorial problem. *Kibern. Sb., Nov. Ser.*, 6:33–40, 1969.

[40] R. Y. de Camargo, F. C. Filho, and F. Kon. Efficient maintenance of distributed data in highly dynamic opportunistic grids. In *Proceedings of the ACM Symposium on Applied Computing (SAC)*, pages 1067–1071, Hawaii, USA, Mar. 2009.

[41] A. Dimakis, P. Godfrey, M. Wainwright, and K. Ramchandran. Network coding for distributed storage systems. In *Proceedings of IEEE Intl. Conference on Computer Communications (INFOCOM)*, pages 2000–2008, Anchorage, AK,USA, May 2007.

[42] A. G. Dimakis, K. Ramchandran, Y. Wu, and C. Suh. A survey on network codes for distributed storage, 2010. (informal publication).

[43] J. R. Douceur and R. P. Wattenhofer. Large-scale simulation of replica placement algorithms for a serverless distributed file system. In *Proceedings of Intl. Symposium in Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, volume 0, pages 311–322, 2001.

[44] P. Druschel and A. Rowstron. PAST: a large-scale, persistent peer-to-peer storage utility. In *Proceedings of 8th Workshop on Hot Topics in Operating Systems*, pages 75–80, Schoss Elmau, Germany, May 2001.

[45] A. Duminuco. *Data redundancy and maintenance for peer-to-peer file backup systems*. PhD thesis, EURECOM - TELECOM ParisTech, Sophia Antipolis, France, 10 2009.

[46] A. Duminuco and E. Biersack. Hierarchical codes: How to make erasure codes attractive for peer-to-peer storage systems. In *Procedings of the IEEE International Conference on Peer-to-Peer Computing (P2P)*, pages 89–98, 8–11 Sept. 2008.

[47] A. Duminuco and E. Biersack. 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)*, pages 376–384, Washington, DC, USA, 2009.

[48] A. Duminuco, E. Biersack, and T. En-Najjary. Proactive replication in distributed storage systems using machine availability estimation. In *Proceedings of the ACM Intl. Conference On Emerging Networking Experiments And Technologies (CoNEXT)*. ACM New York, NY, USA, 2007.

[49] P. Flajolet and R. Sedgewick. *Analytic combinatorics*. Cambridge University Press, 2008.

[50] M. D. Flouris. Clotho: Transparent data versioning at the block i/o level. In *Proceedings of the 12th NASA Goddard, 21st IEEE Conference on Mass Storage Systems and Technologies (MSST 2004*, pages 315–328, 2004.

[51] Freepastry. http://freepastry.org/FreePastry/. Last accessed: Jan 2010.

[52] W. Galuba, K. Aberer, Z. Despotovic, and W. Kellerer. Protopeer: a p2p toolkit bridging the gap between simulation and live deployement. In *Proceedings of the 2nd International Conference on Simulation Tools and Techniques (SimuTools)*, pages 1–9, Rome, Italy, 2009.

[53] J. Gantz. The diverse and exploding digital universe: An updated forecast of worldwide information growth through 2011. Technical Report White paper, International Data Corporation (IDC), 2008.

[54] J. Gantz and D. Reinsel. The digital universe decade - are you ready? Technical Report White paper, International Data Corporation (IDC) - Sponsored by EMC Corporation, 2010.

[55] S. Ghemawat, H. Gobioff, and S.-T. Leung. The google file system. *ACM SIGOPS Operating Systems Review*, 37(5):29–43, Oct. 2003.

[56] A. V. Goldberg and P. N. Yianilos. Towards an archival intermemory. In *ADL '98: Proceedings of the Advances in Digital Libraries Conference*, page 147, Washington, DC, USA, 1998. IEEE Computer Society.

[57] Grid5000Platform. https://www.grid5000.fr/mediawiki/index.php/grid5000:home.

[58] I. Gupta, K. Birman, P. Linga, A. Demers, and R. van Renesse. Kelips: Building an efficient and stable p2p DHT through increased memory and background overhead. In *2nd Intl. Workshop on Peer-to-Peer Systems (IPTPS)*, 2003.

[59] A. Haeberlen, A. Mislove, and P. Druschel. Glacier: highly durable, decentralized storage despite massive correlated failures. In *Proceedings of USENIX symposium on Networked Systems Design and Implementation (NSDI)*, pages 143–158, Berkeley, USA, 2005.

[60] R. W. Hamming. Error detecting and error correcting codes. *Bell System Technical Journal*, 26(2):147–160, 1950.

[61] R. Hasan, Z. Anwar, W. Yurcik, L. Brumbaugh, and R. Campbell. A survey of peer-to-peer storage techniques for distributed file systems. In *Proceedings of the Intl. Conference on Information Technology: Coding and Computing (ITCC'05) - Volume II*, pages 205–213, Washington, DC, USA, 2005.

[62] H. Hennion. Limit theorems for products of positive random matrices. *The Annals of Probability*, 25(4):1545–1587, 1997.

[63] D. Hitz, J. Lau, and M. Malcolm. File system design for an nfs file server appliance. In *Proceedings of the USENIX Winter 1994 Technical Conference*, pages 235–246, San Fransisco, CA, USA, January 1994.

[64] I. Ivkovic. Improving gnutella protocol: Protocol analysis and research proposals. Technical report, University of Waterloo, 2001.

[65] F. M. Kaashoek and D. R. Karger. Koorde: A simple degree-optimal distributed hash table. In *2nd Intl. Workshop on Peer-to-Peer Systems (IPTPS)*, pages 98–107, 2003.

[66] M. Karlsson, M. Mahalingam, M. Karlsson, and M. Mahalingam. Do we need replica placement algorithms in content delivery networks. In *7th International Workshop on Web Content Caching and Distribution (WCW)*, August 2002.

[67] T. Klingberg and R. Manfredi. Gnutella protocol 0.6, web document. `http://rfc-gnutella.sourceforge.net/src/rfc-0_6-draft.html`, 2002.

[68] S. Ktari, M. Zoubert, A. Hecker, and H. Labiod. Performance evaluation of replication strategies in dhts under churn. In *MUM '07*, pages 90–97, New York, USA, 2007. ACM.

[69] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, C. Wells, et al. OceanStore: an architecture for global-scale persistent storage. *ACM SIGARCH Computer Architecture News*, 28(5):190–201, 2000.

[70] G. Kunzmann, R. Nagel, T. Hossfeld, A. Binzenhofer, and K. Eger. Efficient simulation of large-scale p2p networks: Modeling network transmission times. In *Proceedings of 15th EUROMICRO International Conference on Parallel, Distributed and Network-Based Processing*, pages 475–481, Los Alamitos, CA, USA, Feb 2007. IEEE Computer Society.

[71] T. Kurtz. *Approximation of Population Processes*. Society for Industrial Mathematics, 1981.

[72] S. Legtchenko, S. Monnet, P. Sens, and G. Muller. Churn-resilient replication strategy for peer-to-peer distributed hash-tables. In *Proceedings of the 11th International Symposium on Stabilization, Safety, and Security of Distributed Systems*, volume LNCS 5873, pages 485–499, Lyon, France, 2009.

[73] Q. Lian, W. Chen, and Z. Zhang. On the impact of replica placement to the reliability of distributed brick storage systems. In *International Conference on Distributed Computing Systems (ICSCS'05)*, pages 187–196, Los Alamitos, CA, USA, 2005. IEEE Computer Society.

[74] J. Liang, R. Kumar, and K. Ross. Understanding kazaa. http://citeseer.ist.psu.edu/liang04understanding.html, 2004.

[75] D. Liben-Nowell, H. Balakrishnan, and D. Karger. Analysis of the evolution of peer-to-peer systems. In *Proceedings of the 21st Annual Symposium on Principles of Distributed Computing (PODC'02)*, pages 233–242, 2002.

[76] W. Lin, D. Chiu, and Y. Lee. Erasure code replication revisited. In *Procedings of the IEEE International Conference on Peer-to-Peer Computing (P2P)*, pages 90–97, 2004.

[77] P. G. López, C. Pairot, R. Mondéjar, J. P. Ahulló, H. Tejedor, and R. Rallo. PlanetSim: A new overlay network simulation framework. In T. Gschwind and C. Mascolo, editors, *SEM*, volume 3437 of *Lecture Notes in Computer Science*, pages 123–136. Springer, 2004.

[78] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim. A survey and comparison of peer-to-peer overlay network schemes. *IEEE Communications Survey and Tutorial*, 7(2):72–93, Sept. 2005.

[79] M. Luby, M. Mitzenmacher, M. Shokrollahi, D. Spielman, and V. Stemann. Practical loss-resilient codes. In *Proceedings of the 29th annual ACM symposium on Theory of computing*, pages 150–159. ACM New York, NY, USA, 1997.

[80] A. Markov. On the problem of representability of matrices. *Z. Math. Logik Grundlagen Math*, 4:157–168, 1958.

[81] P. Maymounkov and D. Mazières. Kademlia: A peer-to-peer information system based on the xor metric. In *Revised papers from the 1st Intl. Workshop on Peer-to-Peer Systems (IPTPS)*, pages 53–65, London, UK, 2002.

[82] C. B. Morrey III and D. Grunwald. Peabody: The time travelling disk. In *20th IEEE/11 th NASA Goddard Conference on Mass Storage Systems and Technologies (MSS'03)*, page 241, Washington, DC, USA, 2003.

[83] A. Muthitacharoen, R. Morris, T. M. Gil, and B. Chen. Ivy: A read/write peer-to-peer file system. In *Proceedings of 5th Symposium on Operating Systems Design and Implementation (OSDI 2002)*, pages 31–44, 2002.

[84] S. Naicken, A. Basu, B. Livingston, and S. Rodhetbhai. A survey of peer-to-peer network simulators. In *Proceedings of the 7th Annual Postgraduate Symposium*, Liverpool, UK, 2006.

[85] The network simulator – NS2. http://nsnam.isi.edu/nsnam/. Last accessed: Jan 2010.

[86] Overlay weaver - an overlay construction toolkit. http://overlayweaver.sourceforge.net/. Last accessed: Jan 2010.

[87] The oversim simulation framework. http://www.oversim.org/. Last accessed: Jan 2010.

[88] D. A. Patterson, G. Gibson, and R. H. Katz. A case for redundant arrays of inexpensive disks (raid). In *Proceedings of ACM International Conference on Management of Data (SIGMOD'88)*, pages 109–116, New York, NY, USA, 1988.

[89] Peersim: A peer-to-peer simulator. http://peersim.sourceforge.net/. Last accessed: Jan 2010.

[90] F. Picconi, B. Baynat, and P. Sens. Predicting durability in dhts using markov chains. In *Proceedings of the 2nd Intl. Conference on Digital Information Management (ICDIM)*, volume 2, pages 532–538, Oct. 2007.

[91] E. Pinheiro, W. Weber, and L. Barroso. Failure trends in a large disk drive population. In *Proceedings of USENIX Conference on File and Storage Technologies (FAST)*, 2007.

[92] Planetlab. http://www.planet-lab.org/. Last accessed: Jan 2010.

[93] J. S. Plank. A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems. *Software – Practice & Experience*, 27(9):995–1012, September 1997.

[94] J. S. Plank, J. Luo, C. D. Schuman, L. Xu, and Z. Wilcox-O'Hearn. A performance evaluation and examination of open-source erasure coding libraries for storage. In *Proceedings of the 7th USENIX Conference on File and Storage Technologies (FAST)*, February 2009.

[95] C. G. Plaxton, R. Rajaraman, and A. W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *Proceedings of the 9th annual ACM symposium on Parallel algorithms and architectures (SPAA'97)*, pages 311–320, New York, NY, USA, 1997. ACM.

[96] S. Quinlan, J. McKie, and R. Cox. Fossil, an archival file-server. `http://www.cs.bell-labs.com/sys/doc/fossil.pdf`.

[97] M. O. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *Journal of ACM*, 36(2):335–348, 1989.

[98] S. Ramabhadran and J. Pasquale. Analysis of long-running replicated systems. In *Proceedings of IEEE International Conference on Computer Communications (INFOCOM)*, pages 1–9, Barcelona, Catalunya, Spain, Apr. 2006.

[99] S. Ramabhadran and J. Pasquale. A resource allocation problem in replicated peer-to-peer storage systems. In *Proceedings of IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*, pages 1–8, March 2007.

[100] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*, 2001.

[101] I. S. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, 1960.

[102] S. Rhea, P. Eaton, D. Geels, H. Weatherspoon, B. Zhao, and J. Kubiatowicz. Pond: The oceanstore prototype. In *Proceedings of USENIX Conference on File and Storage Technologies (FAST)*, pages 1–14, Berkeley, CA, USA, 2003.

[103] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling churn in a dht. In *Proceedings of the USENIX Annual Technical Conference*, pages 10–10, Berkeley, CA, USA, 2004.

[104] G. Riley and M. Ammar. Simulating large networks: How big is big enough? In *Proceedings of First International Conference on Grand Challenges for Modeling and Simulation*, Jan. 2002.

[105] R. Rodrigues and B. Liskov. High availability in dhts: Erasure coding vs. replication. In *Workshop on Peer-to-Peer Systems (IPTPS), Peer-to-Peer Systems IV*, pages 226–239. LNCS, 2005.

[106] S. M. Ross. *Introduction to Probability Models, Ninth Edition*. Academic Press, Inc., Orlando, FL, USA, 2006.

[107] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, volume 2218, pages 329–350, November 2001.

[108] A. Rowstron and P. Druschel. Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, pages 188–201, New York, NY, USA, 2001.

[109] B. Schroeder and G. Gibson. Disk failures in the real world: What does an mttf of 1,000,000 hours mean to you? In *Proceedings of the conference on File and Storage Technologies (FAST)*, 2007.

[110] E. Sit, A. Haeberlen, F. Dabek, B. gon Chun, H. Weatherspoon, R. Morris, M. F. Kaashoek, and J. Kubiatowicz. Proactive replication for data durability. In *5th Intl. Workshop on Peer-to-Peer Systems (IPTPS)*, 2006.

[111] A. B. Stirling and M. K. Stirling. Tools for peer-to-peer network simulation. Internet Draft IRTF P2PRG, July 2006.

[112] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *IEEE ACM Transactions on Networking*, 11(1):17–31, 2003.

[113] D. Stutzbach and R. Rejaie. Understanding churn in peer-to-peer networks. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement (IMC'06)*, pages 189–202, New York, NY, USA, 2006. ACM.

[114] R. D. C. Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2010. ISBN 3-900051-07-0.

[115] S. A. Theotokis and D. Spinellisa. A survey of peer-to-peer content distribution technologies. *ACM Computing Surveys (CSUR)*, 36(4):335–371, December 2004.

[116] H. C. Tijms. *Stochastic modelling and analysis: a computational approach*. John Wiley & Sons, Inc., New York, NY, USA, 1986.

[117] N. S. Ting. Simulating peer-to-peer networks. Master's thesis, University of Saskatchewan Library, Aug 2006.

[118] A. Traeger, E. Zadok, N. Joukov, and C. P. Wright. A nine year study of file system and storage benchmarking. *Transaction on Storage (TOS)*, 4(2):1–56, 2008.

[119] Ubistorage. http://www.ubistorage.com/. Last accessed: Jul. 2010.

[120] G. Utard and A. Vernois. Data durability in peer to peer storage systems. In *Proceedings of the IEEE International Symposium on Cluster Computing and the Grid (CCGRID)*, pages 90–97, USA, 2004.

[121] R. van Renesse. Efficient reliable internet storage. In *Workshop on Dependable Distributed Data Management*, Florianopolis, Brazil, October 2004.

[122] R. van Renesse and F. B. Schneider. Chain replication for supporting high throughput and availability. In *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation (OSDI)*, pages 7–7, Berkeley, CA, USA, 2004.

[123] A. Varga. OMNet++: Discrete event simulation systems. http://www.omnetpp.org/. Last accessed: Jan 2010.

[124] H. Weatherspoon. *Design and evaluation of distributed wide-area on-line archival storage systems*. PhD thesis, University of California at Berkeley, Berkeley, CA, USA, 2006. Adviser-Kubiatowicz,, John.

[125] H. Weatherspoon and J. Kubiatowicz. Erasure coding vs. replication: A quantitative comparison. In *Revised papers from the 1st Intl. Workshop on Peer-to-Peer Systems (IPTPS)*, volume LNCS 2429, pages 328–337, Cambridge, MA, USA, 2002.

[126] Z. Wilcox-O'Hearn. Zfec 1.4.7 - open source erasure code package. http://pypi.python.org/pypi/zfec, 2010.

[127] Z. Wilcox-O'Hearn and B. Warner. Tahoe: the least-authority filesystem. In Y. Kim and W. Yurcik, editors, *ACM StorageSS*, pages 21–26. ACM, 2008.

[128] D. Wu, Y. Tian, K.-W. Ng, and A. Datta. Stochastic analysis of the interplay between object maintenance and churn. *Computer Communications*, 31(2):220–239, 2008.

[129] Wuala - secure online storage. http://www.wuala.com/. Last accessed: Jul. 2010.

[130] Z. Zhang, Q. Lian, S. Lin, W. Chen, Y. Chen, and C. Jin. Bitvault: a highly reliable distributed data retention platform. *ACM SIGOPS Operating Systems Review*, 41(2):27–36, 2007.

[131] Z. Zhang, S. Lin, Q. Lian, and C. Jin. Repstore: A self-managing and self-tuning storage backend with smart bricks. *International Conference on Autonomic Computing (ICAC)*, 0:122–129, 2004.

[132] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 22(1):41–53, 2004.

[133] Y. C. Zheng Zhang, Qiao Lian. Xring: Achieving high-performance routing adaptively in structured p2p. Tech. Report MSR-TR-2004-93, Microsoft Research, 2004.

Bibliography - Routing in Mobile Networks

[134] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer Networks (Elsevier) Journal*, 38(4):393–422, Mar 2002.

[135] S. Bhadra and A. Ferreira. Complexity of connected components in evolving graphs and the computation of multicast trees in dynamic networks. In *Proceedings of Adhoc-Now'03*, volume 2865 of *Lecture Notes in Computer Science*, pages 259–270. Springer Verlag, Oct 2003. Also appeared as an INRIA research report (RR-4531) in Ago. 2002.

[136] BonnMotion: A mobility scenario generation and analysis tool. http://web.informatik.uni-bonn.de/IV/Mitarbeiter/dewaal/BonnMotion/, Page accessed on Oct 2007.

[137] A. Boukerche. Performance evaluation of routing protocols for ad hoc wireless networks. *ACM Mobile Network Applications (MONET)*, 9(4):333–342, 2004.

[138] J. Broch, D. A. Maltz, D. B. Johnson, Y.-C. Hu, and J. Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *Proceedings of the 4th ACM annual international Conference on Mobile Computing and Networking (MobiCom'98)*, pages 85–97, Dallas, TX, USA, 1998. ACM Press.

[139] B. Bui-Xuan, A. Ferreira, and A. Jarry. Computing shortest, fastest, and foremost journeys in dynamic networks. *International Journal of Foundations of Computer Science*, 14(2):267–285, Apr 2003. Also appeared as an INRIA research report (RR-4589) in Oct. 2002.

[140] C. Carter, S. Yi, and R. Kravets. ARP considered harmful: Manycast transactions in ad hoc networks. In *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC 03)*, New Orleans, LA, Mar 2003.

[141] C. Chen and E. Ekici. A routing protocol for hierarchical leo/meo satellite ip networks. *ACM Wireless Networks (WiNet)*, 11(4):507–521, 2005.

[142] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. The MIT press, Cambridge, MA, USA, 1990.

[143] S. Corson and J. Macker. Mobile ad hoc networking (MANET): Routing protocol performance issues and evaluation considerations. RFC 2501, IETF, January 1999.

[144] DTNRG. Delay tolerant networking research group: http://www.dtnrg.org/. (page accessed on aug. 2007).

[145] A. Farago and V. R. Syrotiuk. MERIT: a scalable approach for protocol assessment. *ACM Mobile Networks and Applications (MONET) journal*, 8(5):567–577, 2003.

[146] A. Ferreira. Building a reference combinatorial model for MANETs. *IEEE Network*, 18(5):24–29, Set 2004. A preliminary version appeared as "On models and algorithms for dynamic communication networks: The case for evolving graphs", Algotel 2002, Mèze, France, May 2002.

[147] A. Ferreira, J. Galtier, and P. Penna. Topological design, routing and hand-over in satellite networks. In I. Stojmenovic, editor, *Handbook of Wireless Networks and Mobile Computing*, pages 473–493. John Wiley and Sons, New York, NY, USA, 2002.

[148] IEEE standard for wireless LAN medium access control (MAC) and physical layer (PHY) specifications. http://grouper.ieee.org/groups/802/11/main.html, Page accessed on Mar 2007.

[149] P. Jacquet, P. Muhlethaler, T. Clausen, A. Laouiti, A. Qayyum, and L. Viennot. Optimized link state routing protocol. In *Proceedings of 5th IEEE INMIC'01*, pages 62–68, Lahore, Pakistan, Dec 2001.

[150] S. Jain, K. Fall, and R. Patra. Routing in a delay tolerant network. In *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 145–158, New York, NY, USA, 2004. ACM Press.

[151] D. B. Johnson and D. A. Maltz. Dynamic source routing in ad hoc wireless networks. In Imielinski and Korth, editors, *Mobile Computing*, volume 353, chapter 5, pages 153–181. Kluwer Academic Publishers, 1996.

[152] B. Krishnamachari. *Networking Wireless Sensors*. Cambridge University Press, New York, NY, USA, 2005.

[153] B. S. Manoj, K. J. Kumar, C. Frank, and C. S. R. Murthy. On the use of multiple hops in next generation wireless systems. *ACM Wireless Networks (WiNet)*, 12(2):199–221, 2006.

[154] S. Merugu, M. Ammar, and E. Zegura. Routing in space and time in networks with predictable mobility. Technical report, Technical Report GIT-CC-04-7, Georgia Institute of Technology, 2004.

[155] NS2. The network simulator – ns2. http://nsnam.isi.edu/nsnam/, Page accessed on Mar 2007.

[156] C. Perkins and P. Bhagwat. Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. In *Conference on Communications Architectures, Protocols and Applications (ACM SIGCOMM'94)*, pages 234–244, Sep 1994.

[157] C. E. Perkins and E. M. Royer. Ad-hoc on-demand distance vector routing. In *Proceedings of 2nd IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'99)*, pages 90–100, Feb 1999.

[158] C. E. Perkins, E. M. Royer, S. R. Das, and M. K. Marina. Performance comparison of two on-demand routing protocols for ad hoc networks. In *IEEE Personal Communications*, volume 8, pages 16–28. IEEE Communications Society, feb 2001.

[159] Rice University Monarch Project. The CMU monarch wireless and mobility extensions to NS2. http://www.monarch.cs.rice.edu/, Page accessed on Mar 2007.

[160] F. J. Ros. UM-OLSR implementation (version 0.8.8) for NS2. http://masimum.dif.um.es/?Software:UM-OLSR, Page accessed on Mar 2007.

[161] R. Sen, R. Handorean, G.-C. Roman, and G. Hackmann. Knowledge-driven interactions with services across ad hoc networks. In *ICSOC '04: Proceedings of the 2nd international conference on Service oriented computing*, pages 222–231, New York, NY, USA, 2004. ACM Press.

[162] G. Sharma, R. R. Mazumdar, and N. B. Shroff. On the complexity of scheduling in wireless networks. In *Proceedings of the 12th ACM annual international Conference on Mobile Computing and Networking (MobiCom'06)*, pages 227–238, Sep 2006.

[163] I. G. Siqueira, L. B. Ruiz, A. A. F. Loureiro, Jose, and M. Nogueira. Coverage area management for wireless sensor networks. *International Journal of Network Management*, 17(1):17–31, 2007.

[164] T. Spyropoulos, K. Psounis, and C. Raghavendra. Efficient routing in intermittently connected mobile networks: The single-copy case. *to appear in ACM/IEEE Transactions on Networking.*, 2007.

[165] I. Stojmenovic, A. Nayak, and J. Kuruvila. Design guidelines for routing protocols in ad hoc and sensor networks with a realistic physical layer. *IEEE Communications Magazine (Ad Hoc and Sensor Networks Series)*, 43(3):101–106, March 2005.

[166] I. Stojmenovic(ed.). *Handbook of Wireless Networks and Mobile Computing*. John Wiley and Sons, Feb 2002.

[167] I. Stojmenovic(ed.). *Handbook of Sensor Networks: Algorithms and Architectures*. John Wiley and Sons, Oct 2005.

[168] J. Wu. *Handbook On Theoretical And Algorithmic Aspects Of Sensor, Ad Hoc Wireless, and Peer-to-Peer Networks*. Auerbach Publications, Boston, MA, USA, 2005.

[169] J. Yoon, M. Liu, and B. Noble. Random waypoint considered harmful. In *The 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'03)*, volume 2, pages 1312–1321, 2003.

[170] Z. Zhang. Routing in intermittently connected mobile ad hoc networks and delay tolerant networks: Overview and challenges. *IEEE Communications Surveys*, 8(1):24–37, 1st Quarter 2006.