Faculté des Sciences
Institut d'informatique

Erasure coding for distributed storage systems

Thèse
présentée à la Faculté des Sciences
pour l'obtention du grade de Docteur ès Sciences
par

Roberta Barbi

Acceptée sur proposition du jury :
Dr. Hugues Mercier, Directeur de thèse, Université de Neuchâtel, Suisse
Prof. Elisa Gorla, Co-Directrice de thèse, Université de Neuchâtel, Suisse
Dr. Alessio Caminata, Université de Neuchâtel, Suisse
Prof. Anna-Lena Horlemann, Universität St. Gallen, Suisse
Prof. Peter Kropf, Université de Neuchâtel, Suisse

Soutenue le 20 Février 2019

UNIVERSITÉ DE NEUCHÂTEL

Faculté des Sciences
Secrétariat-décanat de Faculté
Rue Emile-Argand 11
2000 Neuchâtel – Suisse
Tél : + 41 (0)32 718 21 00
E-mail : secretariat.sciences@unine.ch

IMPRIMATUR POUR THESE DE DOCTORAT

La Faculté des sciences de l'Université de Neuchâtel autorise l'impression de la présente thèse soutenue par

Madame Roberta BARBI

Titre:

"Erasure coding for distributed storage systems"

sur le rapport des membres du jury composé comme suit:

- Dr Hugues Mercier, directeur de thèse, Université de Neuchâtel, Suisse
- Prof. Peter Kropf, Université de Neuchâtel, Suisse
- Prof. Elisa Gorla, Université de Neuchâtel, Suisse
- Dr Alessio Caminata, Université de Neuchâtel, Suisse
- Prof. Anna-Lena Horlemann, Université de Saint-Gall, Suisse

Neuchâtel, le 28 février 2019

Le Doyen, Prof. P. Felber

Acknowledgments

I would like to thank my advisor Dr. Hugues Mercier and my co-advisor Prof. Elisa Gorla for the opportunity to join their team and encouragement during these three years. Besides my advisors, I would like to thank the rest of my thesis committee: Dr. Alessio Camminata, Prof. Anna-Lena Horlemann and Prof. Peter Kropf for their feedback on my work, insightful questions and comments.

I would like to thank all the people from the Computer Science department for the time spent together, as well as all the people I worked with, I am thinking about papers coauthors, SafeCloud project partners as well as people from other departments and institutions. The collaboration with them has been a great learning experience.

I am very grateful to Prof. Pascal Felber for involving me in his group.

My most special thanks goes to Dr. Valerio Schiavoni for supporting me along the way and for always being there.

Last but not the least, I thank my friends Riccardo, Serena, Luca e Irene, and mum and dad that always believe in me, even in the dark days.

Abstract

A huge quantity of data is generated and stored every day. Preeminent examples are the 300 million photos that get uploaded daily to Facebook or the 156 million emails sent every minute. Furthermore, it is remarkable that almost 90% of the data in the world has been generated in the last 2 years. This humongous quantity of data is typically hosted in data centres located in different geographic areas, and must be promptly served upon request. Hence, a paramount property for distributed storage systems is reliability, which can be guaranteed in case of failed or unavailable hardware or lack of network connectivity at the price of introducing redundancy in the system. To this aim, erasure coding techniques optimizing the storage overhead/fault tolerance tradeoff have been used in production systems for many years, even though they do not completely suit distributed environments.

Only in the last decade, a number of advanced coding techniques tailored to distributed storage systems have been developed. As an example, Microsoft Azure deployed a local reconstruction code in 2012. System designers now consider maintenance-oriented metrics other than fault tolerance and storage overhead, specifically repair locality, repair bandwidth and disk I/O. As it is not possible to optimize all of them at once, the optimal code depends on the particular application.

This quest of good codes for distributed storage systems has led to a very active research area in which this thesis can be framed. First, we examine the different efficiency metrics as well as state-of-the-art codes addressing them. We then focus on the study of how the repair locality, that is the number of nodes involved in repair operations, affects the use of distributed storage systems. Next, we investigate the problem of block placement in distributed environments and its effect on systems valuable properties such as the fetch latency. Finally, we study how to introduce data entanglement on top of well-known coding schemes to enhance desirable features of the resulting system, i.e., anti-censorship and reliability.

Keywords: erasure coding, distributed storage system, fault tolerance, storage overhead, repair locality, anti-censorship, reliability

Résumé

Une énorme quantité de données est désormais générée et stockée : 300 millions de photos sont téléchargées sur Facebook chaque jour et 156 courriels sont envoyés chaque minute. Par ailleurs, près de 90% des données dans le monde ont été produites au cours de deux dernières années. Cette quantité colossale de données est typiquement hébergée dans des centres de données situés en régions géographiques différentes, et doit être servie promptement sur demande. Raison pour laquelle une propriété fondamentale pour les systèmes de stockage répartis est la fiabilité. Une fiabilité qui ne peut être garantie en cas de panne réseau qu'en ajoutant de la redondance. Pour atteindre cet objectif, des techniques telles que les codes d'effacement qui optimisent le compromis entre le coût de stockage et la tolérance aux pannes ont été utilisées dans les systèmes en production depuis de nombreuses années, bien qu'ils ne soient pas complètement adaptés aux environnements répartis.

Seulement au cours de la dernière décennie, une multitude de techniques avancées adaptées aux systèmes répartis ont été développées. À titre d'exemple, Microsoft Azure déploie depuis 2012 un code avec des propriétés de reconstruction locale. Les architectes systèmes prennent désormais en compte des paramètres portant sur les coûts de maintenance en plus du coût de stockage et de la tolérance aux pannes, tels que la localité ou les coûts en bande passante et en lecture/écriture. Puisqu'il est impossible d'optimiser tout cela en même temps, le code optimal dépend de l'application concernée.

Cette thèse s'inscrit dans ce domaine foisonnant de la recherche de codes optimaux dans les environnement répartis. D'abord, nous examinons différentes métriques ainsi que les techniques de codage de pointe qui satisfont à ces métriques. Ensuite, nous étudions comment la localité, c'est-à-dire le nombre de nœuds impliqués dans les opérations de reconstruction, influe sur l'utilisation de ressources. Par la suite, nous nous penchons sur le problème de la distribution des données et ses effets sur des propriétés critiques telles que la latence. Enfin, nous examinons comment l'enchevêtrement des données au-dessus d'un code d'effacement peut renforcer des propriétés du système comme la résistance à la censure et la fiabilité.

Mots-clés: code d'effacement, système de stockage réparti, tolérance aux pannes, coût de stockage, localité, résistance à la censure, fiabilité

Contents

Notation

A^t	Transpose of matrix A
$\mathscr{C}(k, n-k)$	Linear code of length n and dimension k
d	Minimum distance of a linear code
$\mathbb{F}_p$	Finite field with p elements
G	Generator matrix of a linear code
H	Parity-check matrix of a linear code
HAM	Hamming code of length 7 and dimension 4
I_j	Identity matrix of size $j \times j$
k	Dimension of a linear code
l	Left degree of a left-regular LDPC code
MDS$(k,n-k)$	Maximum Distance Separable code of length n and dimension k
n	Length of a linear code
r	Right degree of a right-regular LDPC code
r	Repair locality of a linear code
$r(i)$	Fraction of repairable failures of size i
RS$(k,n-k)$	Reed-Solomon code of length n and dimension k
$\lambda(\cdot)$	Variable node degree distribution of a LDPC code
$\rho(\cdot)$	Check node degree distribution of a LDPC code
σ	Stopping number of a LDPC code
$\omega(v)$	Hamming weight of the vector v

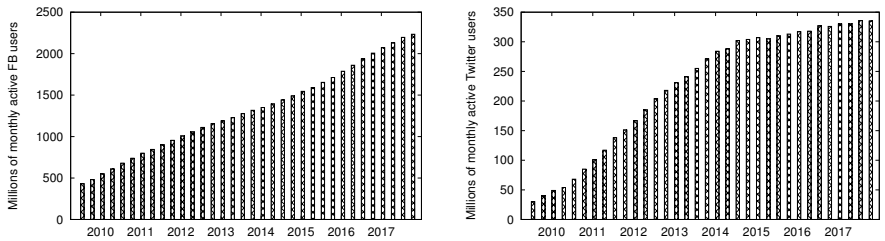
Chapter 1

Introduction

1.1 Context

The amount of data generated in the world grows exponentially. The 90% of all the data stored in 2017 was created starting from 2015 [1] and it is estimated that by 2020 every person on the earth will produce 1.7MB of data per second [2]. Indeed people create data by actively storing information, but also by performing actions, such as Google searches, Facebook likes and sharing content. For example, I have downloaded the data that Google stores about me at `google.com/takeout` (more than 7GB of data). Besides the pictures that I upload on GooglePhotos and the documents saved in GoogleDrive, Google collects information about places where I have been, things that I have searched (and deleted), apps that I use, my YouTube history and also my advertisement profile [3].

More data to be stored comes from Twitter and Facebook. The increasing trend of their users in the last years is shown in Figure 1.1. Nowadays, about 335M users are active in posting 280-characters-bounded messages, for a total of more than 500M tweets a day [4]. On Facebook the number of active users is about 2.23 billion. In the beginning of 2012, when that number was around 1 billion, every 60 seconds there were: 510K posted comments, 293K status updates, and 136K uploaded photos [5]. In August 2012, every day 300M photos were uploaded and 500 TB of data stored [6].



(a) Facebook [7] (b) Twitter [8]

Figure 1.1: Millions of worldwide monthly active users from the first quarter of 2010 to the second quarter of 2018.

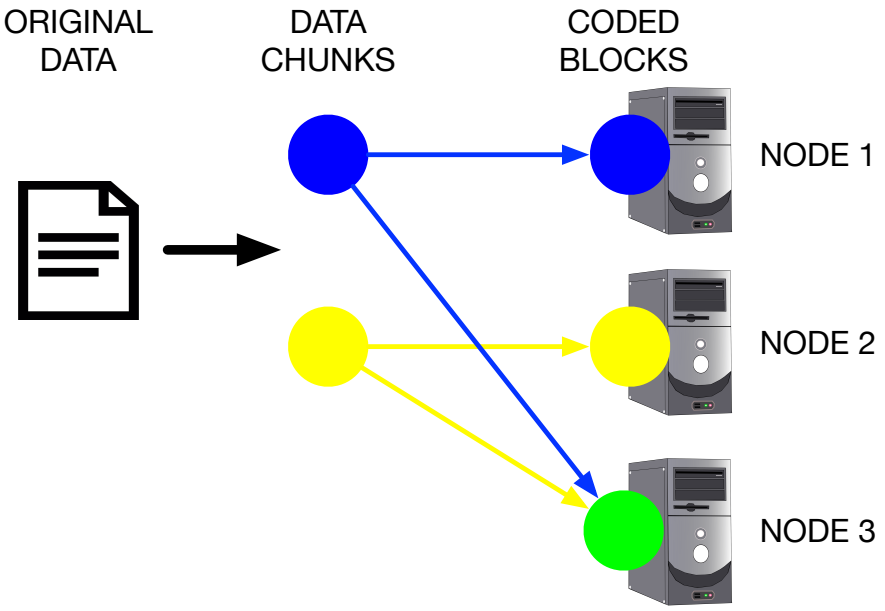


Figure 1.2: Encoding. Data to be stored is split in two data blocks (blue and yellow) which are stored into two different nodes. A third coded block (the green one), obtained combining the first two, is stored in a separate storage node.

This huge amount of data that users outsource to the various platforms must be protected against loss. The objective of this thesis is the design and analysis of resource-efficient storage solutions to reliably store the data that users entrust to large scale distributed storage systems.

1.2 Motivation and objective

A time-honoured mean for achieving reliable storage is the introduction of redundancy. Data redundancy techniques range from storage-intensive data *replication* to *erasure coding*, methods for data protection that meet the need for reduced storage requirements.

The process of adding redundancy to a data object to be stored is called *encoding*, and illustrated in Figure 1.2. Generally, data to be stored is split into chunks, which are combined with each other to produce parities, i.e., the redundancy. In the example, we create one green redundant block combining the yellow and blue data blocks, thus the scheme entails a 50% storage overhead. The original data blocks plus the parities form the coded blocks, which are typically stored in distinct storage nodes, to decrease the impact of node failures on the availability of the data object. The particular way to combine the data blocks into parities determines the amount of storage nodes that can fail simultaneously without causing data loss.

The process of retrieving the data object from the corresponding coded blocks is called *decoding*, and illustrated in Figure 1.3. When the data blocks are stored into functioning storage nodes, we just need to fetch them as they consti-

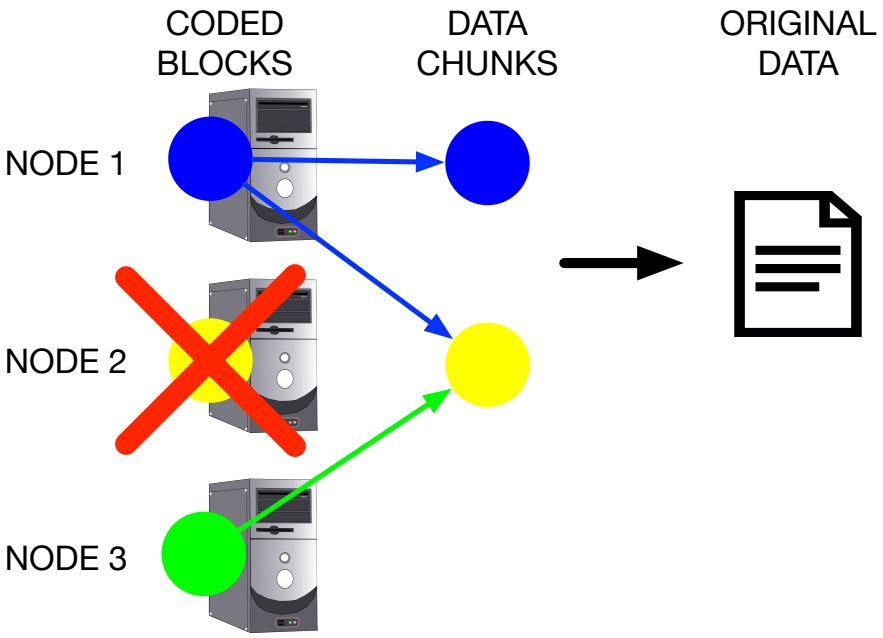


Figure 1.3: Decoding. To retrieve the data from the system two alive storage nodes are sufficient. The lost (yellow) block is recomputed from (blue and green) alive ones. The (yellow and blue) data blocks constitute the original data.

tute the original data object. In our example, this requires Node 1 and Node 2 to be functioning. On the other hand, we can use redundancy to restore lost data blocks by reversing the encoding operation. In particular, in the example, we can recover from any single failed node, but if two nodes fail simultaneously, we lose the document. In Figure 1.3, first, we fetch the blue data block. Then, we recover from the loss of the yellow data block by combining the alive blue data block with the green parity block.

When a node fails, the storage system can identify the particular lost blocks via its metadata. Metadata describes and summarizes basic information about what is stored in the system, so that finding and working with particular instances is easier. For example, in HDFS [9], a distributed file system having master/slave architecture, the entire file system namespace, including the mapping of blocks to files and the file system properties, is placed in a file called the FsImage. The FsImage is stored in the master's local file system and itself protected by maintaining multiple copies [10]. The metadata generated for a HDFS file and a HDFS block is around 150 bytes. Thus, metadata replication is reasonable as long as the quantity of small files (files whose size is significantly smaller than the HDFS default block size of 64MB) is limited [11].

Storage systems must be able to verify the freshness of the data stored. For example, as it is possible that a block fetched from a HDFS slave arrives corrupted (because of faults in a storage device, network faults, or buggy software), the HDFS client implements checksums on the contents of HDFS files [10]. When a client creates a HDFS file, it computes a checksum of each block of the file and stores these checksums in a separate file in the same HDFS namespace.

When a client retrieves file contents, it verifies that the data received from each slave matches the checksum stored in the associated checksum file.

Thanks to the metadata and to the integrity checks, the value of a block in a distributed storage system is either known or unknown, that is, it is not possible that the value is wrong. This type of failure is called an *erasure* and can be modelled by a channel that either transmits the symbol as it is or declares an erasure. For example, the binary erasure channel (BEC) has input alphabet $\mathcal{X} = \{0, 1\}$ and output alphabet $\mathcal{Y} = \{0, 1, e\}$ and is parametrized by the erasure probability $0 \leq \alpha < 1$ so that upon input $x \in \mathcal{X}$, the BEC_α outputs

$$\mathrm{BEC}_\alpha(x) = \begin{cases} x & \text{with probability } 1 - \alpha \\ e & \text{with probability } \alpha \end{cases}. \tag{1.1}$$

The capacity of the BEC is $1 - \alpha$ [12], and this provides a condition on the maximal rate at which reliable communication is possible. In turn, capacity achieving codes are desirable as high code rates lead to low storage overhead.

The storage overhead is not the only parameter affecting the overall performance of the erasure coding technique deployed in a storage system. When nodes fail, the system tries to reconstruct lost blocks by exploiting the erasure code. This process can involve different entities depending on the system architecture, for example a proxy in charge of coding operations and a replacement node taking over the failed one. Regardless of the specific architecture, a number of nodes, holding fresh shares from the original data, must be contacted and must, in turn, read and send through the network the blocks necessary for reconstruction. The number of nodes contacted (*repair locality*), the amount of data read (*I/O*) as well as the amount of bits sent thought the network (*repair bandwidth*) should be minimized to achieve resource-efficient storage solutions.

1.3 Thesis summary

The rest of the thesis is organized as follows. In Chapter 2, we outline foundations of coding theory. In Chapter 3, we introduce and discuss the Markov model for reliability evaluation. We review the state-of-the-art of erasure coding techniques for distributed storage systems in Chapter 4, before presenting our contributions (in Chapters 5, 6, 7, 8), which we summarize as follows.

1.3.1 Locality in distributed storage systems

Distributed storage systems often use erasure coding techniques to provide reliability while decreasing the storage overhead required by replication. Due to the drawbacks of standard MDS erasure correcting codes with respect to maintenance-oriented metrics, numerous coding schemes recently proposed for distributed storage systems target metrics such as repair locality and repair bandwidth. Unfortunately, these schemes are not always practical, and for most of them locality covers information data only. In Chapter 5 and [Pub2], we compare three explicit linear codes for three types of locality: a Reed-Solomon code for *worst-case locality*, a recently proposed pyramid code for *information locality* and the Hamming code Ham, an optimal locally repairable code directly built from its generator matrix for *all-blocks locality*. We also provide an efficient

way for repairing HAM and show that for the same level of storage overhead HAM provides faster encoding, faster repair and lower repair bandwidth than the other two solutions while requiring less than fifty lines of code.

1.3.2 Block placement in distributed storage systems

Regardless of the specific coding techniques, which can be the HAM code presented in Chapter 5 as well as traditional Reed-Solomon codes, codeword blocks must be spread across the system nodes to guarantee reliability without exacerbating the overall latency. So we move to the evaluation of block placement strategies using a distributed tuple space as a practical usecase.

The tuple space abstraction provides an easy-to-use programming paradigm for distributed applications. Intuitively, it behaves like a distributed shared memory, where applications write and read entries (tuples). When deployed over a wide area network, the tuple space needs to efficiently cope with faults of links and nodes. Erasure coding techniques are increasingly popular to deal with such catastrophic events, in particular due to their storage efficiency with respect to replication. When a client writes a tuple into the system, this is striped into k blocks and encoded into $n > k$ blocks, so that any k out of the n coded blocks are sufficient (and necessary) to reconstruct and read the tuple. Chapter 6 and [PUB1] present several strategies to place those blocks across the set of nodes of a wide area network, that all together form the tuple space. We evaluate the performance tradeoffs of the proposed placement strategies by means of simulations. Furthermore, we implement and test a distributed tuple space supporting erasure coding and block placement.[1] Our results reveal important differences in the efficiency of the strategies, for example in terms of the block fetching latency, and confirm that the knowledge of the network topology is paramount to select the appropriate block placement strategy.

1.3.3 Random entanglement for anticensorship

The last two chapters of the thesis provides an analysis of system's valuable properties that can be enhanced using data entanglement on top of an erasure code. In particular, we combine random data entanglement and MDS codes for anti-censorship.

Users entrust an increasing amount of data to online cloud systems for archival purposes. Existing storage systems designed to preserve user data unaltered for decades do not, however, provide strong security guarantees—at least at a reasonable cost. Chapter 7 and [PUB3] introduce RECAST, an anti-censorship data archival system based on random data entanglement. Documents are mixed together using an entanglement scheme that exploits erasure codes for secure and tamper-proof long-term archival. Data is intertwined in such a way that it becomes virtually impossible to delete a specific document that has been stored long enough in the system, without also erasing a substantial fraction of the whole archive, which requires a very powerful adversary and openly exposes the attack. We validate RECAST's entanglement approach

[1] The work has been done in collaboration with Vitaly Buravlev, who took care of the block-placement-aware distributed tuple space prototype implementation. Notice, nevertheless, that block placement heuristics and tuple space usecase are orthogonal. Indeed our block placement heuristics can be applied to different usecases, the distributed tuple spaces being one of them.

via simulations.[2] In one of our settings, we show that RECAST, configured with the same storage overhead as triple replication, can withstand 10% of storage node failures without any data loss. Furthermore, we estimate that the effort required from a powerful censor to delete a specific target document is two orders of magnitude larger than for triple replication.

1.3.4 Structured entanglement for reliable storage

Opposite to Chapter 7, we remove the randomness from the data entanglement process and use binary linear codes in Chapter 8 and [PUB4]. In particular, we combine structured data entanglement and binary linear block codes for reliability.

The combination of structured entanglement and linear block codes gives rise to convolutional LDPC codes, which we study over the binary erasure channel for immutable distributed storage. These codes allow the archival of data objects in a sequential fashion on an increasing number of storage nodes as they arrive in the system, as well as fast repair using a simple message passing decoder. We further target systematic codes, high code rates and low locality, which are paramount in this setting. We describe a family of codes that split each archived data object in s blocks, entangle them with $t = s + p$ blocks already archived, and generate p parity blocks per archived data object. We carefully choose the parity-check matrix and the blocks already archived to maximize the repair capability of the resulting codes, and describe the best constructions for $1 \leq s \leq 5$ and $p = 2$. A Markov analysis[3] shows that, for the same storage overhead, our codes are orders of magnitude more reliable than state-of-the-art Reed-Solomon and locally repairable codes.

[2] In [PUB3], besides validating the approach via simulations, we test RECAST by means of a full-fledged prototype, implemented and evaluated by Dorian Burihabwa.

[3] The reliability analysis takes advantage of the work done in collaboration with Laurent Hayez to approximate the number of repairable failures.

Chapter 2

Background

2.1 Linear codes over finite fields

In this section, we summarize main facts about linear codes and we refer to [13, 14] for a complete introduction. We denote $\mathbb{F}_p$ the *finite field* with p elements, for a prime p. The finite field is endowed with two operations, modular addition and modular multiplication, that is, for $a, b \in \mathbb{F}_p$ we have $(a+b \mod p) \in \mathbb{F}_p$ and $(a \cdot b \mod p) \in \mathbb{F}_p$ [15]. In what follows, we refer to the modular addition over $\mathbb{F}_2$ as the XOR and we write $\oplus$. We denote $\mathbb{F}_{p^e}$ the finite field with p^e elements for a prime p and a positive integer e and we refer to [15] for its definition.

A *linear code* $\mathscr{C}$ of *length* n and *dimension* $k \leq n$ over $\mathbb{F}_p$ is a k-dimensional vector subspace of the vector space $(\mathbb{F}_p)^n$ and we indicate it as $\mathscr{C}(k, n-k)$. The elements of the code are *codewords*, and the n components of a codeword are *blocks*. Two well-known matrices describe a linear code. The *generator matrix* G maps a message of $(\mathbb{F}_p)^k$ to its corresponding codeword

$$\mathscr{C} = \{m \cdot G : m \in (\mathbb{F}_p)^k\}.$$

The *parity-check matrix* H has the property that $GH^t = 0$ and can be used to check whether or not a word x belongs to the code

$$x \in \mathscr{C} \Leftrightarrow H \cdot x^t = 0.$$

The code is *systematic* if messages can be found unchanged in the corresponding codewords. Equivalently, a $k \times n$ generator matrix is in systematic form when k out of its n columns form the $k \times k$ identity matrix I_k.

Proposition 2.1. *[14, Theorem 1.2.1] Let A be a $k \times (n-k)$ matrix. For a code of length n and dimension k with generator matrix $G = (I_k \mid A)$ in systematic form, the parity check matrix is $H = (-A^t \mid I_{n-k})$.*

The *Hamming distance* between two vectors is the number of coordinates in which they differ. The *Hamming weight* $\omega(v)$ of a vector v is the number of non-zero coordinates. Clearly, the Hamming distance between two vectors v,w is the Hamming weight of the vector $v - w$.

The third important parameter of a linear code $\mathscr{C}$, besides the length and dimension, is the *minimum distance* d between its codewords

$$d = \min_{c \in \mathscr{C} \setminus \{0\}} \omega(c)$$

which can be computed from a parity-check matrix of the code as follows.

Proposition 2.2. *[14, Corollary 1.4.14] A linear code has minimum distance d if and only if its parity-check matrix has a set of d linearly independent columns but no set of $d-1$ linearly dependent columns.*

Let $\lfloor x \rfloor$ denote the greatest integer less than or equal to x. The minimum distance expresses the correction capability of the code. Indeed, a code with minimum distance d can correct $\lfloor \frac{d-1}{2} \rfloor$ errors. Erasures are failures occurring at a known position so they do not need to be detected. Hence, a code with minimum distance d can correct $d-1$ erasures.

The Singleton bound is an upper bound on the minimum distance of a code $\mathscr{C}(k, n-k)$, which for linear codes reads as

$$d \leq n - k + 1. \tag{2.1}$$

Codes reaching this bound with equality are called MDS codes (Maximum Distance Separable) and we write $\text{MDS}(k,n-k)$ for a MDS code of length n and dimension k. Thus, MDS codes have the greatest minimum distance possible, that is, fixed n and k (the *storage overhead* $\frac{n-k}{k}$ is also fixed), they maximize the correction capability. We summarize this by saying that MDS codes optimize the storage overhead/fault tolerance tradeoff.

A well-known family of MDS codes is Reed-Solomon (RS) codes [16]. Let $\mathbb{F}_q \setminus \{0\} = \{P_1, \ldots, P_n = P_{q-1}\}$ and $\mathbb{F}_q[x]$ be the polynomial ring over $\mathbb{F}_q$. Then

$$\text{RS}(k,n-k) = \Big\{\big(f(P_1), \ldots, f(P_n)\big) : f \in \mathbb{F}_q[x], \deg(f) \leq k-1\Big\},$$

that is, $\text{RS}(k,n-k)$ is obtained evaluating all polynomials with coefficients in $\mathbb{F}_q$ of degree up to $k-1$ at all non-zero points of $\mathbb{F}_q$ [15]. Thus, its generator matrix is

$$G_{\text{RS}(k,n-k)} = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ P_1 & P_2 & \cdots & P_n \\ \vdots & \vdots & \ddots & \vdots \\ P_1^{k-1} & P_2^{k-1} & \cdots & P_n^{k-1} \end{pmatrix}. \tag{2.2}$$

In practice, Cauchy-based Reed-Solomon codes are the fastest in encoding and decoding in their family. The theory behind these codes is presented in [17] and the construction is based on Cauchy matrices. Indeed, Cauchy matrices can be inverted significantly faster than arbitrary matrices, in time proportional to n^2 for a $n \times n$ matrix. More in detail, a Cauchy matrix is of the form

$$\begin{pmatrix} \frac{1}{x_1+y_1} & \frac{1}{x_1+y_2} & \cdots & \frac{1}{x_1+y_n} \\ \frac{1}{x_2+y_1} & \frac{1}{x_2+y_2} & \cdots & \frac{1}{x_2+y_n} \\ & & \ddots & \\ \frac{1}{x_m+y_1} & \frac{1}{x_m+y_2} & \cdots & \frac{1}{x_m+y_n} \end{pmatrix}$$

where $x_i + y_j \neq 0$ for all $1 \leq i \leq m$ and $1 \leq j \leq n$, all x_i for $1 \leq i \leq m$ are distinct, and all y_j for $1 \leq j \leq n$ are distinct. In particular, for an integer $L > 1$, we can build a $2^{L-1} \times 2^{L-1}$ Cauchy matrix over $\mathbb{F}_{2^L}$ by choosing, for $1 \leq i \leq 2^{L-1}$:

1. $x_i \in \mathbb{F}_{2^L}$ whose binary expansion is the binary expansion of $i-1$,
2. $y_i \in \mathbb{F}_{2^L}$ whose binary expansion is the binary expansion of $2^{L-1} + i - 1$.

A Cauchy matrix is non-singular and its sub-matrices are themselves Cauchy, hence non-singular [17]. As a consequence, given C a $(n-m)\times m$ Cauchy matrix over $\mathbb{F}_{2^L}$, the matrix $G = (I_m \mid C)$ is the generator matrix of a systematic MDS code of dimension m and length n over $\mathbb{F}_{2^L}$ [17].

Unfortunately, finite field arithmetic highly penalizes the performance. Hence, a method for writing Reed-Solomon codes based on Cauchy matrices as XOR-based codes is presented in [17]. The basic idea of the XOR-based representation is that elements of $\mathbb{F}_{2^L}$ can be represented as vectors of size L and matrices of size $L\times L$ over $\mathbb{F}_2$. In particular, the vector representation of the sum $\alpha+\beta \in \mathbb{F}_{2^L}$ can be obtained adding the vector representations for $\alpha \in \mathbb{F}_{2^L}$ and $\beta \in \mathbb{F}_{2^L}$. Furthermore, the matrix-vector multiplication yields the vector representation of the product $\alpha\beta \in \mathbb{F}_{2^L}$.

To obtain the matrix representation of $f \in \mathbb{F}_{2^L}$, first, we write $\mathbb{F}_{2^L} = \mathbb{F}_2/(p(x))$ where $p(x) \in \mathbb{F}_2[x]$ is an irreducible polynomial of degree L. Then, we identify an element $f \in \mathbb{F}_{2^L}$ with the polynomial $f = \sum_{i=0}^{L-1} f_i x^i \in \mathbb{F}_2[x]$ and call *coefficient vector* the column vector $(f_0, \ldots, f_{L-1})^t$. The matrix representation of $f \in \mathbb{F}_{2^L}$ is

$$\tau(f) = \begin{pmatrix} \vdots & & \vdots & & \vdots \\ f \mod p(x) & xf \mod p(x) & \ldots & x^{L-1}f \mod p(x) \\ \vdots & & \vdots & & \vdots \end{pmatrix} \tag{2.3}$$

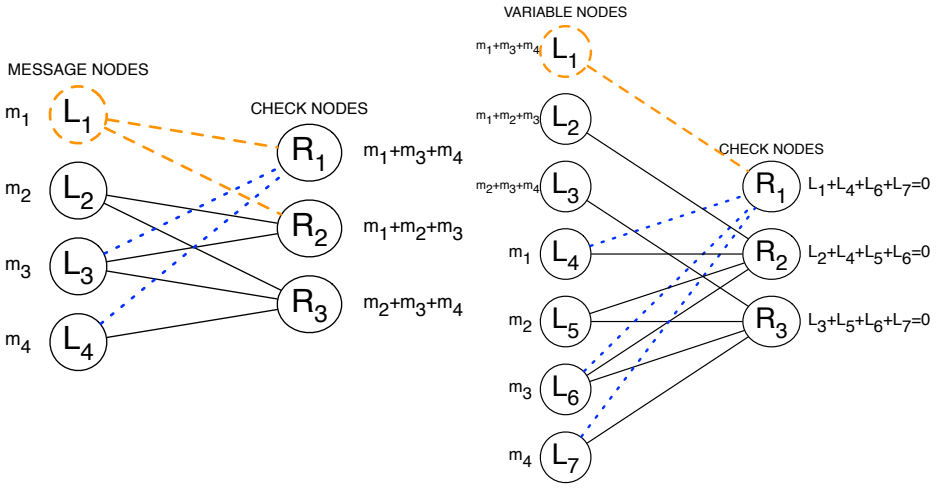
where the i-th column is the coefficient vector of $x^{i-1}f \mod p(x)$.

Based on the vector and matrix representations for the elements of $\mathbb{F}_{2^L}$, a general method for turning a systematic, linear code over $\mathbb{F}_{2^L}$ into a systematic, linear code over $\mathbb{F}_2$ is given in [17]. For a code of length n and dimension m, let $L \geq \max(\log(m), \log(n-m))$. Let the message M have m blocks each containing L words of arbitrary size w, and replace each field element in the message over $\mathbb{F}_{2^L}$ by its vector representation over $\mathbb{F}_2$. Let $\tilde{C}$ be a $(n-m)\times m$ Cauchy matrix over $\mathbb{F}_{2^L}$. Then consider $C = (I_m \mid \tilde{C}) = (c_{ij})$ for $1 \leq i \leq n$ and $1 \leq j \leq m$ and replace each element in C by its matrix representation over $\mathbb{F}_2$ to obtain the generator matrix G of the XOR-based code: $G = (\tau(c_{ij}))$ for $1 \leq i \leq n$ and $1 \leq j \leq m$. The j-th packet E_j of the encoding of M consists of the rows $r_{jL+1}, \ldots, r_{(j+1)L}$ of the matrix $G \cdot M$.

The encoding of the XOR-code can be done using $O(m(n-m)L^2)$ XORs of words of size w. The decoding of the XOR-code can be done using $O(mkL^2)$ XORs of words of size w and $O(k^2)$ arithmetic operations in the field $\mathbb{F}_{2^L}$, assuming that $m-k$ information packets and k redundant packets are given. An efficient implementation achieving the results from [17] is available at [18].

2.2 Binary LDPC codes

A low-density parity-check (LDPC) code is a block code that has a sparse parity-check matrix [19, 20]. In this thesis, we consider binary LDPC codes only, that is, LDPC codes having parity-check matrix over $\mathbb{F}_2$. LDPC codes are usually represented with bipartite graphs, also called Tanner graphs. In the literature, we find two distinct ways of expressing such a graph: the message/check node method, e.g. [21], and the variable/check node method, e.g. [22]. While the first method only applies to systematic LDPC codes, the second can describe systematic and non-systematic LDPC codes. We present the two approaches



(a) Message/check node graph B_1. (b) Variable/check node graph B_2.

Figure 2.1: Tanner graph representations for LDPC codes.

using a toy example. We consider the code of dimension $k = 4$ and length $n = 7$ defined by the following generator G and parity-check H matrices

$$G = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}, \quad H = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}.$$

2.2.1 Message/check node method

The bipartite graph B_1 in Figure 2.1(a) has k left message nodes and $n-k$ right check nodes. Each left node represents a data block and each right node the XOR of the left nodes connected to it. Hence, the code is in systematic form.

The matrix $M(B_1)$ corresponding to the message/check node graph is the sub-matrix corresponding to the non-systematic part of G or H, that is

$$M(B_1) = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix}.$$

On the message/check node graph, to encode we simply compute parity bits as prescribed by right nodes. We now describe one step of iterative decoding on this graph. For example, suppose node L_1 is erased. We check its neighbouring right nodes (follow the orange dashed lines in Figure 2.1(a)) and we pick one, say C, connected only to alive nodes (except for L_1). Let us call A the set of alive variable nodes connected to C (follow the blue dotted lines in Figure 2.1(a)). We compute L_1 as the XOR of the selected neighbouring right node C with the alive nodes in A. Iterative decoding proceeds in this fashion for all the erased node. It stops when either all erased left nodes have been repaired or

when for the remaining erased nodes it cannot find a right neighbour connected only to alive nodes (except the one to repair).

2.2.2 Variable/check node method

In this thesis, we use the variable/check node representation in which left nodes represent encoded blocks and right nodes are constraints over variable nodes, that is, the XOR of the left nodes incident to a certain right node is 0. The bipartite graph B_2 in Figure 2.1(b) has n variable nodes and $n-k$ check nodes. The variable/check node graph corresponds to the parity-check matrix H of the code; in particular, there is an edge from L_u to R_v if and only if $h_{vu} = 1$ where h_{vu} is the element on row v and column u of H.

For encoding, from the graph B_2 the $k \times n$ generator matrix G is generated and then used to calculate the n encoded symbols held by variable nodes [23]. To see how one step of the iterative decoding works, suppose node L_1 is erased. We check its neighbours right nodes, it has only R_1. If all variable nodes adjacent to R_1 (except for L_1) are alive, we can compute L_1 as the XOR of those variable nodes (follow the orange dashed line, then the blue dotted ones in Figure 2.1(b)). We complete the discussion about iterative decoding on the variable/check node graph in Section 2.2.4.

2.2.3 Degree distribution and stopping set

For LDPC codes we define the variable node degree distribution $\lambda(x) = \sum \lambda_i x^{i-1}$ and the check node degree distribution $\rho(x) = \sum \rho_i x^{i-1}$ where λ_i, ρ_i are the fractions of edges of degree i on the left and on the right, respectively. The average node degree on the left, denoted a_l, and on the right, denoted a_r, satisfy respectively $a_l^{-1} = \sum_i \frac{\lambda_i}{i}$ and $a_r^{-1} = \sum_i \frac{\rho_i}{i}$ [21].

We say that the LDPC code is left regular with left degree l if and only if $\lambda_{\mathsf{l}} = 1$ and $\lambda_i = 0$ for $i \neq \mathsf{l}$, that is, $\lambda(x) = x^{\mathsf{l}-1}$. We say that the LDPC is right regular with right degree r if and only if $\rho_{\mathsf{r}} = 1$ and $\rho_i = 0$ for $i \neq \mathsf{r}$, that is, $\rho(x) = x^{\mathsf{r}-1}$. A (l, r)-regular LDPC code is both l-left and r-right regular.

In [24], the authors give a combinatorial characterization of decoding failures and use it to develop a finite-length analysis of LDPC codes when used over the binary erasure channel, see Section 4.5.3. In particular, they identify stopping sets as the key object to evaluate the finite-length performance of LDPC codes. A *stopping set* S is a subset of the n variable nodes such that all neighbours of S are connected to S at least twice [24]. The stopping number σ is the length of the smallest stopping set and it represents the minimum number of erasures that cannot be corrected by message passing [22].

For a LDPC code of minimum distance d (i.e., the smallest Hamming weight of non-zero codewords) it holds $\sigma \leq d$ [25]. For regular codes with left degree 2, the stopping number is at most in $O(\log n)$, which follows from $\sigma \leq d$ and the fact that the minimum distance d is in $O(\log n)$ for these codes [25]. Moreover, if the variable nodes have degree 2 and the graph has girth g (length of the shortest cycle), then the stopping number is $\sigma = \frac{g}{2}$ [26].

2.2.4 Message passing on the BEC

We now complete the discussion on iterative decoding started in Section 2.2.2. In message passing iterative decoding algorithms, messages are exchanged between the variable and the check nodes in discrete time steps [27, 28, 29]. In a variable-to-check message round, each variable node passes along its value $\{0, 1, e\}$ to the neighbouring check nodes. In a check-to-variable message round, each check node passes to a neighbour v either an erasure, if it receives an erasure from at least one neighbour besides v, or the parity of the bits received from neighbours other than v.

The algorithm above can be implemented so that each edge of the Tanner graph is used exactly once; this version of the iterative decoder has been labelled the *peeling decoder* [29]. Once the value of a variable node is known, it is sent to neighbouring check nodes and the variable node and its incident edges are removed from the graph. When a check node has degree 1, i.e., the values of its neighbouring variable nodes are all known except one, it sends the parity value to its remaining neighbour and both check node and the edge incident to it are removed.

Message passing is not guaranteed to decode all message nodes and decoding is successful provided that [21, 27]

$$\rho(1 - \epsilon\lambda(x)) > 1 - x \tag{2.4}$$

for all $0 < x \leq 1$, or equivalently provided that for all $0 < y \leq 1$

$$\epsilon\lambda(1 - \rho(y)) < 1 - y. \tag{2.5}$$

Notice that this condition does not assure the repair of all the missing left nodes; it only guarantees continuation of the recovery process as long as the number of edges in the induced subgraph is a constant fraction of the original number of edges [27]. To prove that the decoding is successful, in addition, the left degree is required to be different from 1 and 2 [27, Theorem 2].

The dual condition (2.5) can be used to compute the maximal fraction of erasures that can be recovered using message passing. Indeed, such a value is the supremum ϵ satisfying (2.5) for all $0 < y \leq 1$, that is, $\epsilon < \frac{1-y}{\lambda(1-\rho(y))} = f(y)$. We compute the minimum e of the function $f(y)$ as the unique root in $(0, 1]$ of $f'(y) = 0$. Then $\epsilon < f(e)$ and $f(e)$ gives the average percentage of erasures that can be corrected by a LDPC code with distributions $\lambda(x)$, $\rho(x)$. We aim at having capacity-achieving codes, i.e., codes with rate R such that $1 - R$ is arbitrarily close to ϵ from (2.5).

LDPC codes arising from regular graphs are not close to optimal. Indeed, for a random bipartite regular graph with left degree $\mathsf{l} \geq 3$ and right degree $\frac{\mathsf{l}}{\beta}$ the condition (2.4) holds only if $\epsilon \leq 4\left(1 - \left(\frac{1}{1-\mathsf{l}}\right)^{\frac{1}{\frac{1}{\beta}-1}}\right)$ [21]. Hence, the maximum acceptable loss rate goes to 0 as l goes to ∞.

For $\mathsf{l} = 2$, the maximum acceptable loss rate goes is 0 [29]. Using condition (2.5), it is possible to prove that the performance of regular graphs deteriorates as the left degree increases. Thus, the best performance is obtained when $\mathsf{l} = 3$ [30], and to have positive rate for $\mathsf{l} = 3$, it must be $\mathsf{r} \geq 4$ [29].

2.2.5 Complexity considerations

There is no polynomial time algorithm for computing the size of the smallest stopping set of a Tanner graph unless P = NP [31]. Furthermore, in [32] it is shown that the size of the smallest stopping set is also hard to approximate. For any $\epsilon > 0$, a $(1+\epsilon)$-approximation algorithm (running in polynomial time in the size of the input x) for a minimization problem P is an algorithm that for any input x for P returns a feasible solution y such that the cost of y is bounded by $(1+\epsilon)$ times the cost of an optimal solution for the instance x. For the problem of finding the size of the smallest stopping, there does not exist any polynomial time $(1+\epsilon)$-approximation algorithm for any constant $\epsilon > 0$ unless P = NP [32].

These results are further improved in [33]: the authors show that the problem of finding the smallest stopping set cannot be approximated within $o(\log N)$, where N denotes the description length of the problem, unless P = NP. In the same paper, the hardness of approximation results for LDPC codes (which correspond to Tanner graphs with sparse parity-check matrix) is also proved. In general, the fact that a problem is NP-hard does not imply that a special instance of that problem is NP-hard. But for the stopping set problem, there exists a constant $\alpha > 1$ such that it is NP-hard to α-approximate the stopping set of minimum cardinality in the Tanner graph of an LDPC code [33].

FTP (fixed-parameter tractable) is the class of parametrised problems that allow for the existence of a polynomial time algorithm. The problem of finding the stopping set of minimum cardinality is W[1]-hard (FPT = W[0] and W[0] $\subseteq$ W[1]) [33]. It has complexity growing exponentially with the size of the smallest stopping set, but only polynomially with the length of the code. This means that one can find the smallest stopping of fairly long codes provided that the size of the stopping set is small enough (10-15 [33]).

To overcome the intractability results, enumeration algorithms have been developed. The algorithm in [34] can find an exhaustive list of stopping sets up to size 13. An efficient way to find all the stopping sets smaller than a certain threshold for a fixed LDPC code is proposed in [35] and further developed in [36]. Integer programming is used in [37] to describe and calculate the smallest stopping set size on the BEC, for a wide variety of practical code lengths of both regular and irregular LDPC codes.

Chapter 3

Reliability evaluation for distributed storage systems

Erasure codes are a mean to make storage systems reliable. But how reliable? Let us think about two Reed-Solomon codes, e.g., RS(k,$n-k$) and RS($2k$,$2n-2k$), with the same storage overhead but different levels of fragmentation. Do they offer the same reliability? To answer such a question, that is, to be able to compare different coding techniques with respect to the reliability they offer, traditional reliability models were constructed with some simplifying assumptions: the only failures are whole-device failures, devices fail at a constant rate, devices repair at a constant rate, and failures are independent [38]. With these hypotheses computing the mean time to data loss (MTTDL) via Markov chain is a relatively easy task.

Despite the fact of being not universally accepted, see Section 3.4, we use Markov chains for reliability evaluation because they are easy to handle, widely used, e.g. [39, 40, 41], and applicable to a wide variety of coding techniques including MDS, LRC, XOR-based and LDPC codes.

3.1 Markov model for MDS codes

We present the Markov model to evaluate the MTTDL of a representative codeword of a MDS(k,$n-k$) code, which can withstand up to $n-k$ erasures.

The representation of a Markov chain with 5 states is given in Figure 3.1. In State 0 the codeword is intact and in the last state of the chain, the data loss (DL) state, it cannot be repaired. The transition rates σ_i and μ between these states are characterized by mean time to failure (`mttf`), or equivalently drive failure rate $\lambda = \frac{1}{\texttt{mttf}}$, and mean time to repair (`mttr`), or equivalently drive repair rate $\mu = \frac{1}{\texttt{mttr}}$. We show common values for those parameters in state-of-the-art

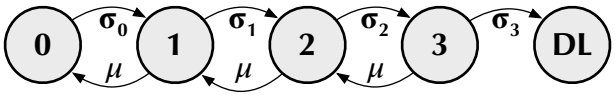


Figure 3.1: A Markov chain for a MDS code correcting up to 3 erasures.

System	N	`mttf`	`mttr`
HDFS-RAID [40]	3000	4 years	0.25 s
Windows Azure Storage [41]	400	NOT PROVIDED	30 min
Google cells [39]	1000-7000	10-50 years (disks) 4.3 months (nodes) 10.2 years (racks)	NOT PROVIDED

Table 3.1: Markov chain parameters for state-of-the-art systems. The number of nodes N is not included in the Markov model.

systems in Table 3.1. The longer the mean time to failure, the more reliable the system; the shorter the mean time to repair, the more reliable the system. The states and the transition rates are the building blocks of the Markov model:

- The state of the chain is the number of erased blocks in the representative codeword.
- The rate σ_i of the transition from State i to State $i+1$ is

$$\sigma_i = (n-i)\lambda p_i \tag{3.1}$$

  where p_i is the probability that the code can withstand one more node failure, given that it has already tolerated i failures. We compute such a conditional probability as

$$p_i = \frac{r(i+1)}{r(i)} \tag{3.2}$$

  where $r(i)$ is the fraction of repairable failures of size i [42].
- The rate μ from State $i+1$ to State i is determined by the recovery rate of a single block as we assume that the recovery does not depend on the total number of available chunks [39].

In general, the transition matrix [43] for the Markov model of a MDS code correcting at most τ erasures is [42]

$$\hat{P} = \begin{pmatrix} -\sigma_0 & \sigma_0 & & & & 0 \\ \mu & -(\mu+\sigma_1) & \sigma_1 & & & 0 \\ & \ddots & \ddots & \ddots & & \\ & & \mu & -(\mu+\sigma_{\tau-1}) & \sigma_{\tau-1} & 0 \\ & & & \mu & -(\mu+\sigma_\tau) & \sigma_\tau \end{pmatrix}. \tag{3.3}$$

From the transition matrix $\hat{P}$, to find the MTTDL, we compute the matrix P which is the negative of $\hat{P}$ after removing the last column. Then

$$\mathrm{MTTDL}(\hat{P}) = \begin{pmatrix} 1 & 0 & \dots & 0 \end{pmatrix} P^{-1} \begin{pmatrix} 1 & 1 & \dots & 1 \end{pmatrix}^t \tag{3.4}$$

which is the sum of the entries in the first row of P^{-1} and can be found by solving for y_1 the linear system

$$P \begin{pmatrix} y_1 & y_2 & \dots & y_t \end{pmatrix}^t = \begin{pmatrix} 1 & 1 & \dots & 1 \end{pmatrix}^t.$$

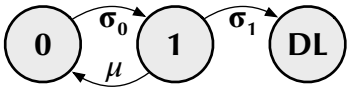


Figure 3.2: A Markov chain for a MDS code correcting single erasures only.

The solution y_1 of the linear system is the MTTDL of a representative stripe. A storage system comprises many such stripes, and to compute the MTTDL of the system we divide the MTTDL of a representative stripe by the number of stripes in the system [42]. We now solve analytically Equation (3.4) for $\tau = 1, 2$, that is, for codes correcting at most 1 and 2 erasures respectively.

Repair up to 1 erasure The Markov chain corresponding to a MDS$(n-1,1)$ code, $n \geq 2$, is shown in Figure 3.2. The corresponding transition matrix is

$$\hat{P} = \begin{pmatrix} -\sigma_0 & \sigma_0 & 0 \\ \mu & -(\mu+\sigma_1) & \sigma_1 \end{pmatrix}$$

and the MTTDL of a representative stripe is y_1 such that $Py = 1$ where P is the negative of matrix $\hat{P}$ deprived of the last column, i.e.,

$$\begin{pmatrix} \sigma_0 & -\sigma_0 \\ -\mu & \mu+\sigma_1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

By Gaussian elimination, we get

$$y_1 = \frac{1}{\sigma_0} + \frac{\sigma_0+\mu}{\sigma_0\sigma_1}.$$

Repair up to 2 erasures The Markov chain corresponding to a MDS$(n-2,2)$ code, $n \geq 3$, has 3 transient states plus the DL state and its transition matrix is

$$\hat{P} = \begin{pmatrix} -\sigma_0 & \sigma_0 & 0 & 0 \\ \mu & -(\mu+\sigma_1) & \sigma_1 & 0 \\ 0 & \mu & -(\mu+\sigma_2) & \sigma_2 \end{pmatrix}.$$

The MTTDL of a representative stripe is y_1 such that $Py = 1$ where P is the negative of matrix $\hat{P}$ deprived of the last column, i.e.,

$$\begin{pmatrix} \sigma_0 & -\sigma_0 & 0 \\ -\mu & \mu+\sigma_1 & -\sigma_1 \\ 0 & -\mu & \mu+\sigma_2 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}.$$

By Gaussian elimination, we get

$$y_1 = \frac{\mu+\sigma_2}{\mu\sigma_2}\Big(\frac{\mu}{\sigma_1} + \frac{\mu^2}{\sigma_0\sigma_1} + 1\Big) - \frac{1}{\mu} + \frac{1}{\sigma_0}.$$

3.2 Model for non-MDS codes

Non-MDS codes are sometimes called *irregular* codes [38] to emphasize the fact that their repair properties are not simply described by their minimum distance.

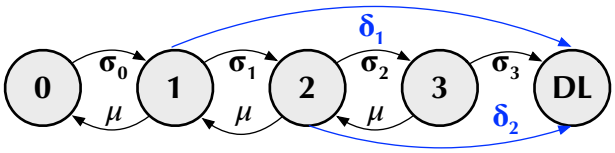


Figure 3.3: A Markov chain for a non-MDS code correcting all single failures and some failures of size 2 and 3.

Indeed, a non-MDS code $\mathscr{C}(k, n-k)$ with minimum distance d can withstand all failures up to size $d-1$ but also some of the i failures for $i \geq d$. Thus, there are some $r(i)$ strictly greater than 0 and strictly smaller than 1, meaning that in the Markov model there are arrows to the DL state not arising from the second to last state.

An example of Markov model used to evaluate the reliability of a representative codeword for an irregular code of minimum distance $d = 2$ is given in Figure 3.3. The difference from Figure 3.1 is the transition rate

$$\delta_i = (n-i)\lambda(1-p_i) \tag{3.5}$$

to go directly to DL from State i (for a MDS-code $\delta_i = 0$ for every $i \leq n-k$). The transition matrix corresponding to the Markov chain in Figure 3.3 is

$$P = \begin{pmatrix} -\sigma_0 & \sigma_0 & 0 & 0 & 0 \\ \mu & -(\mu+\sigma_1+\delta_1) & \sigma_1 & 0 & \delta_1 \\ 0 & \mu & -(\mu+\sigma_2+\delta_2) & \sigma_2 & \delta_2 \\ 0 & 0 & \mu & -(\mu+\sigma_3) & \sigma_3 \end{pmatrix}.$$

3.3 More accurate model for MDS codes

With the method presented in Section 3.1, the reliability of a single stripe is extrapolated to large systems by dividing by the number of stripes. In [42] the authors show that this is not exactly correct though it is a good approximation in practice. Indeed, let us consider two codewords of a RS(2,2), which can correct two erased blocks per codeword. With the standard method in Section 3.1, the MTTDL of a representative stripe of length 4 is computed setting $r(0) = r(1) = r(2) = 1$ and $r(3) = r(4) = 0$. Then, the MTTDL of the system is extrapolated by dividing the stripe MTTDL by two, which is the number of stripes in the system. However, RS(2,2) can correct up to four erased blocks if they are in different codewords. In detail, two erased blocks are always repairable, $r(3) = \frac{48}{56}$, $r(4) = \frac{36}{70}$ and five or more erased blocks lead to data loss. This suggests

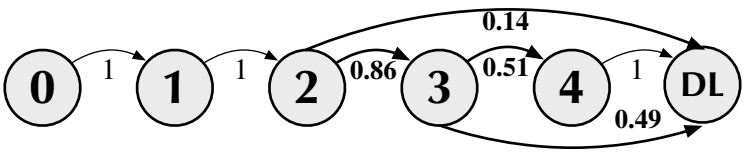


Figure 3.4: Markov chain for two RS(2,2)'s stripes.

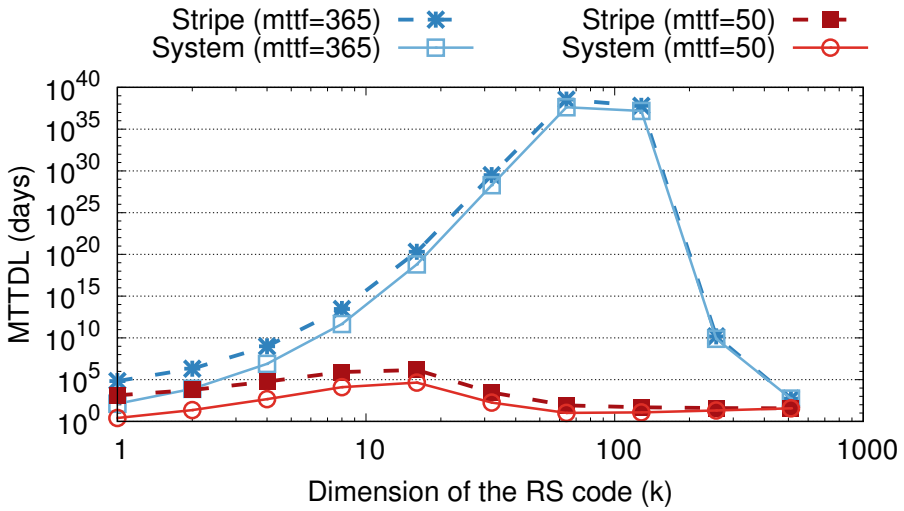


Figure 3.5: Does fragmentation improve reliability? We calculate the MTTDL for RS(k,k) codes with $k \in \{2^i : 0 \leq i \leq 9\}$ and `mttf` $= \{50, 365\}$ days respectively for the red and blue curves. We set `mttr` $= 1$ day.

that the MTTDL of the system is better than exactly half that for a single stripe of length 4 and is correctly modelled by the Markov chain in Figure 3.4. In the figure, we report the fraction $r(i)$ of repairable failures of size $1 \leq i \leq 5$ computed considering the two RS(2,2) stripes as a unique codeword of length 8. Notice that such a Markov chain has a transition matrix of the form presented in Section 3.2.

In general, to describe the Markov model that evaluates the reliability of a system comprising S stripes of a RS(k,$n-k$) code as the reliability of a system hosting a single codeword of length nS, we need to compute $r(i)$ for $n-k+1 \leq i \leq (n-k)S$. Indeed, all failures are repairable below the distance and $(n-k)S+1$ erasures or more result in data loss, as there is at least a stripe with $(n-k)+1$ erasures that cannot be repaired.

3.4 Discussion

Markov chains are not universally accepted as a model for storage systems reliability because of their hypotheses mismatching the real world. In this section, we discuss the criticized unrealistic hypotheses and present a possible alternative.

3.4.1 Fragmentation

Under independent failures, fragmentation improves reliability [44]. More explicitly, increasing the dimension k (the fragmentation level) of the erasure code, fixed the storage overhead, the reliability gets better. In term of Markov chains, which do assume independent failures, fixed the storage overhead, the higher the fragmentation level is $(\frac{k}{n}, \frac{2k}{2n}, \frac{3k}{3n}, \dots)$, the longer the MTTDL.

Nevertheless, in our computation, we only observe such a behaviour up to a certain threshold. Indeed, in Figure 3.5, we see that a RS(8,8) code has better

Trace	Duration (days)	Mean # up nodes	f (per day)	a
PlanetLab	527	303	0.017	0.97
Microsoft PCs	35	41970	0.038	0.91
Skype	25	710	0.12	0.65
Gnutella	2.5	1846	0.30	0.38

Table 3.2: Availability traces: estimates for a and f from [46] where a is the probability that a node is available and f is the fraction of nodes that fail permanently per time unit.

reliability than a RS(1,1) code as expected, but for $k \geq 128$ the reliability of the system gets worse. In particular, we compare systems of the same size, i.e., we consider 512 codewords of length 2, 256 codewords of length 4, 128 codewords of length 8, and so on up to 2 codewords of length 512 and 1 codeword of length 1024. We numerically compute the MTTDL of a representative stripe, then divide by the number of codewords in the system to get the MTTDL of the system, see Section 3.1.

The drop observed in Figure 3.5 is in contradiction with the idea that fragmentation always improves reliability under the assumption of independent failures [44]. When `mttr` and `mttf` differ by several orders of magnitude (as for example in [40]), the drop comes when the dimension of the code is extremely large. Indeed, to show the drop in correspondence of dimension 32 we use `mttf` = 50 and `mttr` = 1. It remains an open problem to determine whether the mismatch more fragmentation/higher reliability is due to some unrealistic assumption of the model or the model well reflects the fact that fragmentation can improve reliability only up to a certain threshold.

3.4.2 Independence of failures

A model that captures the average bandwidth used to maintain a file in the system and the resulting average availability of the file is proposed in [45] and exploited in [46]. The model has two main parameters: f is the expected fraction of nodes permanently failing per time unit and a is the probability that a node is available ($1-a$ is the probability for a node to experience a transient failure). Values for f and a are estimated in [46] based on traces of several distributed systems and shown in Table 3.2. The model assumes that a node is available independently of the availability of other nodes.

Such independence assumption is empirically verified on the Overnet sharing network in [47]. In order to characterize the dependence between every host pair, the authors of [47] calculate the conditional probability $P(Y \mid X)$ of host Y being available given that host X is available from the empirical data for every host pair (X, Y). If this value is equal to the probability $P(Y)$ that host Y is available then the availabilities of X and Y are independent. The finding is that more than 30% of the host pairs (X, Y) have $P(Y \mid X) = P(Y)$ and that for 80% of them $|P(Y \mid X) - P(Y)| \leq 0.2$, thus a significant *independence* between host pairs is verified.

Nevertheless, this independence hypothesis is not universally accepted, thus traditional Markov chain models are rejected as valuable instruments for estimating reliability, e.g. in [48], because the assumptions that make the model

easy do not match modern systems. Independence of failures implies exponential distribution of failures. But both failure and repair distribution are better fitted by the Weibull distribution [48].

3.4.3 Memorylessness

Memorylessness of Markov chains makes the model forget about progress. For example, after a disk repair the component is treated as it is brand new.

3.4.4 Beyond Markov chains

As incorporating time-dependent properties in an analytical model is hard and simulations can better capture them, a simulation based approach for measuring reliability is proposed in [48]. In particular, the HFRS simulator [49] is a Python tool for reliability analysis, based on Monte Carlo discrete event simulation. The tool simulates disk failures, disk recovery, sector failures and scrubbing, and spits out a reliability estimate, given the device failure and repair distributions, in terms of probability of data survival (or loss) for a certain system lifetime.

There are two versions of the HFRS [38]. Because of the presence of library `npmath` for random number generation, we believe we benefit from v.2, whose output is the probability of data loss for a specified mission time. The HFRS simulator can be used to evaluate the reliability of MDS codes as well as binary codes with irregular fault tolerance, i.e., flat and array XOR codes. For MDS codes, it is sufficient to input the dimension of the code, the number of parity blocks, the minimum distance d and the minimum number of device failures that leads to data loss (d as well in this case). For XOR-based codes, the description must also include the Tanner graph, the minimal erasure list and the fault tolerance vector. The left/right vertices in the Tanner graph represent data/parity symbols respectively and an edge between left and right vertices represents membership in a parity equation. To define the minimal erasure list and the fault tolerance vector we need a bit of vocabulary from [50].

An *erasure pattern* is a set of erased symbols that results in at least one data symbol being unrecoverable. A *minimal erasure* (pattern) is an erasure pattern in which every erased block is sufficient and necessary for it to be an erasure pattern. Every block is necessary as knowing any block of a minimal erasure allows the recovery of all the other. The blocks are sufficient to form the minimal erasure as having an extra erased block b allows the recovery of b only. Then, the minimal erasure list (MEL) is the list of the code's minimal erasures. The erasure list (EL) is the list of the code's erasure patterns: its elements are either minimal erasures or supersets of minimal erasures and hence MEL $\subseteq$ EL. The erasure vector (EV) counts the number of erasure patterns: EV_i is the number of erasure patterns of size i in the EL. The fault tolerance vector (FTV) gives the probability that a certain number of failures cause data loss, i.e.,

$$\mathrm{FTV}_i = \frac{\mathrm{EV}_i}{\binom{n}{i}}. \tag{3.6}$$

MEL and FTV corresponds to two different modes of operation of the HFRS for determining whether a specific set of failures leads to data loss: the former corresponds to the highest fidelity while the latter can be used for coarser-grained analysis [50].

Parameters of the simulator are the sector failure model, specified by the total number of sectors and the sector failure probability and the component failure and repair distributions, specified by the shape, scale and location parameters of the Weibull distribution. We call $\kappa > 0$ the shape parameter and $\lambda > 0$ the scale parameter of the Weibull distribution. When the independent variable of the Weibull distribution represents the time-to-failure, the shape parameter κ can be interpreted as follows:

- $k < 1$ indicates that the failure rate decreases over time. This happens if there is significant infant mortality, or defective items failing early. The failure rate decreases over time as the defective items are weeded out of the population.
- $k = 1$ indicates that the failure rate is constant over time. The Weibull distribution reduces to an exponential distribution.
- $k > 1$ indicates that the failure rate increases over time. This can be caused by an ageing process, or parts that are more likely to fail as time goes on.

Despite taking into consideration realistic hypotheses, the HFRS simulator cannot deal with non-binary irregular codes (e.g., LRC codes in Section 4.3) and non-binary vector codes (e.g., regenerating codes in Section 4.4), reducing the spectrum of state-of-the-art codes that can be evaluated and compared.

Chapter 4

State-of-the-art codes for distributed storage systems

Reliability can be ensured by the introduction of redundancy, the simplest form being replication. As a generalization of replication, erasure coding offers better storage efficiency. In particular, MDS codes offer the best fault tolerance/storage overhead tradeoff. However, MDS codes are suboptimal in distributed environments because the same amount of blocks is used for reconstruction regardless of the need to reconstruct a single block or the whole document. This shortcoming is particularly evident in distributed storage systems because of the *repair problem* [40]: when a single node fails, one block is lost from each stripe stored on that node. MDS codes are suited to recover from the loss of many blocks in a stripe, not really for recovering single failures in many stripes. In the latter case, they entail an avoidable network burden, as the number of nodes contacted and the amount of data moved through the network are suboptimal. We dedicate the first part of this section to erasure codes tailored to the repair problem. In particular, we give further details on the repair operation in Section 4.1. In Section 4.2, we introduce the repair cost metrics that motivate the study of novel erasure coding schemes, which we review in Sections 4.3 and 4.4.

In Section 4.5, we overview few, still uncommon, works on LDPC codes targeting storage applications. Finite-length analysis as well as the study of *small* LDPC codes slowly rise [23, 24], even though this family of codes has traditionally been studied asymptotically, i.e., when the code length tends to infinity. This hypothesis is unrealistic for storage systems. Finally, in Section 4.6, we present data entanglement, which can be used to enhance desirable properties of distributed storage systems such as anti-censorship (Chapter 7) and reliability (Chapter 8).

4.1 Repair operation

Due to the frequent temporary and permanent failures that occur in distributed storage systems, blocks may be unavailable [51]. *Repair operations* replace these blocks in order to maintain the desired level of reliability. Regardless of the particular method which depends on the coding techniques, repair operations

are executed as a background job. Opposite, degraded reads serve requests for currently unavailable data, by reconstructing it on-the-fly. Notice that a repair operation often reconstructs a block per codeword, while a degraded read has the purpose of serving the whole data object to the user.

Repair operations fall in one of two categories: exact or functional repair. With *functional repair*, a failed node is replaced by a node that is functionally equivalent, that is, a specific property, depending on the particular scheme, is maintained by the repaired system, see for example [52, 53]. In contrast, the problem of recovering the failed node exactly, by restoring the exact same blocks as they were on the failed node, is known as *exact repair* [54]. There is no change in the coefficients of a replaced node under exact-repair, which greatly simplifies the implementation and avoids communication overheads during the repair [52]. Furthermore, exact repair is mandatory for systematic codes [55].

4.2 Metrics

Distributed storage systems attempt to provide two types of reliability: availability and durability [46]. A data object is available when it can be reconstructed from the data stored on currently available nodes. The durability of a data object is maintained if it has not been lost due to permanent node failures, that is, it may be available at some point in the future. Important coding metrics that concur in determining a reliable system are:

1. *Fault tolerance*, i.e., the number of node failures that can be tolerated by the system without data loss;
2. *Storage overhead*, i.e., the ratio of the storage space dedicated to the redundancy over the total storage space;
3. *Speed of encoding* and *decoding*.

The higher the storage overhead and the fault tolerance the more reliable the system. But while a high fault tolerance is desirable, a high storage overhead is not. This leads to choose coding techniques with length n and dimension k such that:

- $n-k$ is small compared to k, typically $\frac{n-k}{k} < 1$;
- the minimum distance d is as close as possible to the Singleton bound, that is, to $n-k+1$ for linear codes.

This motivates the traditional choice of Reed-Solomon codes, that can be built for any k and n and reach the Singleton bound with equality.

Regarding the speed of encoding and decoding, in their fastest instance $\text{RS}(k,n-k)$ codes can be encoded and decoded in quadratic time with respect to k, see Section 2.1. More precisely, algorithms for encoding and decoding Reed-Solomon codes in time $O(n \log^2 n \log\log n)$ are known, but for small values of n, quadratic time algorithms are faster than the theoretically, asymptotically fast algorithms [27].

When coding is used on a distributed storage system, if a node fails, in order to maintain the same level of reliability, the system needs to re-create encoded information at a replacement node. The consideration of the repair network

	Section	Length	Dimension	Fault tolerance	Repair locality	Helper nodes	Systematic	Vector code	Binary	Year
MDS [16]	2.1	n	k	$n-k$	k		✓	✗	✗	1960
LRC [56]	4.3.1	$n(r+1)$	kr	$n-k$	r		✓	✗	✗	2014
LRC [57]	4.3.2	$n+\tau-1$	k	$n-k$	k[a]		✓	✗	✗	2013
LRC [40]	4.3.3	16	10	4	5		✓	✗	✗	2013
LRC [41]	4.3.4	10	6	3	3		✓	✗	✗	2012
LRC [58]	4.3.5	n	k	$n-k-\lceil\frac{k}{r}\rceil+1$	r		✗[b]	✗	✗	2014
LRC [59]	4.3.6	n	k	$n-k-\lceil\frac{k}{r}\rceil+1$	r[c]		✗[d]	✗	✗	2016
MBR [55]	4.4.2	n	k			$[k,n-1]$	✓	✓	✗	2011
MSR [51]	4.4.3	n	k			$[2k-2,n-1]$	✓	✓	✗	2015
MSR [60]	4.4.4	$k+2$	k	2		$k+1$	✓	✓	✓	2016

[a] A more general construction exists, which anyway improves information locality only. For simplicity, we divide the k data blocks into τ groups with $\tau \mid k$, create 1 redundant block per group and leave unchanged $n-k-1$ global parities. Then, the information locality is $\frac{k}{\tau}$.

[b] The authors offer a systematic version of the LRC code, for which the optimality of the minimum distance is generally not guaranteed.

[c] It must be $r+1 \mid n$.

[d] The generator matrix described in [59] is not in systematic form. Nevertheless, the authors describe a method to make it systematic, by preserving the locality and distance properties.

Table 4.1: State-of-the-art codes summary.

traffic gives rise to new design challenges [53]. Thus, besides the traditional metrics, three major repair cost metrics have been identified in [56]:

4. *Disk I/O*, i.e., the number of bits read during the repair;
5. *Repair bandwidth*, i.e, the number of bits communicated in the network during the repair;
6. *Repair locality*, i.e., the number of nodes that participate in the repair process.

We now describe these new metrics in more detail.

4.2.1 Disk I/O

The disk read overhead during a repair using a Reed-Solomon code of dimension k is k times that under replication. Indeed, to reconstruct a single block of size b, k blocks of size b are read from the nodes participating to the repair [51]. Consequently, under RS codes, repair requires a large amount of disk I/O, rising the need for lighter techniques.

4.2.2 Repair bandwidth

When a node fails, storage systems must set up a replacement node using information from the functioning nodes. Hence, a key question is how to restore the information in the replacement node while transferring as little data as possible across the network. If a RS(k,$n-k$) code is used, for every block on the failed node, k blocks must be downloaded from surviving ones and combined by

the replacement node to reconstruct the lost coded block. Downloading a full block from each of k surviving nodes is suboptimal [46]. Indeed, data reconstruction can be performed by downloading functions of the data stored in the surviving nodes.

To redeem Reed-Solomon codes, we cite a very recent work by Guruswami, showing that RS codes can also take advantage of the freedom to download partial symbols for the exact repair problem. The exact repair problem for high-rate $\text{RS}(k,n-k)$ codes can be solved with repair bandwidth of $O(n)$ bits [54].

4.2.3 Repair locality

Repair locality deals with the repair of single node failures, i.e., with the reconstruction of a single erased block in a codeword [40]. The repair locality of a code is the maximum over the blocks locality [61]. The *block locality* of a block b is the smallest integer r such that b is a linear combination of exactly r other blocks. In other words, the block is said to have locality r if it can be recovered by accessing at most r other blocks [61]. The block locality, and thus the repair locality, ranges from 1, if the block is replicated, to k as this is the number of source blocks. For repair efficiency, the lower the better.

A code with repair locality r has at least one block having block locality exactly r and every block is a linear combination of at most r other blocks. When it is clear from the context, we shall write locality for both block locality and repair locality. If we deal with the locality of the code we refer to repair locality, while block locality refers to the locality of a block of a representative codeword. Locality can be weakened to protect only data blocks: the *information locality* of a code is the maximum over all the data block localities. Moreover, we refer to a code offering the same block locality r for every block as a code with *all-blocks locality* r. For example, MDS codes of dimension k have all-blocks locality k, which is the highest, thus the worst, possible.

We now give operational definitions of block locality. Let $g_1, \ldots, g_n$ be the columns of a generator matrix G for a certain code. We say that the block in coordinate $1 \leq i \leq n$ has locality r, if the column g_i can be written as linear combination of r other columns of G (and not less than r). Equivalently, for any coordinate $1 \leq i \leq n$, there exists a row in the parity-check matrix of the code of Hamming weight at most $r+1$, whose support includes i [62]. The definition of code locality can be relaxed as in [56] to include both linear and non-linear codes and to allow the size of the input and output blocks to be different.

4.3 Codes for repair locality

Codes that aim at as-small-as-possible repair locality go by the name of locally repairable [56], locally reconstructible [41], locally recoverable [58] and self-repairing codes [63]. In the following we summarize what they are, what they target and some state-of-the-art constructions.

A locally repairable code (LRC) of length n, dimension k and locality r can recover any coded symbol by accessing and processing at most r symbols [59]. The LRC is optimal if it reaches the bound [61]

$$d \leq n - k - \left\lceil \frac{k}{r} \right\rceil + 2. \tag{4.1}$$

Optimal LRCs exist when $r+1 \mid n$ [56].

A structure theorem for optimal codes when $r \mid k$ is given in [61], but explicit codes are constructed only in the case $d = 4$. In the following, we review explicit constructions of codes targeting low repair locality and offering a wide spectrum of parameters.

4.3.1 Locally repairable codes

A locally repairable code of length n, (all-symbols) locality r, minimum distance d, data object size M and size of the encoded symbols α is defined in [56] as a code that encodes the data object of size M bits in n coded symbols of size α bits such that:

- Any of these n coded symbols can be reconstructed by accessing and processing at most r other symbols;
- The data object of size M can be reconstructed by accessing any $n-d+1$ of the n coded symbols.

On top of a MDS code of length n and dimension k, we can build an optimal locally repairable code (LRC) of all-symbols locality r as follows [56]. We divide a data object X of size $M = rk$ symbols in r parts $X = [x^1, \ldots, x^r]$, with each x^i for $1 \le i \le r$ having size k. We encode each of the r chunks independently, into coded vectors $y^i = x^i G$ of length n, using the MDS code. Finally, we generate a single parity XOR vector from all the coded vectors: $s = \bigoplus_{i=1}^{r} y^i$. The total $(r+1)n$ blocks are placed in n nodes, $r+1$ blocks per node, in circular fashion, requiring storage capacity $\alpha = \frac{r+1}{r}\frac{M}{k} = r+1$ [56]. The code retains the minimum distance $d = n-k+1$ of the MDS code, but has rate $\frac{r}{r+1}\frac{k}{n}$. When $r+1 \nmid k$, its minimum distance meets the optimal bound (4.1).

4.3.2 Pyramid codes

We recall the construction for basic pyramid codes introduced in [57]. Pyramid codes are built on top of an embedded MDS code in systematic form and retain the distance of the inner MDS code, thus they are not MDS. Nevertheless, they offer better information locality than a MDS code with same length and dimension.

Let e_i be the vector having 1 in coordinate i and 0 elsewhere. Let

$$G = \begin{pmatrix} e_1 & \ldots & e_k & g_{k+1} & g_{k+2} & \ldots & g_n \end{pmatrix}$$

be the generator matrix of a systematic MDS code. To build the pyramid code, one of the parity blocks of the MDS code is substituted with τ blocks. For example, we substitute the first MDS parity with τ blocks. First, the set $\{1, \ldots, k\}$ is partitioned into τ disjoint subsets $S_1, \ldots, S_\tau$. Then, the following blocks

$$\Big(\sum_{i \in S_1} g_{i,k+1} x_i, \sum_{i \in S_2} g_{i,k+1} x_i, \ldots, \sum_{i \in S_\tau} g_{i,k+1} x_i \Big)$$

are put in place of the first parity block of the MDS code. Hence, a generator matrix for the pyramid code reads as

$$\begin{pmatrix} e_1 & \ldots & e_k & g_{k+1}|_{S_1} & \ldots & g_{k+1}|_{S_\tau} & g_{k+2} \ldots & g_n \end{pmatrix}$$

which implies that the code has length $n - 1 + \tau$, retains the minimum distance of the MDS code and has information locality $r = \max_{1 \le i \le \tau} |S_i|$.

4.3.3 Xorbas-LRC

The authors of [40] prove that a locally repairable code of length n, dimension k, logarithmic block locality $r = \log k$ and distance $d_{\text{LRC}} = n - (1 + \delta)k + 1$ exists, where $\delta = \frac{1}{\log(k)} - \frac{1}{k}$. Hence, any subset of $k(1 + \delta)$ blocks can be used to reconstruct the original data object.

An explicit construction, XORBAS(10,6), implemented in HDFS-Xorbas, is given for $k = 10$, $n = 16$ and $r = 5$ [40]. The XORBAS(10,6) code has four parity blocks constructed with a standard RS(10,4) code and 2 local parities providing efficient repair in the case of single block failure. The authors show that the code is optimal, i.e., it has the largest possible minimum distance $d = 5$ for given locality $r = 5$ and length $n = 16$. Although the code is not in systematic form, the authors of [40] provide a linear transformation that renders the code systematic, while retaining its distance and locality properties. With a Markov chain based analysis, see Section 3, the authors show that, in terms of reliability, the higher repair speed of XORBAS(10,6) with respect to RS(10,4) compensates for the additional storage overhead [40].

4.3.4 Local reconstruction codes

With the goal of reducing the reconstruction cost, Local Reconstruction Codes are proposed in [41]. In particular, k data fragments are divided into l groups and a local parity is computed for each group. In addition p global parities are computed for a total length $n = k + l + p$. For a linear code of length n and dimension k to:

- Repair a single erased block using $\frac{k}{l}$ other blocks;
- Repair up to arbitrary $p + 1$ erased block;

it is necessary that $n - k \ge l + p$ [41, Theorem 1]. The l coding equations, that is the equations that dictate how the parities are computed from the data blocks, are determined so that the LRC achieves the Maximally Recoverable (MR) property (the code does decode any failure pattern which is information-theoretically decodable [57]).

In Windows Azure Storage, the WAS(6,4) code is used, which takes in input 6 data blocks and divides them in two groups. We call x_0, x_1, x_2 the data blocks of the first group and y_0, y_1, y_2 the data blocks of the second group. The WAS(6,4) code sets 2 local parities p_x, p_y and 2 global parities p_0, p_1, according to the coding equations

$$
\begin{aligned}
p_x &= x_0 + x_1 + x_2 \\
p_y &= y_0 + y_1 + y_2 \\
p_0 &= \alpha_0 x_0 + \alpha_1 x_1 + \alpha_2 x_2 + \beta_0 y_0 + \beta_1 y_1 + \beta_2 y_2 \\
p_1 &= \alpha_0^2 x_0 + \alpha_1^2 x_1 + \alpha_2^2 x_2 + \beta_0^2 y_0 + \beta_1^2 y_1 + \beta_2^2 y_2
\end{aligned}
$$

with $\alpha_i, \beta_s \neq 0$, $\alpha_i \neq \beta_s$ and $\alpha_i + \alpha_j \neq \beta_s + \beta_t$. The WAS(6,4) code repairs any 3 failures and also 86% of 4 failures, achieving the MR property. With a Markov

chain based analysis, the authors show that WAS(6,4) is more reliable than a RS(6,3), which is in turn more reliable than 3-replication [57].

4.3.5 Tamo and Barg optimal locally recoverable codes

We describe Tamo and Barg construction for a LRC code of length n, dimension k and locality r [58]. Let $\mathbb{F}_q$ be a finite field with $q > n$ elements. The idea of this construction is to find a partition $\mathcal{A}$ of $A \subset \mathbb{F}_q$ of size $|A| = n$, together with a good polynomial $g(x)$ for $\mathcal{A}$, see [58, Theorem 3.2 and 3.3].

The code is defined as an evaluation code: $\mathscr{C} = \left\{\left(f_a(\alpha), \alpha \in A\right) : a \in \mathbb{F}_{q^k}\right\}$ where $f_a(x)$ is derived from the good polynomial $g(x)$ and $a \in \mathbb{F}_{q^k}$. In turn, recovery of one erased symbol can be done by interpolation over r known symbols. Thus the locality is r and the minimum distance $d = n - k - \lceil \frac{k}{r} \rceil + 2$ is optimal.

4.3.6 Optimal locally repairable codes based on RS codes

An explicit family of optimal locally repairable codes based on Reed-Solomon codes is introduced in [59]. The construction is optimal for any length n, dimension k and locality r such that $r + 1 \mid n$.

Let $m = n\frac{r}{r+1}$ and $1 < k < r$. Take the output of a RS(k,$m-k$) code, divide it into $\frac{m}{r}$ non-overlapping groups, each consisting of r coded symbols. Encode the symbols of each group into $r + 1$ new symbols using a specific RS(r,1). The $k \times n$ generator matrix of the LRC code is $G = V \cdot (I \otimes A)$ where:

- V is a $k \times m$ Vandermonde matrix with the i^{th} column being equal to $\bar{v}_i = (1, \alpha_i, \ldots, \alpha_i^{k-1})^t$, where $\{\alpha_1, \ldots, \alpha_m\} = \mathbb{F}_p$;
- $I = I_{\frac{m}{r}}$ is the $\frac{m}{r} \times \frac{m}{r}$ identity matrix;
- A is a $r \times (r+1)$ matrix having 1 on the main diagonal and w on the lower diagonal, where w is a primitive element of $\mathbb{F}_{p^{k+1}}$. A serves as generator matrix for the RS(r,1) and it provides the locality property of the code.

The locally repairable code generated by G has locality r and optimal minimum distance $d = n - k - \lceil \frac{k}{r} \rceil + 2$ [59].

4.3.7 Homomorphic self-repairing codes

Let $o \in \mathbb{F}_{q^M}$ be an object of size M to be stored over a network of n nodes. Let k be a positive integer such that $k \mid M$. We can write $o = (o_1, \ldots, o_k)$ with $o_i \in \mathbb{F}_{q^{\frac{M}{k}}}$. A homomorphic self-repairing code HSRC(k,$n-k$) [63] encodes $o = (o_1, \ldots, o_k)$ as $(p(\alpha_1), \ldots, p(\alpha_n))$ where

$$p(x) = \sum_{i=0}^{k-1} o_i x^{q^i} \in \mathbb{F}_{q^{\frac{M}{k}}}[x]$$

and $\alpha_1, \ldots, \alpha_n \in \mathbb{F}_{q^{\frac{M}{k}}}$. The parameters of the code must satisfy $k < n \leq q^{\frac{M}{k}} - 1$.

Encoding is done via suitable polynomial evaluation and decoding requires Lagrange interpolation. Self-repair, i.e., the reconstruction of an encoded block,

is performed as the XOR of some other encoded blocks and satisfies two prominent properties: (1) encoded blocks can be repaired using other encoded blocks without reconstructing the original data first, (2) a block is repaired from a fixed number of encoded blocks, the number depending only on the number of erased blocks.

4.4 Codes for repair bandwidth

In a storage system using erasure coding, the reconstruction operations can exacerbate the system resources. When a node fails, for its information to be refreshed, the system needs to transfer data across the network. The bigger the amount of data, the more bandwidth is used. In this section, we present state-of-the-art coding schemes that perform repair operations minimising the amount of data transferred. To this aim, we first introduce the setting of the seminal paper of Dimakis et al. [46], then we summarize the product-matrix framework, which has successfully contributed to the construction of explicit coding schemes.

A data object of size $\mathcal{M}$ is divided in k original blocks and encoded into n coded blocks, which are then distributed to n storage nodes, each storing α bits. Whenever a node fails, a replacement node downloads from any d surviving nodes $\beta \leq \alpha$ bits each, where d is the number of helper nodes. The total amount $\gamma = \mathsf{d}\beta$ of data downloaded for repair purposes is called *repair bandwidth*. Data is stored across n nodes and by connecting to any k nodes is possible to repair a failed node. The minimality of k implies $\mathsf{d} \geq k$ [55]. If repair is required, then at least one node is failed, then $\mathsf{d} \leq n-1$. Thus, for a code of length n and dimension k we have $k \leq \mathsf{d} \leq n-1$. A code with parameters $(n, k, \mathsf{d}, \alpha, \gamma)$ exists if a code with storage α and repair bandwidth γ exists. For any $\alpha \geq \alpha^*(\mathsf{d}, \gamma)$ the points $(n, k, \mathsf{d}, \alpha, \gamma)$ are feasible, and linear network coding suffice to achieve them, see [46] for the threshold function $\alpha^*(\mathsf{d}, \gamma)$. The codes sitting on the points of this tradeoff are termed *regenerating codes* and two extremal points are distinguished in [46] and deeply investigated by successive works:

- Minimum-storage-regenerating (MSR) codes achieve the best storage efficiency, and the minimum storage point is given by the pair

$$(\alpha_{\mathrm{MSR}}, \gamma_{\mathrm{MSR}}) = \left(\frac{\mathcal{M}}{k}, \frac{\mathcal{M}\mathsf{d}}{k(\mathsf{d}-k+1)} \right). \tag{4.2}$$

  MSR codes have equivalent fault tolerance/storage overhead tradeoff as MDS codes: $\frac{\mathcal{M}}{k}$ bits are stored in each node and any k coded blocks are sufficient to recover the original file. Moreover, MSR codes outperform MDS codes in term of the network repair bandwidth.

- Minimum-bandwidth-regenerating (MBR) codes achieve the minimum repair bandwidth, and the minimum bandwidth point is given by the pair

$$(\alpha_{\mathrm{MBR}}, \gamma_{\mathrm{MBR}}) = \left(\frac{2\mathcal{M}\mathsf{d}}{2k\mathsf{d}-k^2+k}, \frac{2\mathcal{M}\mathsf{d}}{2k\mathsf{d}-k^2+k} \right). \tag{4.3}$$

  Using MBR codes, the storage size α is equal to total number of bits downloaded γ, so there is no bandwidth overhead, just like replication.

A common framework used to build explicit regenerating code schemes is known as the Product-Matrix (PM) framework [55]. Each codeword of a Product-Matrix (PM) code is represented by a code matrix $C \in \mathbf{M}^{n\times\alpha}$ whose i^{th} row c_i^t contains the α symbols stored by the i^{th} node $C = \begin{pmatrix} c_1^t & \dots & c_n^t \end{pmatrix}^t$. Each code matrix is the product $C = \Psi M$ of an encoding matrix $\Psi \in \mathbf{M}^{n\times\mathsf{d}}$ and a message matrix $M \in \mathbf{M}^{\mathsf{d}\times\alpha}$. The rows of the encoding matrix $\Psi = \begin{pmatrix} \psi_1^t & \dots & \psi_n^t \end{pmatrix}^t$ are called encoding vectors. In particular, ψ_i^t is the encoding vector of node i, that is, ψ_i^t is the vector used to encode the message for node i: $c_i^t = \psi_i^t M$.

Some basic guidelines for the choice of the parameters follow. First, for redundancy, k should be sufficiently large for the code to be efficient. Then, n can be chosen for the code to attain the desired availability, just as in the case of traditional erasure codes. More explicitly, we can estimate the unavailability probability as $U(n,k) = \sum_{i=0}^{k-1} \binom{n}{i} a^i (1-a)^i$, where a is the probability that a node is available [46], and select n accordingly. Once the target level of availability for the system is fixed, theoretical papers suggest that the best value for the number of helpers is $\mathsf{d} = n-1$. However, this choice is not optimal as soon as the probability that a node is available is strictly smaller than one [64].

4.4.1 Optimal MSR and MBR PM codes

Optimal MBR(k,$n-k$) codes for any d and MSR(k,$n-k$) for $\mathsf{d} \geq 2k-2$ with $\beta = 1$ are constructed in [55]. Notice that in [65], a proof of non-achievability of the cut-set bound for MSR(k,$n-k$) with $\mathsf{d} < 2k-3$ when $\beta = 1$ is provided. From $\beta = 1$ we have $\gamma = \mathsf{d}$, which substituted into equations (4.2) and (4.3) leads to

$$(\alpha_{\mathrm{MSR}}, M_{\mathrm{MSR}}) = (\mathsf{d}-k+1, k(\mathsf{d}-k+1))\,; \tag{4.4}$$

$$(\alpha_{\mathrm{MBR}}, M_{\mathrm{MBR}}) = \left(\mathsf{d}, \frac{k(2\mathsf{d}-k+1)}{2}\right). \tag{4.5}$$

For such a set of parameters, a construction for exact (a failed node is replaced with a node storing exactly the same data) and systematic (the $\mathcal{M}$ message symbols are explicitly present among the $k\alpha$ code symbols stored in k selected nodes) optimal regenerating codes is given in [55].

Moreover, given an optimal regenerating code of length n, dimension k, d helpers and parameters $(\alpha, \beta, \mathcal{M})$, another optimal regenerating code with parameters $(\delta\alpha, \delta\beta, \delta\mathcal{M})$ for any positive integer δ can be constructed by dividing the δM message symbols into δ groups of $\mathcal{M}$ symbols each and applying the $(\alpha, \beta, \mathcal{M})$ code to each group independently. For MBR and MSR codes, $\frac{\alpha}{\beta}$ and $\frac{\mathcal{M}}{\beta}$ are functions of n, k and d only. It follows that from an optimal MBR or MSR code of length n, dimension k, d helpers with $\beta = 1$, one can construct an optimal MBR or MSR code of length n, dimension k, d helpers for any $\beta > 1$. This process being called data striping [55].

4.4.2 MBR PM Code

We summarize the construction for MBR(k,$n-k$) codes with $\beta = 1$ from [55]. The code parameters must satisfy Equation (4.5). First, we prepare the message.

The $\mathsf{d} \times \mathsf{d}$ message matrix M is defined as the symmetric matrix

$$M = \begin{pmatrix} S & T \\ T^t & 0 \end{pmatrix}. \tag{4.6}$$

The submatrix $S \in \mathbf{M}^{k \times k}$ is built such that

- The $\binom{k+1}{2}$ entries in the upper-triangular half of the matrix are filled up by $\binom{k+1}{2}$ distinct message symbols coming from the set $\{u_i\}_{i=1}^{\mathcal{M}}$;
- The $\binom{k}{2}$ entries in the strictly lower-triangular portion of the matrix are then chosen to make the matrix S symmetric.

The matrix $T \in \mathbf{M}^{k \times (\mathsf{d}-k)}$ is filled up with the remaining message symbols.

To encode we perform the matrix product $C = \Psi M$, and both Cauchy and Vandermonde matrices satisfy the requirements for the encoding matrix Ψ. However, these matrices lead to non-systematic codes. In order to have systematic encoding, the authors of [55] choose

$$\Psi = \begin{pmatrix} I_k & 0 \\ \tilde{\Psi} & \tilde{\Delta} \end{pmatrix}$$

where I_k is the $k \times k$ identity matrix and $[\tilde{\Psi}\ \tilde{\Delta}]$ forms a $(n-k) \times \mathsf{d}$ Cauchy or Vandermonde matrix, and perform encoding with the matrix in Equation (4.6).

This code can repair any failed node by connecting to any d out of the $n-1$ alive node [55, Theorem 2]. Suppose node f fails. Let us call $H = \{h_1, \ldots, h_{\mathsf{d}}\}$ the set of arbitrary d helper nodes and consider $\Psi_{\text{repair}} = \begin{pmatrix} \psi_{h_1}^t & \ldots & \psi_{h_d}^t \end{pmatrix}^t$. Stored in any d out the $n-1$ alive nodes, we find $c_i^t = \psi_i^t M$ for $i \in H$. Then, we compute $c_i^t \psi_f^t = \psi_i^t M \psi_f^t$ for $i \in H$ and get

$$v = \Psi_{\text{repair}} M \psi_f = \begin{pmatrix} \psi_{h_1}^t M \psi_f^t & \ldots & \psi_{h_{\mathsf{d}}}^t M \psi_f^t \end{pmatrix}^t.$$

We retrieve $M\psi_f$ from v through left multiplication by $\Psi_{\text{repair}}^{-1}$, that is, $(M\psi_f) = \Psi_{\text{repair}}^{-1} v$. Equivalently, we can solve the linear system $\Psi_{\text{repair}}(M\psi_f) = v$.

The authors of [64] evaluate the computational costs of the proposed product-matrix code. The non-systematic scheme doubles the encoding costs and quadruples the decoding costs when compared to non-systematic Reed-Solomon codes. Due to the pre-coding step, the systematic product-matrix code is more costly than the non-systematic one. In particular, the product-matrix code in systematic form rises the encoding costs by a factor of 7 when compared to systematic Reed-Solomon codes.

4.4.3 Repair-by-transfer PM codes

The problem of the I/O overhead is not solved by MSR codes, which generally are inferior to RS codes in this regard. This does not mean either that RS are good in I/O: the reconstruction operations of a $\text{RS}(k,n-k)$ result in k times the amount of disk I/O and network transfer compared to replication. Building on the MSR PM code from [55], labelled *vanilla codes* in their systematic version, the I/O overhead problem is addressed in [51].

The authors of [51] identify two properties of a MSR code, satisfied in particular by vanilla codes, that help in transforming the code into a disk-read optimal one, while retaining the storage-reliability-bandwidth optimality. In particular, the I/O optimized version of vanilla codes, called PM-RBT codes, reduces by a factor of about $\mathsf{d} - k + 1$ the number of I/O consumed, for general parameters. An implementation for PM-RBT codes is available at [66].

The reconstruction of vanilla codes goes as follows. Every block $1 \le i \le n$ is assigned a reconstruction vector $g_i = (g_{i1}, \ldots, g_{i\alpha})^t$ of length α and the n vectors $\{g_1, \ldots, g_n\}$ are designed so that any α of them are linearly independent, where α is the stripe-width, that is, the number of symbols per data block.

A block f can be repaired by any set D of d helper nodes hosting one of the remaining $n-1$ blocks of the stripe. Let t_{hfD} be the symbol that a helper node h transfers for the reconstruction of f when the set of helpers is D. For reconstruction of block f, the helper node h computes and transfers the symbols:

$$t_{hfD} = \sum_{j=1}^{D} s_{hj} g_{fj} \tag{4.7}$$

where $\{s_{h1}, \ldots, s_{h\alpha}\}$ are the α symbols stored in node h. Then, the decoding node (the node performing the reconstruction) receives $t_{h_1fD}, \ldots, t_{h_\mathsf{d}fD}$ and use them to reconstruct f. Vanilla codes are not optimal in I/O as most coefficients g_{ij} are non-zero and s_{ij} must be read for every $g_{ij} \neq 0$.

To optimize such codes with respect to I/O the vector g_f in equation (4.7) must be a unit vector and $t_{hfD} \in \{s_{h1}, \ldots, s_{hw}\}$. Then, the helper is said to perform Reconstruction-By-Transfer (RBT), that is, it does not perform any computation and just sends one stored symbol to the decoding node, and it is called a RBT-helper. At a RBT-helper, the amount of data read from the disk is equal to the amount sent through the network. An algorithm to choose RBT-helpers so that the I/O consumed during reconstruction is optimized across the system is presented in [51, Algorithm 2].

4.4.4 Butterfly codes

Butterfly codes are binary systematic $\text{MSR}(k,n-k)$ codes with $\mathsf{d} = n-1$ helpers and redundancy $n-k = 2$ [60]. Given $k \ge 2$, the $\text{BUTTERFLY}(k,2)$ code with $\mathsf{d} = k+1$ can be constructed as follows. The matrix $D_k \in \mathbb{M}^{2^{k-1}\times k}(\mathbb{F}_2)$ represents a data object to be encoded. In D_k we distinguish two matrices $A, X \in \mathbb{M}^{2^{k-2}\times k-1}(\mathbb{F}_2)$, and two column vectors $a, x \in \mathbb{M}^{2^{k-2}\times 1}(\mathbb{F}_2)$. We label the columns as D_k^i for $0 \le i \le k-1$, that is

$$D_k = \begin{pmatrix} a & A \\ x & X \end{pmatrix} = \begin{pmatrix} D_k^{k-1} & \ldots & D_k^0 \end{pmatrix}.$$

The systematic codeword we obtain via encoding is $C_k = (D_k^{k-1}, \ldots, D_k^0, H, B)$ where $H = \mathcal{H}(D_k)$ is the horizontal parity and $B = \mathcal{B}(D_k)$ is the butterfly parity. For $k = 2$,

$$\mathcal{H}\begin{pmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{pmatrix} = \begin{pmatrix} m_{11} \oplus m_{12} \\ m_{21} \oplus m_{22} \end{pmatrix}$$

$$\mathcal{B}\begin{pmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{pmatrix} = \begin{pmatrix} m_{12} \oplus m_{21} \\ m_{11} \oplus m_{12} \oplus m_{22} \end{pmatrix}$$

whereas, for $k > 2$

$$\mathcal{H}(D_k) = \begin{pmatrix} a \oplus \mathcal{H}(A) \\ P_{k-1}[P_{k-1}x \oplus \mathcal{H}(P_{k-1}X)] \end{pmatrix}$$

$$\mathcal{B}(D_k) = \begin{pmatrix} P_{k-1}x \oplus \mathcal{B}(A) \\ P_{k-1}[a \oplus \mathcal{H}(A) \oplus \mathcal{B}(P_{k-1}X)] \end{pmatrix}$$

where P_{k-1} is the $k \times k$ anti-diagonal matrix with counter-diagonal elements set to 1 and all other elements equal to 0.

Butterfly codes can decode the original matrix when any two of the codeword columns are missing, hence they are MDS [60]. For the recovery of a single column the amount of data that is transferred is optimal and equals half of the remaining data. The amount of data read to repair is also optimal, except when we recover the butterfly parity.

Butterfly codes are implemented and tested in HDFS (encoding is a background job) and Ceph (online encoding) demonstrating that the encoding approach (online/background) highly impacts the performance of the system. The real-time approach (Ceph) achieves efficient storage utilization but suffers high storage access overhead due to excessive fragmentation. In Ceph the small message size comes as a consequence of the online encoding which reduces the size of the encoded messages and hence the size of a symbol [60]. The background data encoding (HDFS) achieves better performance but reduces the storage efficiency because the input data is stored before being encoded.

4.4.5 Hitchhiker

Hitchhiker is an erasure code built on top of Reed-Solomon codes with the aim of reducing network traffic and disk I/O during reconstruction without increasing the storage overhead [67]. The construction is based on the Piggybacking framework, which operates on pairs of stripes of a RS code [68].

The Hitchhiker erasure code optimizes only the reconstruction of data blocks: reconstruction of parity blocks is performed as for Reed-Solomon codes. Furthermore, Hitchhiker optimizes data reconstruction for single failures only. If multiple blocks belonging to a single codeword are unavailable, Hitchhiker performs reconstruction as the RS code.

We now summarize the three versions of Hitchhiker, using as example the Hitchhiker code of length 14 and dimension 10, extensively described in [67]. For $1 \le i \le 14$, let b_i be the i-th block of a representative Hitchhiker codeword. Opposite to the 20 blocks used to repair a pair of stripes of the RS(10,4) code, for a pair of codewords:

- Hitchhiker-XOR reconstructs blocks b_i for $1 \le i \le 6$ using 13 blocks, and blocks b_i for $7 \le i \le 10$ using 14 blocks, and requires only XOR operations in addition to the underlying RS reconstruction.
- Hitchhiker-XOR+ recovers any block using any other 13 blocks, but requires the underlying RS code to have the all-XOR-parity property, that is, one of the parity blocks must be the XOR of all the data blocks.
- Hitchhiker-nonXOR recovers any block using any other 13 blocks, at the cost of additional finite-field arithmetic.

The increase in computational time for Hitchhiker due to the extra operations is compensated by the decrease in reconstruction time and by the reduction in the amount of data read and downloaded during repair.

4.5 LDPC codes

Performance and efficiency of erasure coding is crucial in distributed storage systems. Low-Density Parity-Check (LDPC) codes have arisen as an alternative to standard erasure codes, such as Reed-Solomon codes, trading off vastly improved decoding performance for inefficiencies in the amount of data required to perform decoding. For a LDPC code encoding k equal-sized data blocks into n equal-sized coded blocks to be distributed over the network, the decoding requires to download fk coded blocks to calculate the original k data blocks, with $f > 1$. For suitable degree distributions λ and ρ, there are codes that asymptotically achieve capacity, that is, they may be successfully decoded with fk downloaded blocks, where $f \to 1$ from above as $k \to \infty$. But in distributed systems we cannot break data in infinitely many pieces. Nevertheless, research on LDPC codes typically studies their collective and asymptotic behaviour. In the following, we overview established results about finite length LDPC codes.

4.5.1 Analysis of LDPC codes for finite lengths

In [23], three types of "born-to-be-long" codes are analysed to assess their performance for finite lengths, namely n ranges in

$$\{2i : 1 \leq i \leq 75\} \cup \{250, 500, 1250, 2500, 2500, 5000, 12500, 25000, 50000, 125000\}.$$

For rates $\frac{k}{n} \in \{\frac{1}{3}, \frac{1}{2}, \frac{2}{3}\}$, systematic, IRA and Gallagher codes are tested over a collection of 80 published distributions λ and ρ via Monte Carlo simulations. The average number of blocks required to successfully reconstruct the data over 1000 random downloads is reported using the overhead factor f. For example, if $k = 100$ and $n = 200$, $f = 1.10$ means that on average 110 random blocks of the total 200 blocks are required to reconstruct the 100 original blocks of data.

For the three rates, the overhead factor f starts at 1 when $n = 1$, increases with n up to $n = 20$, then it flattens. Finally, f decreases after $n = 100$, see [23, Figure 5]. Opposite, the reconstruction performance in term of f is different among the three types of codes and for small n systematic codes perform the best. However, after $n = 100$, IRA codes outperform the others, see [23, Figure 6].

Moreover, published distributions are inadequate for producing codes with a small overhead factor f for small n. Following the notation in [23], we call *published codes* LDPC codes arising from the list of 80 published distributions for λ and ρ. We call *Monte Carlo codes* the ones generated as follows: first generate random λ and ρ, then determine the ten best pairs minimizing f, finally pick random ρ for the ten best λ and conversely. Distributions for *derived codes* are obtained from the best published and Monte Carlo codes using their left and right nodes quantities to define new λ and ρ. For $n \leq 15$ Monte Carlo codes give the best results and for about $15 \leq n \leq 100$, derived codes perform the best, while after $n = 100$, published and derived codes are roughly equivalent.

A comparison of LDPC against RS codes is presented in [23]: the time for downloading to a client is combined with the time for reconstruction, and used to calculate the average time to download a file of size 1GB. Among LDPC codes, systematic codes outperform IRA codes, which in turn outperform Gallagher codes. As the download speed improves, the rate increases and n increases, LDPC codes greatly outperform RS codes. However, when n is small (about $n \leq 30$ depending on the rate) and the download speed is slow, RS outperform LDPC codes.

4.5.2 Repair bandwidth analysis

The repair bandwidth is defined in the average sense in [22]: if a variable node is erased, the repair bandwidth to refresh that variable node is the number of blocks downloaded, averaged over all choices of neighbouring check nodes (assuming all other variable nodes to be pristine). This value is then averaged over all variable nodes. In particular, let d_i be the degree of the check node i and E the number of edges in the Tanner graph. Then, the (average) repair bandwidth γ is defined as

$$\gamma = \frac{\sum_{i=1}^{n-k} d_i(d_i - 1)}{E}. \tag{4.8}$$

For the average variable/check node degree a_l and a_r (we write l and r when the code is left/right regular), the following statements hold [22]:

- A regular check node degree r minimizes the repair bandwidth, and this minimal value is $\gamma = \mathsf{r} - 1$ [22, Lemma 1];
- A graph yields a LDPC code of rate $\frac{k}{n}$ with minimum repair bandwidth if and only if the graph is both check node regular and variable node regular with $\mathsf{r} = \frac{a_{l_{\min}}}{1-\frac{k}{n}}$ and $\mathsf{l} = a_{l_{\min}}$ [22, Theorem 1]. The authors set $a_{l_{\min}} = 2$ as it must be $a_{l_{\min}} \geq 2$ to reconstruct single erasures.

The repair bandwidth γ of a LDPC code does not depend on n but on the average check node degree a_r. Moreover, γ and a_r increase with $\frac{k}{n}$ for a fixed a_l. Unfortunately, regular LDPC codes are known not to reach the channel capacity [21], thus the repair bandwidth minimality has to be relaxed to improve reliability. In particular the condition on variable nodes is relaxed in [22], preserving the check node regularity, i.e., $\rho(x) = x^{\mathsf{r}-1}$ so that $\gamma_{\min} = \mathsf{r} - 1$. Then, a small value of r is required for repair efficiency and this must be tradedoff for ensuring the desired reliability level. In [22, Table I], we find some left degree distributions designed for the decoding threshold to achieve the BEC capacity. In the best case the designed decoding threshold is 98.4% with respect to the BEC capacity.

4.5.3 Finite-length analysis

A finite-length analysis of LDPC codes over the BEC is given in [24]. A regular ensemble of LDPC codes $\mathcal{C}(n, x^{\mathsf{l}-1}, x^{\mathsf{r}-1})$ is characterized by block length n, variable nodes degree l and check nodes degree r. Let $\mathbb{P}(G, \alpha)$ be the block erasure probability when transmitting over the BEC_α, see equation (1.1), using a code

$G \in \mathcal{C}(n, x^{l-1}, x^{r-1})$ and a message-passing decoder. Then, $\mathbb{E}_{\mathcal{C}(n,x^{l-1},x^{r-1})}[\mathbb{P}(G,\alpha)]$ denotes the ensemble average over all realizations of the channel.

Although the behaviour of a particular code can differ significantly from that of the cycle-free case for moderate block lengths, the behaviour of individual instances is likely to concentrate around the ensemble average. An exact expression for the ensemble average is [24]

$$\mathbb{E}_{\mathcal{C}(n,x^{l-1},x^{r-1})}[\mathbb{P}(G,\alpha)] = \sum_{e=0}^{n\frac{l}{r}-1} \alpha^e (1-\alpha)^{n-e} \Big(1 - \frac{N(e, n\frac{l}{r}, 0)}{T(e)}\Big) + \sum_{e=n\frac{l}{r}}^{n} \alpha^e (1-\alpha)^{n-e}$$

where $T(e) = \frac{(nl)!}{(nl-el)!}$ and $N(v,c,d)$ is defined in [24, Equations (2.1)-(2.4)].

The quantity $1 - \frac{N(e,n\frac{l}{r},0)}{T(e)}$ is the probability that a randomly chosen subset of size e of the variable nodes contains a nonempty stopping set, while the probability that the size of the erasure set is e is given by $\alpha^e(1-\alpha)^{n-e}$. Notice that the probability that a randomly chosen subset of size e of the variable nodes contains a nonempty stopping set goes to 1 for $e \geq n\frac{l}{r}$, that is, more than $n\frac{l}{r}$ erasures cause data loss.

4.5.4 Reliability analysis

The reliability of LDPC codes is studied using the Markov model for non-MDS codes as presented in Section 3.2 in [22], and the unconditional probability for the Markov model is estimated by decoding simulations of the LDPC codes on the binary erasure channel. The closed form solution for the MTTDL in [22, Lemma 2] confirms that increasing the stopping number improves the reliability.

When the left degree is $l = 2$, the stopping number is half the girth, see Section 2.2.2. Then, we must increase the girth, to rise the reliability of a regular LDPC codes with $l = 2$. With this motivating reason, the design of LDPC code with good girth properties using the PEG algorithm [69] is done in [22]. The authors fix $l = 2$ and rate $\frac{k}{n} = \frac{2}{3}$ and generate a few PEG-LDPC codes for those parameters. In [22, Table III], a comparison of the PEG-LDPC codes against RS codes and LRC codes based on storage overhead, repair bandwidth and MTTDL shows that the most reliable LDPC code, LDPC(140,70), has dimension 140 and length 210. The LDPC(140,70) code is more reliable than the RS(10,5) code while halving the repair bandwidth and entailing the same storage overhead. Moreover, the LDPC code with length 60 and dimension 40 has the same reliability as XORBAS(10,6), see Section 4.3.3.

4.5.5 Tornado codes for MAID archival storage

Massive Arrays of Idle Disks (MAID) have been proposed as a low-power and lower-latency replacement for magnetic tapes as a backing store. The Tornado Coded Archival Storage (TCAS) system, developed and implemented in [70], is combined with a MAID simulator to emulate a large scale storage system. The TCAS file-system breaks files into extents and aggregates the extents into fixed-size stripes. When a stripe is full or a timeout expires, the entire stripe is encoded and distributed to storage devices. Each stripe consists of 48 data blocks and 48 redundant blocks of size 1 MB.

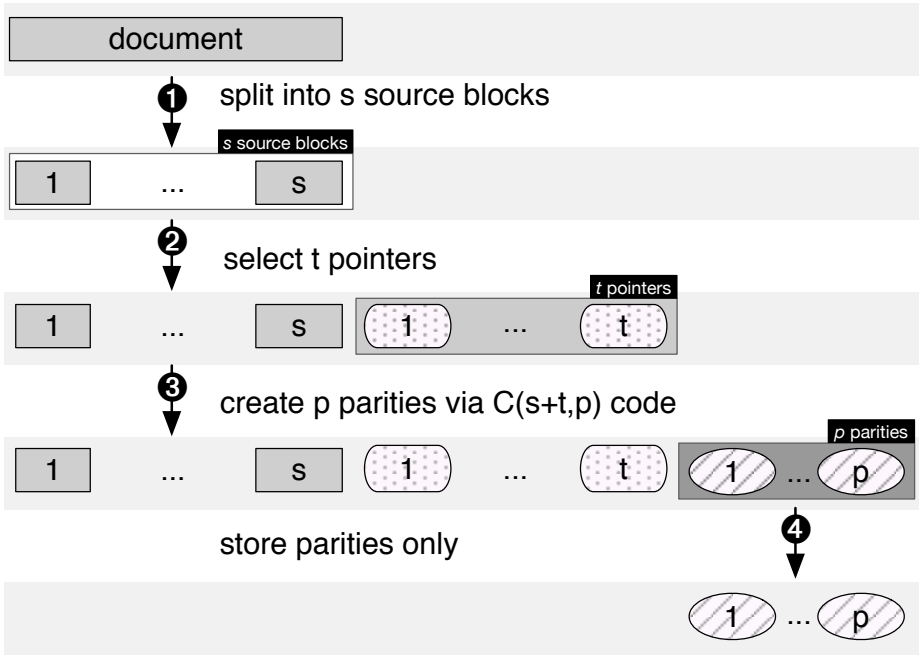


Figure 4.1: Entanglement logical flow.

The TCAS system is deployed on a 12 storage servers cluster and because the Tornado code has length 96, a cyclic 96:12 mapping is used to associate coded blocks with devices. TCAS designers test three Tornado codes with the above parameters in high availability configuration and show that the three codes survive the loss of any three storage servers [70, Figure 8].

4.6 STeP-archival

In this section we revisit STeP-archival, the technique presented in [71] for creating interdependencies between data to be stored and data already archived in the system. Entanglement allows to enhance desirable properties for the storage system, namely anti-censorship, see Chapter 7 and reliability, see Chapter 8. Upon archival, the blocks of a document are entangled with some blocks of documents previously archived in the system. The entanglement builds strong ties between content, hindering data loss and preventing silent censorship of rarely accessed data. In particular, an attacker wishing to censor a target document must cause collateral damage by corrupting several other archived documents.

More formally, a (s,t,e,p)-archive [71] is a storage system where each archived document consists of a codeword of s source blocks, t pointers or tangled blocks (i.e., old blocks already archived in the system), p parity blocks, and that can correct $e = p - s$ erasures per codeword. The logical flow to archive a document in a (s,t,e,p)-archive is illustrated in Figure 4.1. The document to be archived is split into $s \geq 1$ source blocks (solid fill) ❶. From the archive, t distinct old blocks, the *pointers* (dot pattern), are selected ❷ and a linear code $\mathscr{C}(s+t,p)$ is used to create $p \geq s$ parity blocks (stripe pattern) ❸ depending on both source

blocks and pointers. In particular, if the p parity blocks are computed using a $\text{RS}(s+t,p)$ code, then any $s+t$ blocks of the codeword are necessary and sufficient to reconstruct all its $s+t+p$ blocks, see Section 2.1. In the last step ❹, we only archive the p parity blocks, making the code non-systematic. The t pointer blocks come from the archive so we do not need to store them again. The choice of not storing the source blocks enhances security [71]. As a tradeoff, read performance is degraded both in terms of latency and bandwidth since reading a document from a STeP-archive always requires a decoding operation involving $s+t$ blocks, as opposed to a systematic code where the s alive source blocks can be directly accessed.

4.6.1 STeP-archive asymmetry

For an archived document to be lost, it is not sufficient to destroy its code blocks because they can be recovered recursively. On the one hand, it has been proven [71] that finding the minimal set of documents required to irrecoverably censor a target document is NP-hard. On the other hand, the system can repair a recoverable loss using a simple and efficient reconstruction algorithm: we first scan the archive and build a set C of corrupted document with at most e erased blocks. We pick a document from C, repair it, and update the set of documents with at most e erased blocks in C. The algorithm stops when C is empty. At this point, either the system is completely repaired or there is a closed set of documents with strictly more than e erased blocks.

4.6.2 Practical resilience to censorship

Since censoring a document optimally is NP-hard, to evaluate the resistance of the system against data loss, one can rely on suboptimal heuristics. In Chapter 7, we exploit the greedy attacks as well as a branch and bound technique introduced in [71]. We include a brief summary and refer the reader to [71] for more details.

We play the attacker and we select a target to be censored. Erasing blocks in the target produces erasures in other documents because of entanglement. To prevent recursive repair we must corrupt those documents as well. So at each step of the attack we face the choice of which block to delete and we use one of two greedy heuristics to make the decision:

- The *leaping attack* leverages the fact that it is easier to attack recent documents rather than old ones (because they have fewer incoming pointers).
- The *creeping attack* tries to keep the set of corrupted documents as compact in time as possible by preferring documents having approximately the same archival date, which is very effective against window-constrained entanglement strategies.

It is possible to improve on these greedy heuristics by means of a branch and bound technique: at each step of the computation of the minimal set of blocks to be erased, we retain the best partial solutions up a certain buffer size and expand all of them.

Chapter 5

Worst-case, information and all-blocks locality in distributed storage systems: An explicit comparison

5.1 Introduction

The easiest way to provide reliability in distributed storage systems is with data replication. Replication has interesting properties, including its relative simplicity and the fact that a system can repair a failed data object by fetching and copying any of its replicas, which is optimal. Unfortunately, replication entails a significant storage overhead, which can be decreased with erasure correcting codes. With a systematic linear code of length n and dimension k, each codeword consists of n blocks: k source blocks for the original data, and $n-k$ parity blocks. The storage overhead is $\frac{n-k}{k}$, and if the code is MDS, any k of the n blocks are necessary and sufficient to decode the codeword and recover the original data, see Section 2.1. In distributed storage systems the n blocks are typically distributed on different physical disks, thus a MDS code can tolerate up to $n-k$ disk failures, which is optimal. MDS codes, especially Reed-Solomon codes, are commonly used in practice, for instance by Facebook [40, 72].

Distributed storage systems execute periodic repair procedures during which one of the most frequent error occurrence is a single failed block per codeword. Unfortunately, MDS codes were not designed for this purpose and have the prohibitive drawback that k blocks must be fetched to repair a single failure. A lot of research has therefore been recently dedicated to improve the repair locality and repair bandwidth of state-of-the-art codes, and to design new codes tailored for storage. Locally Repairable Codes (LRC), introduced in Section 4.3, can repair a small number of block erasures with less than k blocks at the price of an increased storage overhead, offering a tradeoff between MDS codes and replication. Codes with good locality are now used by major cloud operators such as Microsoft in their Azure storage service [41] and virtualisation platform [73].

Despite the recent advancements of codes with good locality properties, there

is a big gap between theory and practice, in the sense that most LRCs cannot be realistically implemented in practical systems due to performance shortcomings such as high encoding/decoding/repair complexity and latency. Moreover, *information locality* and *locality* are sometimes used interchangeably, while it should be clear that whenever a code only offers information locality, there are stored blocks that do not benefit from the property.

We select the Hamming Code (HAM) as representative of codes with *all-blocks locality*: HAM can repair *every* block with three other blocks, which is optimal for codes with length 7 and dimension 4. Furthermore, HAM is highly efficient on modern architectures because it encodes, decodes and repairs using only hardware-implemented XOR operations. In this chapter, we provide:

- An efficient way to recover all theoretically-recoverable failures for HAM;
- A comparison of HAM against two systematic linear coding schemes with same storage overhead and different locality properties.

We describe encoding, decoding and repair algorithms for HAM, and formally study its locality and fault tolerance. We then evaluate our prototype with both synthetic failures and real-world traces against two codes with the same storage overhead: a RS(4,3) and a pyramid code of length 7 and dimension 4 with information locality 2 built from a RS(4,2), see Section 4.3.2. Comparing for the same level of storage overhead implies that the two non-MDS have lower fault tolerance. Nevertheless the three schemes can withstand as many losses as triplication does, while entailing a storage overhead of 75%. We show that HAM provides faster encoding, faster repair and lower repair bandwidth than the other two codes and we discuss how this improves the fault tolerance by estimating the mean time to data loss. HAM decreases the repair bandwidth by up to 26%, and repairs 5 times faster.

The remainder of this chapter is organized as follows. The existing codes used for comparison are presented in Section 5.2. Fault tolerance and repair procedure are analysed in Sections 5.3 and 5.4 respectively. Finally, we evaluate the performance of our implementation in Section 5.5 and conclude in Section 5.6.

5.2 Linear codes for comparison

For the definitions of block locality, code locality, information locality and all-blocks locality, we refer to Section 4.2.3. We write $(k, n-k, r)$ to denote a code of length n, dimension k with locality $r \leq k$ and $(k, n-k, \langle r \rangle)$ for a code of length n, dimension k with information locality r.

We now introduce three explicit linear codes for the three types of locality: a Reed-Solomon code for *worst-case locality*, a recently proposed pyramid code for *information locality* and HAM an optimal locally repairable code for *all-blocks locality*. We refer to Section 2.1 for an introduction on Reed-Solomon codes and to Section 4.3.2 for the locality definitions. We operate over $\mathbb{F}_{2^8}$: addition over $\mathbb{F}_{2^8}$ is simply the bitwise XOR $\oplus$ and we perform the modular reduction [15] using the primitive polynomial $x^8 + x^4 + x^3 + x^2 + 1$, whose binary representation is 100011101 [74].

5.2.1 Worst-case locality: RS(4,3)

We consider a systematic Reed-Solomon code RS(4,3) of length 7 and dimension 4 over $\mathbb{F}_{2^8}$ based on a Hankel matrix [75]. To build a Hankel matrix we define $\beta_i \triangleq \frac{1}{1-\alpha^i}$ where α is a primitive element of $\mathbb{F}_{2^8}$. We extract a rectangular submatrix H of size $k \times (n-k)$ from the triangle

$$\begin{array}{cccccc} \beta_1 & \beta_2 & \beta_3 & \beta_4 & \cdots & \beta_{255} \\ \beta_2 & \beta_3 & \beta_4 & \cdots & \beta_{255} & \\ \vdots & & & & & \\ \beta_{254} & \beta_{255} & & & & \\ \beta_{255} & & & & & \end{array}$$

and build the Hankel-form generator matrix $G = (I_k \mid H)$ where I_k is the $k \times k$ identity matrix. The RS(4,3) code is thus generated by

$$G_{\text{RS}(4,3)} = \begin{bmatrix} 1 & 0 & 0 & 0 & 244 & 167 & 157 \\ 0 & 1 & 0 & 0 & 167 & 157 & 114 \\ 0 & 0 & 1 & 0 & 157 & 114 & 237 \\ 0 & 0 & 0 & 1 & 114 & 237 & 95 \end{bmatrix}. \tag{5.1}$$

5.2.2 Information locality: Pyr$(4, 3, \langle 2 \rangle)$

Pyramid codes are built on top of an embedded MDS code, as explained in Section 4.3.2. In particular, we build a pyramid code Pyr$(4, 3, \langle 2 \rangle)$ of length 7, dimension 4 and information locality 2 on top of a RS(4,2) over $\mathbb{F}_{2^8}$ whose generator matrix in Hankel form is derived from the first six columns of $G_{\text{RS}(4,3)}$ in (5.1) *partitioning* the sixth. The generator matrix of our Pyr$(4, 3, \langle 2 \rangle)$ pyramid code is therefore given by the first 5 columns of (5.1) and columns 6 and 7 are respectively $(167, 157, 0, 0)$ and $(0, 0, 114, 237)$.

The pyramid code retains the minimum distance of the embedded RS code and hence it is not MDS. However, the extra redundancy improves the information locality: for a codeword $(b_0, b_1, \ldots, b_6)$, we have $b_5 = 167b_0 + 157b_1$ and $b_6 = 114b_2 + 237b_3$. Notice that block b_4 has locality 4; indeed the pyramid code offers only *information locality*.

5.2.3 All-blocks locality: Ham

We now present Ham and describe its properties. The Hamming code Ham is an optimal systematic locally repairable code of length 7, dimension 4 and locality 3. Its choice is based on two main reasons:

- The supports of the codewords of weight 4 of the Extended Hamming code form a 3-(8,4,1) design [13] and such a design can be exploited for repairing Ham, as seen in Section 5.4.
- Non-trivial binary MDS codes (that is codes with $k \neq 1, n-1$ reaching the Singleton bound $d \leq n-k+1$) over $\mathbb{F}_2$ do not exist [76]. Hence, codes optimal with respect to the generalized bound (4.1), i.e., $d \leq n-k-\left\lceil \frac{k}{r} \right\rceil + 2$, should be targeted.

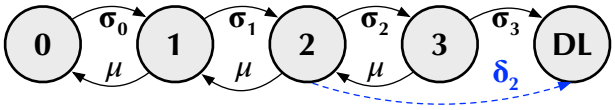


Figure 5.1: Markov chain. States correspond to the number of erasures and DL represents the data loss state. The blue dotted arrow applies to HAM and PYR$(4,3,\langle 2\rangle)$, i.e., $\delta_2 = 0$ for RS(4,3).

We end up at the Hamming code starting from a 4×4 identity matrix (for the code to be systematic) and then adding three different parity columns with three 1s each to ensure locality 3, while paying attention to place only two overlapping 1s on each row to maximize the minimum distance,

$$G_{\text{HAM}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

Lemma 1. HAM *has minimum distance* $d = 3$ *and locality* $r = 3$*, which is optimal.*

Proof. The minimum distance of the Hamming code is well-known [13]. For the locality, we can easily see that each parity block is the sum of exactly three source blocks, and that each source block is the sum of exactly one parity block and two other source blocks. These parameters are optimal because they achieve the bound $d \leq n - k - \left\lceil \frac{k}{r} \right\rceil + 2$ with equality, see Section 4.3. □

HAM has the additional property that it does not require multiplications over finite fields since the elements of G_{HAM} are 0s and 1s. This makes HAM easily amenable to fast implementations in hardware using only XOR operations. Moreover, HAM can work on bytes, being the XOR operation defined on $\mathbb{F}_{2^8}$, without any need for finite field arithmetic. The HAM parities can be computed by XORing the appropriate message bytes, thus HAM can process blocks of the same size as the RS(4,3) and PYR$(4,3,\langle 2\rangle)$ codes. Note that on modern storage systems, the block size is much larger than the field size.

5.3 Fault tolerance analysis

We use the mean time to data loss (MTTDL) to compare the fault tolerance offered by the three schemes. We estimate the MTTDL using a Markov model, as explained in Chapter 3.

We compute the fraction $r(j)$ of failures of size j that can be corrected by the three codes: $r(j) = 0$ for all j greater or equal to the minimum distance of the code. RS(4,3) is MDS, thus its j-erasures properties are $r(0) = r(1) = r(2) = r(3) = 1$.

PYR$(4,3,\langle 2\rangle)$ is built on top of RS(4,2), thus it can correct every 2-erasures, i.e., $r(0) = r(1) = r(2) = 1$. By exhaustive search, we also find that the pyramid code can correct 26 out of the $\binom{7}{3} = 35$ 3-erasures, thus $r(3) = \frac{26}{35} = 74.29\%$.

The minimum distance of HAM is 3, thus $r(0) = r(1) = r(2) = 1$, and like the pyramid code it can correct some 3-erasures. To compute the number of repairable 3-erasures, consider a codeword with blocks

$$(b_0, b_1, b_2, b_3, b_4 = b_1 \oplus b_2 \oplus b_3, b_5 = b_0 \oplus b_2 \oplus b_3, b_6 = b_0 \oplus b_1 \oplus b_3).$$

Block b_0 appears in b_0, b_5, b_6, thus the 3-erasures in positions $\{0, 5, 6\}$ are irrecoverable. The same is true for $\{1, 4, 6\}$ and $\{2, 4, 5\}$. Then notice that $b_0 \oplus b_4 = b_1 \oplus b_5 = b_2 \oplus b_6$ and they all contain b_3, hence the erasure patterns $\{0, 3, 4\}$, $\{1, 3, 5\}$, $\{2, 3, 6\}$ are also irrecoverable. All the other 3-erasures are repairable, thus $r(3) = \frac{35-7}{35} = 80\%$.

Knowing the j-erasures properties, we can define p_j, the probability that the coding scheme can tolerate one more failure, given that j failures have already occurred: $p_j = \frac{r(j+1)}{r(j)}$ where $r(j)$ is the unconditional probability that the code can tolerate j failures as defined above. We compute $p \triangleq (p_0, p_1, p_2)$ for the three codes. We get $p = (1, 1, 1)$ for RS(4,3), $p = (1, 1, 0.74286)$ for PYR$(4, 3, \langle 2 \rangle)$ and $p = (1, 1, 0.8)$ for HAM.

The MTTDL takes into account the maintenance offered by the repair process by means of the drive rebuild rate μ. Also, we call λ the drive failure rate (we set $\lambda = \frac{1}{500000}$ hours [42]).

We consider a Markov chain with states corresponding to the number of erasures, with transition rates σ_j for $j = \{0, 1, 2\}$ given by $\sigma_j = (n - j)\lambda p_j$, see Section 3.2, and $\sigma_3 = (n - 3)\lambda$, where n is the length of the code. For the two non-MDS codes $\delta_2 = (n - 2)\lambda(1 - p_2)$, whereas $\delta_2 = 0$ for RS(4,3). The Markov chain is shown in Figure 5.1. The transition matrix for the Markov model is given by

$$P = \begin{pmatrix} -\sigma_0 & \sigma_0 & 0 & 0 & 0 \\ \mu & -(\mu + \sigma_1) & \sigma_1 & 0 & 0 \\ 0 & \mu & -(\mu + \sigma_2 + \delta_2) & \sigma_2 & \delta_2 \\ 0 & 0 & \mu & -(\mu + \sigma_3) & \sigma_3 \end{pmatrix}$$

where the last column represents the absorbing state corresponding to data loss and each row represents a non-absorbing state. We use $\hat{P}$, the negative of P deprived of the last column, to compute MTTDL per stripe as the first component y_1 of the solution of the linear system $\hat{P}\mathbf{y} = \mathbf{1}$. We postpone explicit results to Section 5.5, as we will use our simulation to get estimates for the rebuild rate μ.

5.4 Repair algorithms

This section describes the repair mechanisms implemented by the three codes. The repair method takes as input alive blocks of a codeword and outputs the codeword itself. Given enough alive blocks, RS(4,3) and PYR$(4, 3, \langle 2 \rangle)$ compute missing blocks in two steps: first the original message is retrieved and then it is re-encoded to get all blocks of the codeword. HAM needs one step only as it repairs erased blocks directly.

RS(4,3) Whenever the number of erasures is at most 3, this code fetches four valid blocks and apply Gaussian elimination on a 4×4 system.

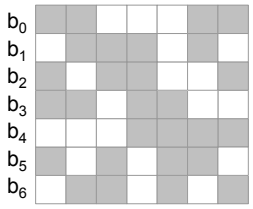


Figure 5.2: Relationships between the blocks of a HAM codeword. Each column corresponds to a parity equation involving the blocks in grey.

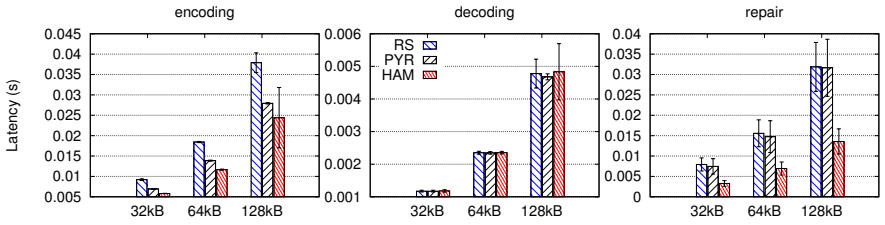


Figure 5.3: Latency of encoding, decoding and repair.

PYR$(4, 3, \langle 2 \rangle)$ The repair procedure for the pyramid code implies two steps. First, it attempts to recover data blocks by linear combination. Second, if the required redundant blocks are not available, the code falls back on a RS-like repair procedure, in which case there is no performance gain compared to the embedded RS code.

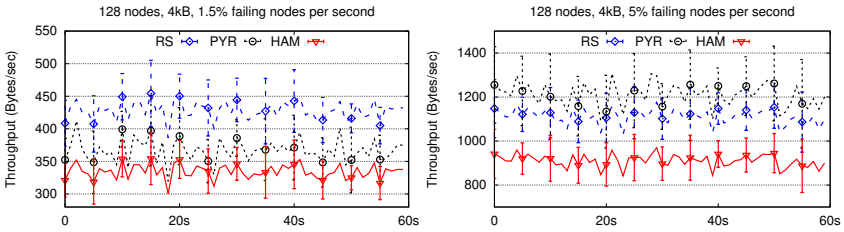
HAM Figure 5.2 depicts the relationships between blocks of HAM codewords. For each column of the figure, the XOR of the grey blocks is 0, thus three grey blocks are sufficient to infer the fourth. For instance, the first column indicates that $b_0 \oplus b_2 \oplus b_3 \oplus b_5 = 0$. To repair an erased block, we look for a column from Figure 5.2 where the erased block is grey, fetch the other three grey blocks and XOR them to recover the erased one.

5.5 Evaluation

In this section, we experimentally evaluate the performance of HAM. First we provide a set of micro-benchmarks to evaluate the encoding, decoding and repair latency. Then, we evaluate the bandwidth costs under synthetic faults. Finally, we inject a 12-hour trace from a large-scale deployment. We compare the performance of HAM against the RS(4,3) and PYR$(4, 3, \langle 2 \rangle)$ presented in Section 5.2.3. We leverage a full, yet not optimized, prototype of the code implemented in Python.

5.5.1 Micro-benchmark: latency

We begin our evaluation by measuring the latency of the schemes to encode, decode and repair input data of increasing size (from 32 kB to 128 kB). Figure 5.3 presents our results. HAM outperforms RS(4,3) and PYR$(4, 3, \langle 2 \rangle)$ in encoding,



(a) When few erasures occur, both locality and information locality lead benefits in repair.

(b) When the number of erasures is close to the repair capability, information locality is not enough to have an advantage on the MDS code.

Figure 5.4: Bandwidth: number of blocks used by repair.

up to 1.5× and 1.2× respectively. This directly derives from the absence of multiplications in HAM, as opposed to RS(4,3) and PYR$(4,3,\langle 2\rangle)$. The decoding latency results are basically equivalent across the three codes: since they are systematic (with dimension $k = 4$), the decoding operation trivially consists in reading the first 4 blocks of the codeword.

The repair micro-benchmark consists in erasing up to 2 random blocks (or 25% of the codeword) and repairing them as fast as possible. We explain the gap in repair performances among the codes as follows. RS(4,3) systematically solves a 4×4 linear system by Gaussian elimination for every erased block, which is costly ($O(k^3)$ in the dimension of the matrix). PYR$(4,3,\langle 2\rangle)$ exploits its information locality: when there is only one erasure per codeword, PYR$(4,3,\langle 2\rangle)$ fetches 2 blocks for repair (if the failure is not in the 4^{th} position, in such a case PYR$(4,3,\langle 2\rangle)$ fetches 4 blocks). However, if more than one erasure occurs, PYR$(4,3,\langle 2\rangle)$ needs to process at least as many blocks as the Reed-Solomon code. In this case, PYR$(4,3,\langle 2\rangle)$ never outperforms RS(4,3) in terms of latency.

On the other hand, HAM never uses Gaussian elimination by leveraging its XOR-based structure. The repair of one failed block results in 2 XORs. If multiple failures occur, HAM fetches 5 blocks and recovers the whole codeword executing 6 XORs. This leads to a factor of 5× speed-up in latency with respect to RS(4,3) and PYR$(4,3,\langle 2\rangle)$. Throughput results are in the same line as latency results.

We can now complete our fault-tolerance analysis. We set $\mu = 1$ for HAM and $\mu = 5$ for the other two codes and get

	RS(4,3)	PYR$(4,3,\langle 2\rangle)$	HAM
$\text{MTTDL}_{\text{stripe}}$ (hrs)	$4.7645 \cdot 10^{15}$	$3.7031 \cdot 10^{12}$	$1.1904 \cdot 10^{14}$

The fault-tolerance of HAM is lower than the one of RS(4,3) because of its lower minimum distance, but higher than the one of PYR$(4,3,\langle 2\rangle)$ thanks to the repair rate.

5.5.2 Micro-benchmark: repair bandwidth

To evaluate the network bandwidth costs of the three codes, we simulate a cluster of 128 remotely distributed nodes. We make the nodes crash at very

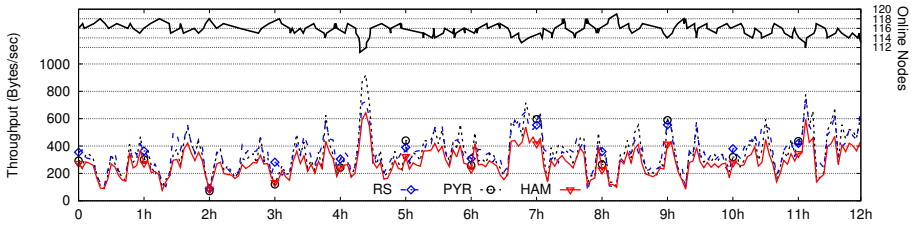


Figure 5.5: Real-world fault trace: network bandwidth required to repair the system during a 12-hour trace.

high rates during a period of 1 minute. Figure 5.4(a) and Figure 5.4(b) present our results for two challenging scenarios, where 1.5% and 5% of the nodes fail every second. We present the mean and standard deviation of the number of blocks fetched during the repair process. HAM consistently fetches fewer blocks, demonstrating the advantage of codes with low locality.

5.5.3 Benchmark: real-world fault trace

We conclude our evaluation by means of a real-world trace [77]. We replay 12 hours of the trace, during which between 112 and 120 simulated nodes are available. The fault dynamic injected in the system is depicted on top of Figure 5.5. In this experiment, the nodes are used to store 2 kB of data. After the 12 hours of the trace, HAM required 11.35 MB to repair the system, as opposed to the 14.33 MB of RS(4,3) (+%26) or 13.97 MB of PYR$(4, 3, \langle 2 \rangle)$ (+%23).

5.6 Summary

In this chapter, we compare three linear coding schemes offering different locality properties: HAM, a locally repairable code with all-blocks locality $r = 3$, RS(4,3), a Reed-Solomon code representing the worst-case locality $r = k = 4$, and PYR$(4, 3, \langle 2 \rangle)$, a Pyramid code with information locality $\langle r \rangle = 2$.

We experimentally evaluate HAM using simulations and a real-world fault trace. Our results show the benefits of relying on efficient XOR operations for encoding, as well as increased performance at repairing faulty systems.

We also present an efficient method for repairing HAM based on designs. When compared to RS(4,3) and PYR$(4, 3, \langle 2 \rangle)$, HAM shows lower latency, thanks to the repairing procedure. Similarly, using a real-world trace, HAM consumes less bandwidth thanks to its all-blocks locality property.

Our results also highlight how information locality and locality should not be confused: we show that a code offering information locality can outperform standard MDS codes in term of repair bandwidth when few erasures occur, and that such a restriction does not apply to the code with all-blocks locality.

Chapter 6

Block placement for fault resilient distributed tuple spaces

6.1 Introduction

We are currently observing a deluge of data originated by our personal devices. Distributed applications must be able to efficiently collect, store, process and expose data. When dealing with such applications, developers need to settle on a specific programming model, to i) facilitate the implementation of such systems and ii) retain user-friendliness and ability to scale, both horizontally and geographically. Distributed storage systems are one prominent example of such applications. They are typically operated across wide area networks, such as Amazon AWS, which currently comprises 15 geographical regions.[1] In such deployment scenarios, applications must transparently tolerate faults, a common threat for distributed systems.

The basic strategy to tolerate faults is to rely on block replication, which entails a huge storage overhead. A state-of-the-art solution to decrease such overhead while providing the same level of fault tolerance is to use erasure coding techniques. With a systematic linear code of length n and dimension k, each codeword consists of n blocks: k source blocks for the original data, and $n-k$ redundant blocks. The storage overhead is $\frac{n-k}{k}$, and if the code is MDS, any k of the n blocks are necessary and sufficient to recover the original data, as detailed is Section 2.1.

From a fault tolerance point of view, it is optimal to place the n blocks of a codeword on different logical units (with respect to failures), so that the MDS code can tolerate up to $n-k$ failures. A logical unit can be a single node (in this case for the optimum it is sufficient to place different blocks of a codeword on different nodes), but it can also be a cluster of nodes (e.g., a set of machines physically hosted in a single room can go down at the same moment if the cooling system of the room fails). In this second scenario, one is tempted to spread different blocks of a codeword into separate and faraway

[1] `http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-regions-availability-zones.html`

clusters. Although being optimal with respect to fault tolerance, this solution affects negatively the latency to fetch the blocks.

The case of distributed tuple spaces A programming model consists of two separate pieces: the *computation* model and the *coordination* model. The computation model allows the programmer to build a single computational unit, while the coordination model is the glue that binds separate activities into an ensemble [78]. The tuple space paradigm, based on this idea, offers a flexible technique to program parallel and distributed systems, by providing the abstraction of a shared space to which all the processes have access. In this model, communication between processes is *indirect* and *anonymous* as it is done through the shared (distributed) space. Moreover, data exist in a tuple space and do not belong to any process. Despite the simplicity of the model, very few implementations of tuple spaces offer fault tolerant facilities usually in the form of data replication ([79, 80]), with the drawbacks of space overhead and consistency maintenance. In this chapter, we consider an extended, distributed tuple space system with erasure coding capabilities. A tuple to be inserted in the tuple space is *erasure coded* and its blocks are placed across the nodes joining the tuple space group.

Chapter organization First, we study how to distribute the encoded blocks of single codewords over a large-scale network, in order to decrease the fetch latency. We do so by designing and evaluating several block placement heuristics, over synthetic and real-world network topologies. Second, we evaluate how the proposed heuristics behave with respect to data loss when injecting faults into the topology. Third, we leverage the results of our simulations to identify two suitable placement strategies that we deploy atop a simple distributed tuple space system with the aim of evaluating their performance in a practical setting.

This chapter is organized as follows. We survey the related work in Section 6.2 and introduce the tuple space paradigm in Section 6.3. In Section 6.4 we describe our block placement heuristics, which we evaluate in Section 6.5. We leverage the results to drive the prototype implementation described in Section 6.6. The implementation supports both erasure coding techniques and a pluggable mechanism to choose among the different placement strategies. We test the prototype in Section 6.7 and conclude in Section 6.8.

6.2 Related work

The tuple space coordination model is very appealing for distributed systems thanks to its space and time decoupling and its synchronization power. Hence, researchers have tried to add fault tolerance and security to tuple spaces. For example, DepSpace [81] is a Byzantine fault tolerant coordination service, which employs process replication for handling crashes and providing fault tolerance. An alternative to process replication is block replication, which entails the problem of block placement.

Block placement policies have been mainly studied in MapReduce contexts, such as Hadoop [82]. The main purpose of Hadoop data placement policy is to provide good balance between reliability, write bandwidth, read performance and load balancing [83]. Placing all replicas on a single node incurs the lowest

write bandwidth penalty but lacks redundancy: if the node fails, the data is lost. On the other hand, placing replicas in different data centres maximizes redundancy but increases the communication bandwidth.

Hadoop default strategy is to place the first replica on the same node as the client (for clients running outside the cluster, a node is chosen at random, although the system tries not to pick nodes that are too full or too busy). The second replica is placed on a different rack chosen at random. The third replica goes to the same rack as the second, but on a different node. Further replicas are placed on random nodes on the cluster, although Hadoop block scheduler avoids placing too many replicas on the same rack. Our *cluster-aware* and *distance-aware* strategies share some similarity with this approach, in that they take into account zones of the system that are more sensitive to simultaneous failures. Several enhancement are introduced in Hadoop with respect to block placement policies, such as pluggable policies (since v0.21) or guarantees of even distributions across the cluster (since v0.17). CoHadoop [84] is a lightweight Hadoop extension that gives applications a fine-grain control of data location. Similarly, our scheduling policies allow deployers to choose the destination of the blocks according to different performance criteria.

Adapt [85] mitigates availability heterogeneity issues in non-dedicated distributed computing environments. Adapt dynamically dispatches data blocks according to hosts storage capacities. Through simulations, this strategy is shown to reduce the application runtime by more than 30%, thanks to increased data locality and reduced data migration cost, though the improvement of performances is less significant for environment with higher network connectivity.

6.3 Tuple spaces in a nutshell

The tuple space paradigm, made popular by Linda [86], is an abstraction of shared associative memory for parallel and distributed computing. A tuple space is a repository of tuples that processes can concurrently access via *pattern-matching*. Processes create new tuples (`out` or `write` operation), test the existence of a tuple (`read` operation) and consume a tuple (via the `in` operation). The simplicity of this coordination model makes it intuitive and easy to use, also for distributed applications, making easy the implementation of some synchronization primitives such as semaphores and synchronization barriers [87]. The tuple space interaction model provides time and space decoupling, in that tuple producers and consumers remain anonymous with respect to each other [88]. Moreover a tuple has to survive its producer termination, which can be caused by a node crash or due to the end of the normal execution. In a distributed tuple space, each node writes tuples in its own local space, but it can read tuples also from remote ones. For example, in Figure 6.1 node **D** reads the tuple produced by node **C**.

Despite the wide development of tuple space implementations [89], very few of them offer support for distribution. While some systems use replication to guarantee data availability [79] or to be resilient to Byzantine faults [80], no existing system handles link or node faults to guarantee availability of data via erasure coding. The extensions presented in Section 6.6 fill this gap.

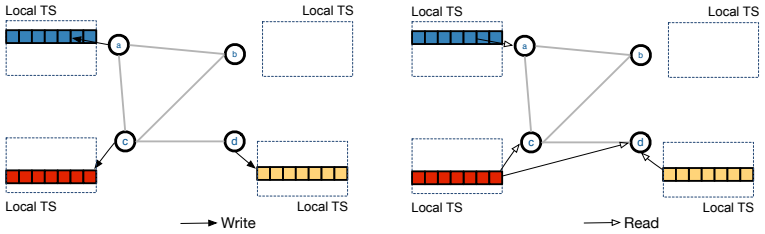


Figure 6.1: Example of distributed tuple space: each node writes tuples in its own local tuple space (left) and read tuples from local and remote nodes (right).

6.4 Block placement strategies

In this section, we describe several heuristics for block placement. Data is stored in the nodes of a graph representing a distributed storage system, using a standard Reed-Solomon code RS(10,4) for redundancy. The code divides inputs in 10 blocks, and encodes them into codewords of 14 blocks in such a way that any 10 encoded blocks are sufficient to recover the original 10. In other words, this linear code can withstand the loss of any 4 blocks of a codeword. The code provides the same level of fault-tolerance as 5 times replication while entailing a storage overhead of 40% only.

In this configuration, from a fault tolerance point of view, it is optimal to place the 14 blocks of a codeword on units failing independently, such as geographically remote nodes. In reality, nodes hosted in the same data centre are more likely to fail or being unreachable at the same time. Indeed there are several threats that can lead a data centre to a power outage, for example, cyber attacks, UPS system failures, air conditioner failures and human errors [90]. Thus, the proposed strategies must consider a tradeoff between:

- *Latency efficiency*: placing blocks apart from each other negatively affects the fetch latencies;
- *Failure resiliency*: if related blocks are placed geographically close to each other, a failure affecting a wide geographical area affects several blocks at once.

With the aim of experimentally understanding this tradeoff, we study five different placement heuristics. They take into account several structural graph properties (e.g., the clustering degree) with the objective of minimizing the latency for fetching blocks.

6.4.1 Round-robin (`rr`)

The graph is divided into $\mathcal{K}$ clusters $C_1, \ldots, C_\mathcal{K}$ using $\mathcal{K}$-means algorithm [91]. Then, the `rr` strategy places blocks in the clusters in turn. The first block goes in a random node inside cluster C_1, the second block goes in a random node in cluster C_2 and so on. The $(\mathcal{K}+1)$-th block goes in a random node inside cluster C_1, and the strategy proceeds in this fashion until it places all blocks.

6.4.2 Degree-proportional (`deg`)

The `deg` strategy places more blocks in nodes with higher degree (number of incident edges). Intuitively, it allows nodes with higher network capacity to serve more blocks, irrespectively of their geographical location.

6.4.3 Cluster-aware (`ca`)

The `ca` strategy places blocks in the cluster hosting the emitting node and two neighbouring clusters. Using $\mathcal{K}$-means, we divide the graph in $\mathcal{K}$ clusters $C_1, \ldots, C_\mathcal{K}$. We say that cluster C_i is at distance 1 from cluster C_j if there is an edge of the graph with source/target in C_i and target/source in C_j. Generally, we say that clusters C_i and C_j are at distance δ from each other if we must cross $\delta - 1$ other clusters to go from C_i to C_j and this is the smallest number possible.

In our simulations, we statically precompute the distances between clusters. We select the first cluster C to be the cluster containing the emitting node and two clusters C_1 and C_2 at distance 1 and 2 from C respectively. Then, we select at random 8 nodes from C, 4 nodes from C_1 and 2 nodes from C_2. These nodes receive the 14 blocks of the codeword. Notice that this heuristic needs at least 3 clusters to work.

6.4.4 Distance-aware (`da`)

The `da` strategy takes into account the distance between the emitting node and the other nodes in the graph. It assumes the knowledge of the diameter $d_{\max}$ of the graph, and proceeds as follows. First, 3 ranges of node-to-node distances (3 being a parameter of the algorithm) must be fixed: *short* (from the minimum to the 33^{rd} percentile of $d_{\max}$), *mid* (from the 33^{rd} to the 66^{th} percentile of $d_{\max}$), and *long* from the 66^{th} to $d_{\max}$.

For each codeword the algorithm selects the emitting node and 7 short-range, 4 mid-range and 2 long-range nodes with respect to the emitting one. The total 14 nodes receive the 14 blocks of the codeword. We report results for 3 ranges (`da3`), for 4 ranges (`da4`, for which the percentiles are 25^{th}, 50^{th}, 70^{th} and the number of blocks are 6, 4, 2, 1 for each range, respectively) and finally for 5 ranges (`da5`, using the percentiles 20^{th}, 40^{th}, 60^{th}, 80^{th} and the number of blocks are 5, 5, 1, 1, 1 for each range respectively).[2]

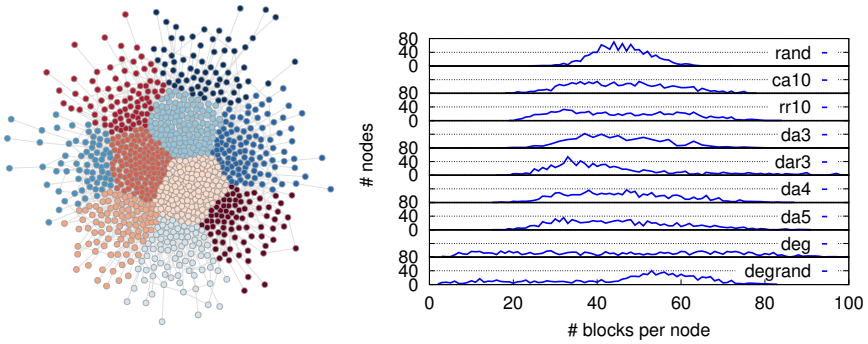
6.4.5 Random-Degree (`drnd`)

This strategy combines a naive random strategy with `deg`. Each strategy contributes for the placement of half of the blocks.

6.5 Block placement simulation

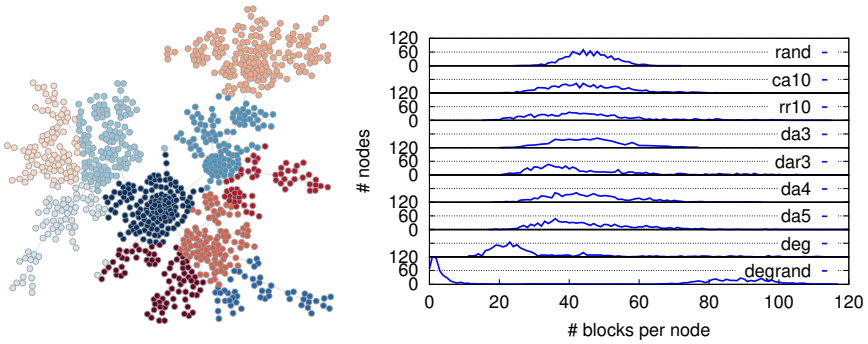
We present our simulations results with the aim of evaluating how the different placement strategies perform with respect to fetch latency and data loss.

[2]We experimentally select the number of blocks assigned to each range to work in practice.



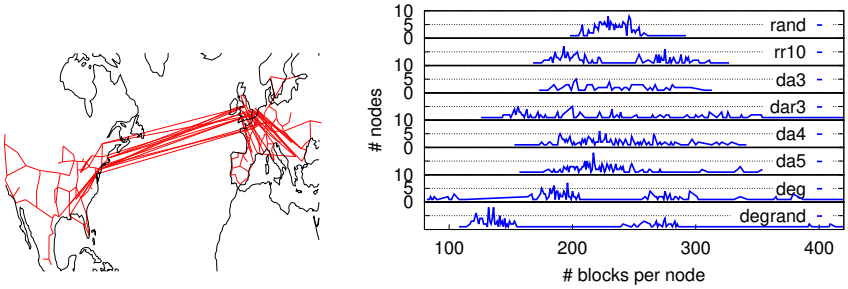
(a) Random graph, 10 clusters. (b) Blocks distribution.

Figure 6.2: Random topology and its blocks distribution.



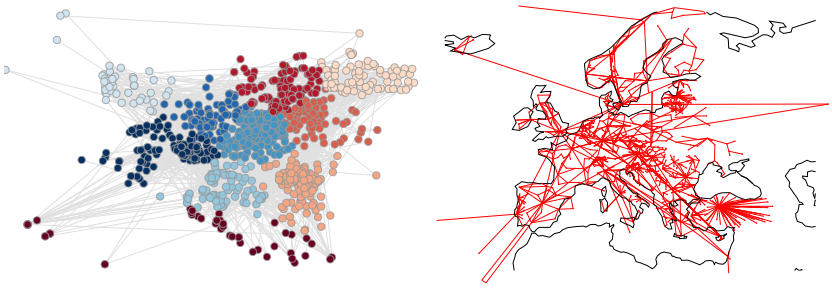
(a) Scale-free graph, 10 clusters. (b) Blocks distribution.

Figure 6.3: Scale-free topology and its blocks distribution.



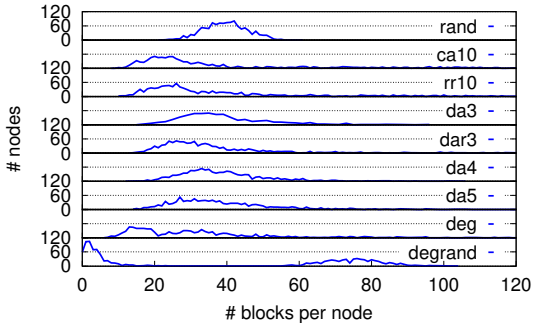
(a) Cogent graph, 2 clusters. (b) Blocks distribution.

Figure 6.4: Cogent topology and its blocks distribution.



(a) NREN graph, 10 clusters.

(b) NREN graph on Europe.



(c) Blocks distribution.

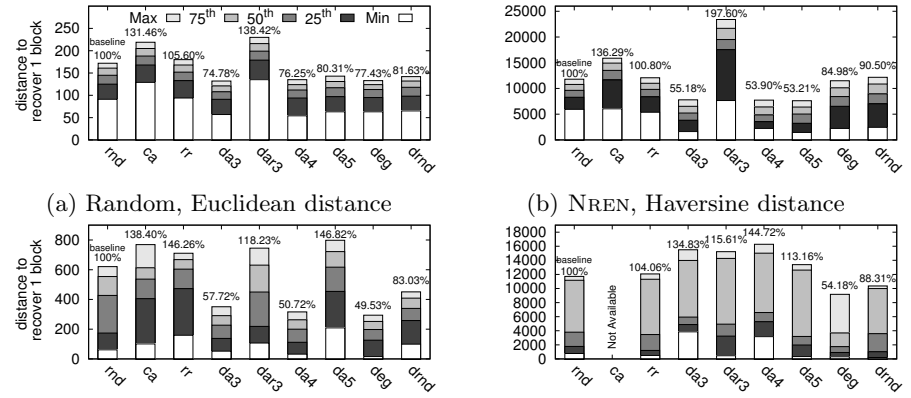
Figure 6.5: NREN topology and its blocks distribution.

6.5.1 Synthetic and real-world topologies

We start by presenting the topologies we use in our simulations. We consider a random graph of 1000 nodes, as depicted in Figure 6.2(a), where we highlight the $\mathcal{K} = 10$ clusters computed by $\mathcal{K}$-means using the Euclidean distance between nodes. Figure 6.3(a) shows the topology for a scale-free graph of 1000 nodes built using the preferential attachment method [92]. This topology closely maps a real Internet topology, yet is simple to study and analyse.

We also consider two real-world topologies. The first, depicted in Figure 6.4(a), is the Cogent network [93]. It counts 197 nodes and 245 edges only, nevertheless it spans Europe and USA presenting trans-oceanic links, i.e., 13 edges connect nodes between Europe and USA. The Full European Nren network [93], shown in Figure 6.5(b), has 1157 nodes and 1465 edges. When computing 10 clusters, as in Figure 6.5(a), we observe 1170 inter-cluster edges, i.e., source and destination nodes belong to different clusters.

As shown in Figures 6.2(a), 6.3(a) and 6.5(a), we fix the number of clusters to be $\mathcal{K} = 10$ in our simulations, except for Cogent topology which is split in $\mathcal{K} = 2$ clusters corresponding to USA and Europe. Results for `ca` strategy are not available for Cogent, since the heuristic requires at least 3 clusters. For Nren and Cogent, we know the geographical coordinates of the nodes. To take into account the curvature of the Earth and place more precisely the centroids of the clusters, we use the Haversine distance [94] as the $\mathcal{K}$-means distance function for Nren and Cogent networks.



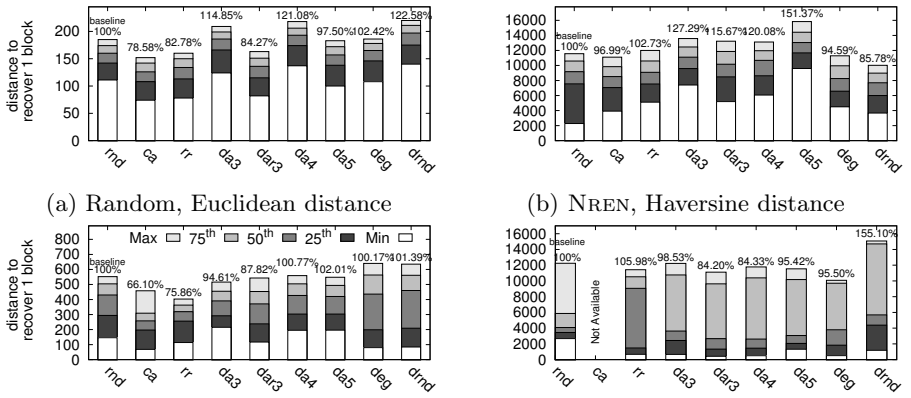
(a) Random, Euclidean distance

(b) Nren, Haversine distance

(c) Scale-free, Euclidean distance

(d) Cogent, Haversine distance

Figure 6.6: Distance for fetching blocks (*lucky node*).



(a) Random, Euclidean distance

(b) Nren, Haversine distance

(c) Scale-free, Euclidean distance

(d) Cogent, Haversine distance

Figure 6.7: Distance for fetching blocks (*unlucky node*).

6.5.2 Load Balancing

We study how the strategies spread blocks on the four different graph topologies. In Figures 6.2(b), 6.3(b), 6.4(b) and 6.5(c) the x axis represents the number of blocks per node, say it b. On the y axis, we display the number of nodes hosting b blocks, for $0 \leq b \leq 100$. For Cogent, we have $100 \leq b \leq 400$ as the graph comprises 197 nodes only, while we distribute the same amount of data blocks.

The `rnd` strategy produces a Gaussian distribution on the random graph, and we observe that on the scale-free graph `deg` and `drnd` give rise to a long-tail block distribution, i.e., several nodes host few blocks while few nodes store plenty of blocks. As an empirical confirmation that scale-free graphs are well-suited for representing Internet topologies, we highlight the similarity between the block distributions of the scale-free graph and the NREN network, in Figures 6.3(b) and 6.5(c) respectively. Overall, block distributions generated by the `da` and `rr` strategies tend to be bell-shaped, while `dar` and `deg` entail left-sided pick and long tail.

6.5.3 Fetching latency

We continue by evaluating how the proposed strategies differ in terms of block recovery latency, as observed by the clients wishing to reconstruct matching tuples. We assume that the fetch latency is proportional to the distance between nodes. Hence, we measure the length of the minimum paths between the node hosting the target block and the client.

We observe that a node storing a consistent amount of blocks necessarily needs to fetch only few ones to reconstruct tuples. Hence, for each topology and each placement heuristic, we distinguish 3 types of clients based on the number of blocks they store. The *lucky* and the *unlucky* node store the greatest and the smallest amount of blocks respectively.

We use a representation based on stacked percentiles throughout the reminder of this section. The white bar at the bottom represents the minimum value, the pale grey on top the maximal value. Intermediate shades of grey represent the 25^{th}, 50^{th} – the median – and 75^{th} percentiles. We compare the results against a baseline `rnd` strategy that randomly places blocks across the graph. Figures 6.6 and 6.7 present the results for the *lucky* and *unlucky* node respectively, which validate the intuition that the number of blocks the client is storing greatly affects the observed fetch latency. For instance, `da3` performs better than the other heuristics in 3 out of 4 topologies when the client is *lucky*. However, this is not the case for the *unlucky* case, where `deg` and `rr` perform better instead. These observations suggest that no strategy wins in all possible topologies, and that deployers need to carefully consider the different tradeoffs for their applications and workloads.

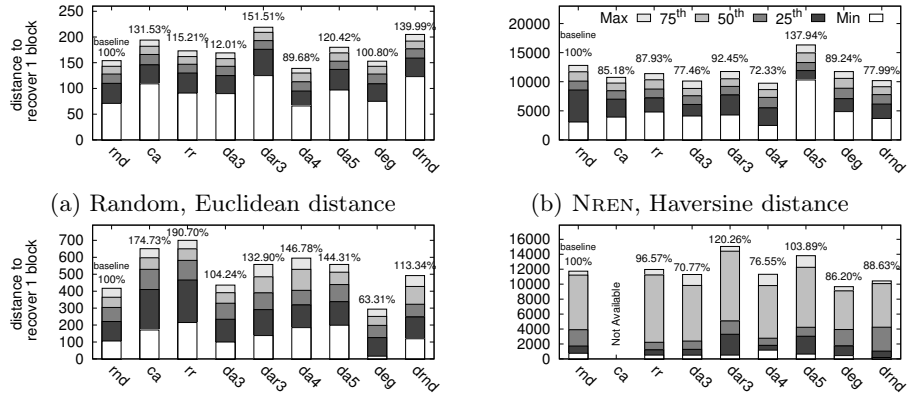
6.5.4 Fetching latency under faults

We perform the same set of experiments as in Section 6.5.3, injecting faults into the graph. For each graph, we select the most populated among the 10 clusters and we crash 1% of its nodes. This setting simulates a catastrophic event occurring to nodes geographically close to each other. Once the faults are injected, we let the *lucky* node, see Figure 6.8, and the *unlucky* node, see Figure 6.9, trigger the reconstruction of all the data stored. During these simulations, we do not observe any data loss. Hence, the heuristics spread blocks sufficiently apart from each other to tolerate crashes within the same cluster.

However, when injecting faults the fetch latency highly depends on the particular failing nodes. In the case of the Cogent topology, the `deg` strategy greatly improves the results produced by the `rnd` placement, while on the scale-free graph performance degrades for the unlucky client. The `da3` strategy outperforms the other heuristics in the NREN topology. In general, distance-aware heuristics seem to be well-suited for the random graph.

6.5.5 Simulation summary

Finally, to evaluate the statistical significance of the differences recorded by the simulations between the various heuristics, we perform two sets of *t-tests* [91] on fault-free graphs. First, we build the dataset of cumulative distances. For each node c, we compute the cumulative distance, that is, the sum of the length of all minimum paths covered to retrieve all the data in the system from that



(a) Random, Euclidean distance

(b) Nren, Haversine distance

(c) Scale-free, Euclidean distance

(d) Cogent, Haversine distance

Figure 6.8: 1% crashes in one cluster, lucky node.

particular client c. We fix a topology and compare different heuristics against each other. We find the following p-values:

	scale-free graph	random graph	Nren	Cogent
t.test(`rnd`,`deg`)	0.2486	0.03055	0.00761	0.7828
t.test(`rnd`,`da3`)	0.4805	0.3242	0.3774	0.2203

These p-values answer the question: "What is the probability that the means of the cumulative distances covered by the two heuristics are equal?". For every graph we find a strategy between `da3` and `deg` such that the probability is less the 25%. We consider this a low evidence that the two means are the same, but still such a value does not provide a decisive response.

For this reason, we further create a dataset of the distances covered to fetch every block by each node in the graph. In the case of the scale-free graph the dataset has 1000 entries times the number of blocks fetched, i.e., 3276000 entries. We run t-tests on random 1000-entries-samples from this dataset to compare different heuristics against each other. We find the following p-values:

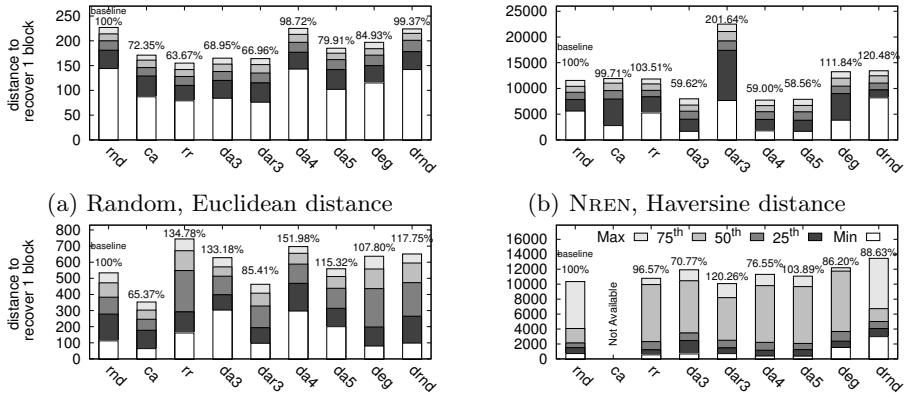
	scale-free graph	random graph	Nren	Cogent
t.test(`rnd`,`deg`)	0.1661	$6 \cdot 10^{-6}$	0.0004	0.6475
t.test(`rnd`,`da3`)	0.4215	0.0406	0.2936	0.1042

For every topology, we can find a strategy between `deg` and `da3` with support less than 16% for the hypothesis that the distance covered is the same as the one covered when `rand` is used. We take into account the modelling and statistical results to select `deg` and `da3` for implementation in a real tuple space, which we use to evaluate their performance in a practical setting.

6.6 Implementation

We implement and deploy three of the described block placement strategies (`da3`, `deg` and `rnd`) atop SimpleTS,[3] a tuple space implemented in Python (v3.4.0).

[3] `https://github.com/jmbjorndalen/SimpleTS`



(a) Random, Euclidean distance

(b) Nren, Haversine distance

(c) Scale-free, Euclidean distance

(d) Cogent, Haversine distance

Figure 6.9: 1% crashes in one cluster, unlucky node.

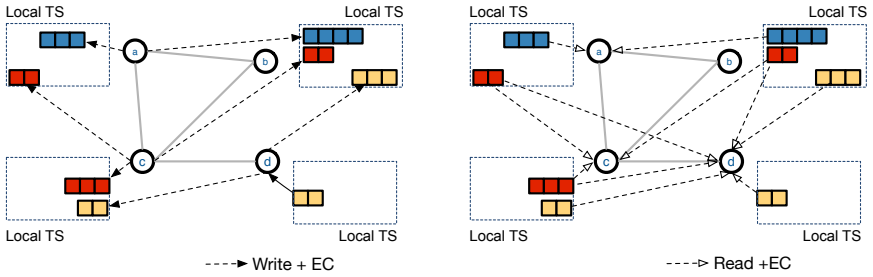


Figure 6.10: Distributed tuple space endowed with erasure coding: `write` operations spread blocks apart driven by a specific strategy; `read` operations fetch blocks from remote nodes.

The original implementation of SimpleTS does not support remote tuple space nodes. Therefore, we extend it to support a distributed scenario, leveraging Pyro (v4.0),[4] a remoting library for Python. Overall, our modifications to the SimpleTS source code consist of 250 additional lines of code.

To add erasure coding and block placement techniques, we extend the tuple space APIs with additional operations to properly handle writing, reading, and deletion of encoded tuples. For example, using a rs(10,4), the `out(t)` operation that emits the tuple `t` in the tuple space, becomes `out_ec(t)`. This version encodes the tuple, splits it into 14 blocks and, according to the chosen strategy, distributes these blocks among the other nodes. To this end, from the original tuple a list of tuples of the following form is created:

```
<tupleUID, blocksAndIndicesList, nodeList>
```

where `blocksAndIndicesList` is a list of pairs (b_i, i) indicating that b_i is the i-th block of the codeword, `tupleUID` is a unique identifier of the original tuple `t`, and `nodeList` is a list of nodes containing the remaining blocks. Figure 6.10 shows the extended version of SimpleTS with erasure coding abilities.

[4] `https://pythonhosted.org/Pyro4/`

Reading a tuple only requires to fetch 10 out of the 14 existing blocks. The tuple space programming paradigm requires the reading operations to operate via pattern-matching [86] and in the case of encoded tuples, the tuple space needs to decode the tuple. Therefore, this operation sequentially reads the tuples from the tuple space and checks whether the reconstructed tuple matches the template. Specifically, it leverages the `nodeList` index to discover and retrieve the missing blocks from other nodes in order to reconstruct the tuple. Clearly, in the worst case, to find a matching tuple the system has to decode the entire tuple space. We assume the existence of an up-to-date indexing service that serves the purpose of speeding up the process of discovering the location of the required blocks. In our evaluation, we assume to know the location of the nodes storing the blocks required to decode the tuple.

We implement both `da3` and `deg` strategies on our tuple space and test them on a scale-free network of 100 nodes. We emulate a large-scale network deployment using Docker (v1.13.1). We map each SimpleTS node (with its local tuple space) to a standalone container. The latency between two nodes, say i and j, is proportional to their minimum distance on the graph. Latency (by mean of a `sleep` system call) is then interposed by the proxy interface of the Pyro service exposed by each tuple space process. In practice, when node i contacts node j to read (or write) a tuple, node j sleeps $latency_{i,j}$ milliseconds before replying. An alternative method is to add latency at the OS level, e.g., by implementing a software router.[5]

6.7 Prototype evaluation

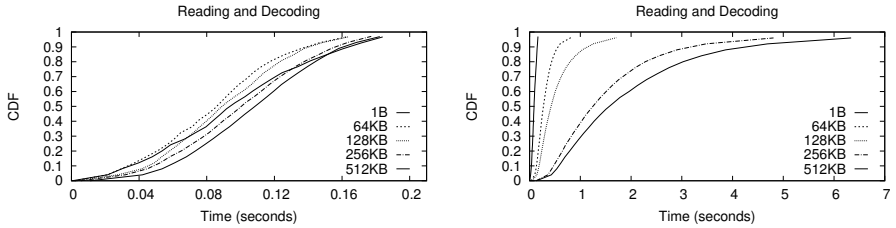
We present our evaluation of the extended SimpleTS system. Due to the lack of hardware resources, we are limited to a cluster of 100 nodes mimicking a scale-free network. Each node is executed by a SimpleTS Docker container, and only communication delays among nodes are emulated.

6.7.1 Erasure coding overhead

To evaluate the overhead of erasure coding, we execute an initial set of microbenchmarks for reading times. In this experiment, we vary the size of data stored in each tuple, from 1 byte to 512KB. At the beginning, we randomly distribute 1000 tuples across 100 nodes. Then, 10 random nodes read all the 1000 tuples, and we report the Cumulative Distribution Function (CDF) of the reading time. In Figure 6.11(a), the size of the tuple modestly affects the reading time with respect to the tuple space without erasure coding.

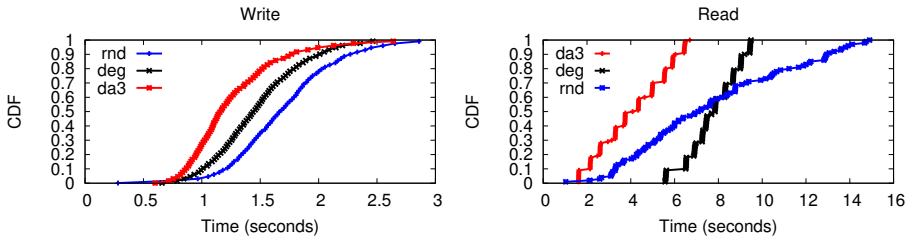
When erasure coding is enabled, see Figure 6.11(b), the reading time is more sensitive to the tuple size: it grows from milliseconds for 1 byte tuples to several seconds for 512KB tuples. For bigger tuples, the time for encoding and decoding is significantly higher. A highly optimized erasure coding library, such as Intel ISA-L [95], would greatly reduce the overhead and make it more practical.

[5] We report on our failed attempt in using Linux `tc`'s traffic shaping (using *delay.sh* `https://gist.github.com/arr2036/6598137`) to emulate network latencies. In particular, the current Docker networking layer does not cope well with this approach, where all nodes in a given network class (such as all the Docker containers running in the same host) apply the same delay, preventing the emulation of more complex graph topologies.



(a) Without erasure coding. (b) With erasure coding.

Figure 6.11: Tuple reading performance for increasing tuple sizes.



(a) Write. (b) Read.

Figure 6.12: Times for different strategies of blocks placement.

6.7.2 Experiments with different strategies

This experiment evaluates the performances of the tuple space using different block placement strategies. At the beginning, each of the 100 nodes writes 10 tuples. The tuples are encoded and split into blocks. Those are dispatched to remote nodes according to the given strategy. Finally, a node is chosen to fetch and read the blocks of its own tuples. Figure 6.12 presents our results for write and read operations.[6] We plot the CDF of the timings to write/read the tuples into/from the tuple space respectively in Figures 6.12(a) and 6.12(b). The `da3` strategy achieves the best performances for writes, as the writing time depends on the number of nodes used to spread tuple blocks. Random placement offers the worst performance because it involves a large number of nodes.

The first read is accomplished faster when blocks are distributed with `rnd` strategy. But under this strategy, the overall reading time is the longest (almost 15 seconds). Under the `deg` strategy we take almost 6 seconds to read the first block but the global job is concluded in less than 3 seconds. In our experiment, the most performant read is obtained when blocks are placed with `da3` strategy: we take less than 2 seconds to read the first block and conclude the job in around 4 seconds.

[6] We omit the results for the withdrawing operation. The trend is similar to the read results with a small overhead due to the fetching of all the 14 blocks.

6.8 Summary

The problem of block placement in a wide-area network setting is of paramount importance, but several distributed applications rely on a random strategy. We present a study of several block placement heuristics to dispatch blocks over the nodes of a distributed storage system. In particular, we consider a distributed application implemented via the tuple space paradigm, which we endow with erasure coding capabilities to efficiently cope with network faults, while being storage efficient and allowing fast fetching time. We extend an open-source Python-based tuple space implementation with distribution capabilities and erasure coding features.

We evaluate our block placement heuristics over synthetic and real-world graph topologies, up to thousands of nodes. The results of the simulations together with the prototype evaluation reinforce the belief that it is important to gather structural information about the network topology underlying a distributed storage system and select the block placement strategy accordingly.

In this chapter, we considered the distributed tuple space as practical use-case. We stress that our strategies are general purpose and can be deployed in other distributed systems, such as distributed key-value stores.

Chapter 7

RECAST: Random Entanglement for Censorship-resistant Archival STorage

7.1 Introduction

Users entrust an increasing number of documents to online cloud systems for long-term archival purposes. When archiving some content, users wish for their content to remain available and readily accessible at any time. Hence, an archival system must be highly durable. There are many threats to durability, and the longer the lifetime of some content, the worse it gets.

Broadly speaking, one can distinguish between passive threats (e.g., hardware failures) and active threats (e.g., a censor wishing to delete a specific document). A common countermeasure to mitigate passive threats is redundancy: extra information is stored in the system, with the goal of tolerating node or disk failures. On the other hand, effectively defending against active attacks still presents open challenges. Various solutions have been proposed, none of which is fully satisfactory. For example, security by obscurity [96, 97] encrypts content before archival so that the attacker cannot distinguish the data to censor, but this severely limits the operation that can be performed on data and the way it can be accessed or shared. Data entanglement [98, 99] attempts to weave together popular and unpopular data so that an attacker is forced to do collateral damage when censoring its target, but in a way that is unfit for deployment in practical systems.

We introduce RECAST, an anti-censorship data archival system based on a random data entanglement strategy proposed in [71] that provides strong guarantees while being readily applicable to real systems. RECAST's design and entanglement principles are based around the following intuition: by entangling pieces of data with one another, we enable potentially unpopular and rarely accessed content to benefit from the protection offered to other data in the system. Second, the asymmetry of the construction makes the system easy to

repair and hard to corrupt. While the randomness component of RECAST's entanglement makes it NP-hard to compute the minimal set of documents to be deleted to censor a target [71], error-correcting codes allow to repair the system recursively as long as the damage done by the attacker is recoverable.

To assess the security of RECAST, we assume a powerful adversary model where the censor has access to the metadata and thus knows how documents are processed, split, and distributed over the storage nodes. To further protect the system against a powerful attacker targeting metadata, RECAST includes an emergency disaster recovery mechanism that enables the system to rebuild metadata from the data itself.

We measure the protection by collateral damage units, that is, the number of additional documents to be deleted by a censor when targeting a specific document. In that respect, we design RECAST so as to offer strong long-term as well as fast short-term protection. We first build upon *uniform data entanglement* [71], which offers very strong long-term protection but leaves recently archived documents poorly protected. To address this limitation, we rely on *normal data entanglement* and temporary replication that provide fast short-term protection, but do not spread entanglement adequately across the archive. To get the best of both worlds, we introduce a hybrid approach, `nu`-entanglement, which exploits the strong long-term protection of uniform entanglement and the fast short-term protection of normal entanglement and temporary replication.

The rest of the chapter adopts the notation summarized in Table 7.2 and is organized as follows. We summarize our design goals in Section 7.2. Section 7.3 surveys similarities and differences with state-of-the-art censorship-resistant archival systems. In Section 7.4 we briefly recall entanglement and the STeP-archival strategy, and we refer to Section 4.6 for a more complete introduction. In Section 7.5 we propose a new practical technique to data entanglement, called `nu`-entanglement, which offers fast short-term and strong long-term protection for all the documents in the archive. We evaluate RECAST security in Section 7.6 and we conclude in Section 7.7.

7.2 Design goal

In this chapter, we aim at designing a long-term censorship-resistant system ensuring content integrity and durability. We understand durability as the ability to eventually retrieve any of the archived documents, see Section 4.2.

While high availability and confidentiality of data are desirable features of any storage solution, durability is paramount to long-term archival systems. Indeed, maintaining high availability may get expensive over time for documents that are infrequently accessed. Moreover, maintaining data confidentiality and managing the associated encryption keys over a long period of time (*e.g.* decades) unnecessarily exposes the system as a central point of failure. With time, encryption algorithms are broken and keys once considered large enough prove to be too short. Therefore confidentiality is best left deferred to the client's discretion. In contrast, the ability to retrieve documents for audit purposes or disaster recovery at a configurable cost answers a more practical need, especially in a context where independent providers operating as storage backends may become unreliable over time or worse, may attempt to censor, tamper or delete user content.

To address this problem, RECAST allows to store and retrieve documents ensuring their durability by establishing random document interdependency links, which enable recursive reconstruction of data beyond the local capability of the underlying erasure code. The tradeoff is that documents cannot be updated nor deleted from the archive. A file update must consist in a new file upload, and we do not offer file removal as enabling this operation for a user would enable it for the censor as well. This means that a system administrator compelled by law to remove content [100] must work as hard as an attacker who wishes to censor a file without causing collateral damage, i.e., recode the entire system starting from the file to delete. For completeness, we investigate the (huge) cost of this operation in our evaluation (see Section 7.6, document removal micro-benchmark).

7.3 Related work

The problem of anti-censorship for digital data has been first explored in [101] and subsequently extensively studied in e.g. [96, 97, 98, 99, 102, 103]. We can broadly distinguish between three main approaches: (i) *replication* to protect against a censor compromising a small number of servers [101], (ii) *anonymity* of the user to hide his identity and/or of the content to hide its location [96, 97], (iii) *entanglement* to prevent an attacker from deleting a single target document without causing collateral damage [98, 99].

To prevent censorship a system must detect that data has been compromised and be able to repair it if needed. While tamper detection is a basic feature offered by all censorship-resistant systems, data reconstruction has received much less attention: it is either not supported by the system [98] or implemented trivially by fetching a fresh replica of the corrupted data [96, 99, 102], assuming one exists. On the contrary, RECAST offers a powerful recursive data reconstruction method, as explained in Section 4.6.

In the remainder of this section, we survey the systems that directly offer storage-based anti-censorship properties, synthetically presented in Table 7.1.

DAGSTER [98] is a censorship-resistant publishing system. It uses an anonymous channel between data publishers/consumers and the servers. For each b-bits block of the original document, it stores the block encrypted and XORed with c old blocks from the pool of archived blocks. Such newly created block together with the c old blocks form a codeword. The original block can be locally retrieved (from the codeword) only if all blocks of the codeword are available. Similar to RECAST, the size b of blocks and the number of entangled blocks c can be configured but RECAST offers a much wider choice for coding parameters, thus enabling greater fault tolerance. Encryption guarantees server deniability only, as the encryption keys are stored in clear in the Dagster resource locator.

Similarly, TANGLER [99] allows users to publish a set of documents anonymously. It uses a naming convention and a public/private key pair that ensure that only the owner can update his own data. Each document block is entangled with exactly 2 old blocks β_1, β_2 using (3,4)-Shamir secret sharing [112]. The two parity blocks in output p_1, p_2 are then stored in the system. The owner distributes p_1 and p_2 (as well as the old blocks β_1 and β_2 used for entanglement), thus replicating other documents. Any three of these blocks can be used to reconstruct the original document. In contrast, RECAST provides a more flexible

	Type	Anonymous upload	Anonymous download	Server deniability	Fault tolerant	Caching	Open source	REST API (`put`/`get`)	Support for `update`	Support for `delete`	Recursive repair
DAGSTER [98]	(iii)	✓	✓	✓	✗	✗	✗	✓	✗	✗	✗
DEEPSTORE [104]	(i)	✗	✗	✓	✓	✗	✗	✓	✗	✗	✗
FREEHAVEN [97]	(ii)	✓	✓	✓	✓	✗	✗	✓	✗	✗	✗
FREENET [102, 105]	(i)	✓	✓	✓	✓	✓	[106]	✓	✗	✓	✗
OCEANSTORE [107]	(i)	✗	✗	✓	✓	✓	[108]	✓	✓	✗	✗
POTSHARDS [109]	(i)	✗	✗	✓	✓	✗	✗	✓	✗	✗	✗
PUBLIUS [96]	(ii)	✓	✗	✓	✓	✗	[110]	✓	✓	✓	✗
SAFESTORE [103]	(i)	✗	✗	✗	✓	✓	✗	✓	✓	✓	✗
TANGLER [99]	(iii)	•	•	✗	✓	✗	✗	✓	✓	✗	✗
RECAST	(iii)	✗	✗	✓	✓	✗	[111]	✓	✗	✗	✓

Table 7.1: Censorship-resistant systems. All provide tamper-evidence features. •=missing details.

scheme where the number of blocks required to retrieve a document depends on the configuration of the entanglement code in use.

FREENET [102, 105] is an anonymizing and censorship-resistant distributed storage system. To guarantee file anonymity, nodes know only their neighbours in the chain of queries, ignoring their specific role (producer/consumer of the file). Freenet lacks permanent storage guarantees: the least recently used files are deleted from a node's datastore when the arrival of a new file causes the datastore to exceed the designated size, which is configurable by the Freenet node operator. RECAST on the other hand is designed to store documents indefinitely, making it prohibitively difficult to delete a single document. Freenet places multiple copies of the same document across the nodes overlay, according to their popularity. A censor must target all those copies to permanently delete a document. RECAST makes it very hard for a censor to delete a single document without affecting a large portion of the archive.

FREEHAVEN [97] is an anonymous publishing system enabling users to associate an expiration time to documents, similarly to Comet [113]. It embeds a reputation system that relies on the capacity of nodes to store documents until their expiration. In contrast, RECAST assumes to always have sufficient storage capacity, and documents never vanish. Moreover, RECAST is not intended to provide anonymity, an orthogonal concern to be handled by upper layers.

PUBLIUS [96] encrypts and spreads documents over a set of (static) servers. The data encryption key is secret-shared among k of the n servers using (k, n)-Shamir secret sharing. Each server hosts the encrypted PUBLIUS content and a share of the key. The content can be tamper-checked because it is cryptographically tied to the Publius address: any modification of one of these two components causes the tamper check to fail. RECAST implements tamper-checks by exploiting decoding in document reads: if any block is corrupted, then the

decoding fails and reconstruction is triggered.

Mnemosyne [114] is a steganographic peer-to-peer storage system. Nodes are continuously filled with random data. Upon archival of a real document, its content is encrypted and made indistinguishable from the random substrate, preventing an attacker to determine the existence of a file and ensuring privacy and deniability of the content itself. As in Publius, data is spread across nodes [115] mainly for resiliency and each of the nodes is unaware of the other nodes holding parts of the file. Recast tolerates a stronger threat model where storage nodes have access to metadata information.

OceanStore [107] is a persistent data store on top of an untrusted infrastructure. It uses promiscuous caching (data is cached everywhere and at any time) to enhance locality and availability. This solution relies on classic erasure coding techniques, specifically a Cauchy Reed-Solomon code of length 32 and dimension 16, see Section 2.1, to ensure durability.

DeepStore [104] is a large-scale archival system for immutable data. It achieves a good efficiency/redundancy compromise by applying compression with inter-file dependencies and replicating the most valuable pieces of compressed information. DeepStore relies on the Presidio framework [116] to implement hybrid compression across heterogeneous data, as well as deduplication to eliminate redundant data.

Lockss [117] is an archival system based on a voting protocol. The opinion pools provide system peers with confidence in content authenticity and integrity. Moreover, content is replicated among peers and replicas are regularly audited to promptly repair any damage. Recast implements temporary replication to protect recently inserted documents and uses an auditing subsystem to remove temporary replicas from the system as soon as blocks are sufficiently entangled.

SafeStore [103] is a distributed storage system offering long-term data durability. It exploits an aggressive isolation scheme across multiple storage service providers. It uses an informed hierarchical erasure code that maximizes data durability and provides redundancy in the fault-isolated domains. Depending on the code's parameters, a certain number of faults can be recovered in each fault-isolated domain and an auditing subsystem further monitors data loss and triggers reconstruction. Recast and Safestore sit on opposite sides from an architectural point of view: while Recast uses entanglement, SafeStore exploits aggressive isolation, which leads to high durability but does not offer strong resilience against an active censor.

Percival [118] and Potshards [109] are two systems relying on secret-splitting techniques. The latter offers long-term security by using two levels of secret splitting and placement. The first level provides secrecy by XORing content with random data. It produces n fragments using (k, n)-Shamir secret sharing and places them into shards for availability. Shards retrieval is based on indexes accessible by all users, in a similar manner to Recast's metadata indexes. Potshards uses *approximate pointers* to allow for quick reconstruction of user data without the need of external information, similar to Recast's emergency recovery procedure.

Finally, the design of Recast is based on STeP-archival [71]. STeP-archival is the theoretical foundation of Recast and provides tools such as greedy attack heuristics and recursive reconstruction that we exploit in this chapter. We elaborate on such a base to build a *practical* STeP-archive. In practice, we overcome the poor short term protection of the uniform random entanglement

u	uniform entanglement
n	normal entanglement with standard deviation $\sigma = 1000$
nσ	normal entanglement with standard deviation σ
nu	normal-uniform entanglement with standard deviation $\sigma = 1000$
nuσ	normal-uniform entanglement with standard deviation σ
l	leaping attack
c	creeping attack
rα	level of temporary replication α

Table 7.2: Notation. E.g., we write `u-l-(s,t,e,p)-r`α for the leaping attack running on a (s,t,e,p)-archive with uniform entanglement and keeping α temporary copies for each document in the tail of the archive. The replication parameter is omitted when temporary replication is not in place.

studied in [71] by developing new entanglement heuristics. Section 4.6 presents STeP-archival in greater details and for its centrality in this chapter we recall its main parameters in Section 7.4.

7.4 STeP-archival

We briefly recall STeP-archival introduced in Section 4.6. A (s,t,e,p)-archive is a storage system where each archived document consists of a codeword of s source blocks, t pointers or tangled blocks, p parity blocks, and that can correct $e = p - s$ erasures per codeword. The document to be archived is split into $s \geq 1$ source blocks. From the archive, t distinct old blocks, the *pointers*, are selected and a RS$(s+t,p)$ code is used to create $p \geq s$ parity blocks depending on both source blocks and pointers. Finally, we only archive the p parity blocks.

To fully characterize the STeP-archive we have to choose a pointer selection strategy. Different flavours of entanglement are studied in [71]. In particular, uniform random entanglement selects pointer blocks uniformly at random. The randomness of this approach is an advantage as pre-planning an attack against such a structure is not feasible. Uniform entanglement has the main drawback of requiring increasingly longer periods of time to protect young documents as the archive increases. For example, Figure 7.1 shows that after inserting 10000 documents, 25% of them are either not pointed to or pointed to only once. An attacker wishing to tamper with these documents can thus do so without causing collateral damage. The authors of [71] also show how to speed up the protection of new documents by selecting the pointer blocks from a sliding window bounded to the recent past, although this has the drawback to bound the entanglement level of the documents in the archive.

7.5 Hybrid entanglement

In this section, we leverage the benefits of the uniform and window entanglement strategies. We present a hybrid entanglement technique mixing pointers chosen accordingly to a uniform and a normal distribution. We further enhance anti-censorship for recently archived documents using temporary replication.

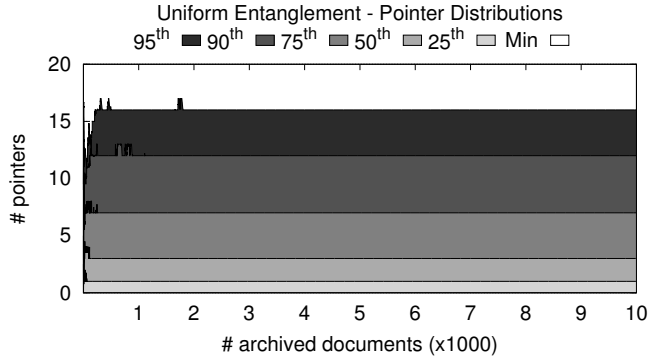


Figure 7.1: Variation of number of pointers to each document in a u-$(1, 5, 2, 3)$-archive with 10^4 documents. At the end, the 25% of documents is pointed to at most once, hence not protected by entanglement. Notation in Table 7.2.

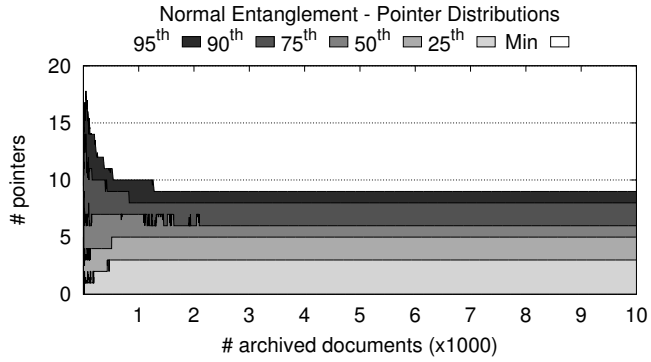


Figure 7.2: Variation of number of pointers to each document in a n-$(1, 5, 2, 3)$-archive with 10^4 documents. Normal entanglement evenly spreads the protection among documents: at the end the 50% of documents have between 5 and 9 pointers.

7.5.1 Normal entanglement

To archive document d_{i+1}, we extract t pointer blocks from a left half-normal distribution with standard deviation σ centred in the last archived document d_i. This models a sliding window of about size 2σ, in which the probability to point to a document older than $d_{i-2\sigma}$ is 5%, and the probability to point to a document older than $d_{i-3\sigma}$ is 0.3%. On the one hand, this means that documents are quickly protected: we just need to wait for 2σ documents to reach the average protection offered by the archive. Furthermore, this does not depend on the size of the archive but only on the chosen standard deviation σ. Comparing Figures 7.1 and 7.2 we see that the first quartile (25%) grows from pointed to at most once under uniform entanglement, to pointed to at most three under normal entanglement.

As discussed in the previous section, approximating a sliding window bounds the propagation of the entanglement. In particular, the creeping attack de-

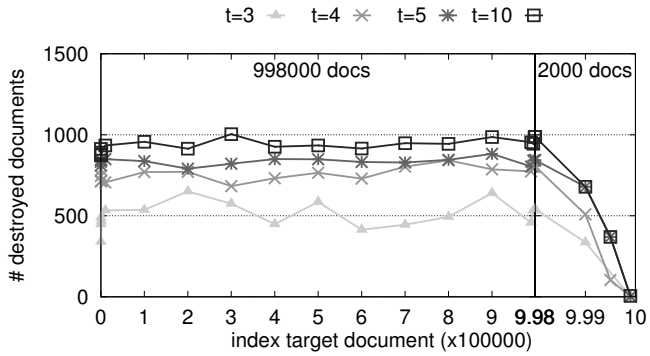


Figure 7.3: Number of corrupted documents to erase a target (x-axis) with a creeping attack in a n-$(1, t, 2, 3)$-archive for $t = 3, 4, 5, 10$. Notation in Table 7.2.

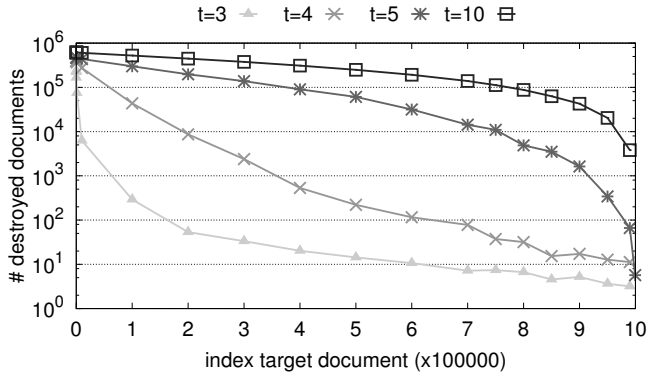


Figure 7.4: Number of corrupted documents to erase a target (x-axis) with a leaping attack in a nu-$(1, t, 2, 3)$-archive with $t = 3, 4, 5, 10$.

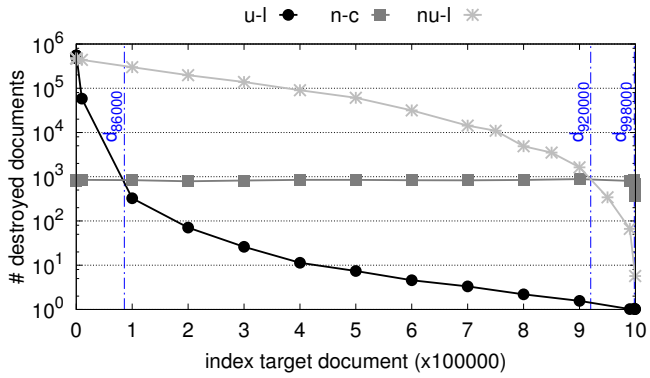


Figure 7.5: Best greedy attacks on uniformly-entangled, normally-entangled and nu-entangled $(1, 5, 2, 3)$-archives.

scribed in Section 4.6 is very effective. To confirm this intuition, we simulate the creeping attack on an archive storing 10^6 documents: as expected, the average number of documents to be censored, i.e., the protection, is constant through the archive and only the last 2000 documents have lower protection (Figure 7.3, right-most side of the x-axis). As expected, increasing the number of pointers t improves the average protection against the creeping attack: in Figure 7.3 the average number of documents to be destroyed to censor the target is about 506 for $t = 3$, 758 for $t = 4$, 835 for $t = 5$ and 935 for $t = 10$.

7.5.2 Hybrid nu-entanglement

To overcome the limitations of the described entanglement techniques, we blend normal and uniform entanglement together. We select each of t pointers by flipping a coin, so that with probability $\frac{1}{2}$ we have:

- a uniform pointer, providing the good randomness and the strong long-term protection of uniform entanglement,
- a normal pointer, offering the fast short-term protection of normal entanglement.

The best greedy heuristic on a STeP-archive with `nu`-entanglement is the leaping attack described in Section 4.6. Indeed the effectiveness of the creeping attack drops thanks to the restored randomness. In Figure 7.4, we observe a greater influence of the number of pointer blocks on the protection. In particular for $t = 5$, the long-term protection grows by almost 3 orders of magnitude with respect to normal entanglement and the short term-protection grows by about 2 orders of magnitude with respect to uniform entanglement.

We compare the three entanglement methods in Figure 7.5 over an archive of 10^6 documents. Using uniform entanglement, around $6 \cdot 10^5$ documents can be censored tampering with only 10 documents. On the other hand, using normal entanglement with $\sigma = 1000$, all the documents have the same protection except for the last $2\sigma = 2000$ documents. Moreover, when normal entanglement is in place, protection is offered fast: to erase d_{998000} (a recent one), a greedy attacker needs to tamper with more than 800 documents. In contrast, using uniform entanglement, d_{998000} can be censored by tampering with one document. As discussed, `nu`-entanglement mixes the two approaches to gain the best of both worlds: `nu`-entanglement offers greater protection than uniform entanglement for all the documents older than document d_{86000} and greater than normal entanglement up to document d_{920000}. Hence, `nu`-entanglement outperforms the other entanglement approaches on more than the 80% of the archive when tested with suboptimal greedy attacks.

7.5.3 Temporary replication

Regardless of the pointers selection method, entanglement takes time to provide protection to new documents (e.g., the last archived document is not pointed to). Hence, we use temporary replication until the entanglement kicks-in. The replicas are spread over the various storage nodes and enable fast recovery in case of node failure, at the cost of increased storage overhead. To reduce such costs, we periodically examine the level of protection of blocks and once a given

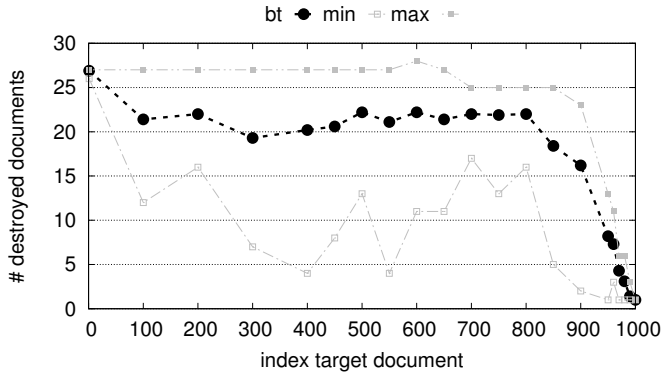


Figure 7.6: Backtracking the creeping attack on a n100-$(1, 5, 2, 3)$-archive of 1000 documents.

threshold is passed, RECAST removes the corresponding replicas from the storage nodes.

In the remainder of this section we study how to set such a threshold in a STeP-archive configured with `nu`-entanglement. We conduct this study by means of simulations, in order to establish when a given block's replicas can be safely removed. For uniform entanglement, as the number of blocks increases over time, the random selection of pointers lowers the probability of picking recent ones. Hence the replication level and the replication threshold are determined by the normal pointers. On a normally-entangled STeP-archive with 1000 documents, we run the creeping attack to get a first estimate, then we backtrack the tree of solutions to find the optimal one, exploiting branch pruning [71] to speed up the execution time. We present the results in Figure 7.6. The average protection offered by $t = 5$ normal pointers selected with standard deviation $\sigma = 100$ is 21.8. It drops with the 800th document, i.e., 2σ documents before the end of the archive. The average number of documents to be deleted is 8.2 on the full archive and 11.6 out of the tail. Hence to guarantee homogeneous protection to a RECAST system that uses a STeP-archive with $t = 10$ `nu`-pointers when the normal pointers are extracted with standard deviation $\sigma = 100$, we decide to replicate the tail, i.e., 2σ documents, 10 times each. We evaluate the failure resilience of this archive in Section 7.6.3.

7.5.4 Summary

Uniform entanglement provides strong long-term protection but needs a massive use of replication to reach an adequate level of short-term protection. To reduce the storage overhead, we study normal entanglement which provides constant long-term protection, fast short-term protection and a level of replication independent on the size of the archive. While the cost of replicas grows linearly with the size of the archive when using uniform entanglement, it goes to zero in the case of normal entanglement. Indeed, as the number of documents that need to be replicated is constant, in the long-term it becomes a negligible fraction of the whole archive. Finally, we present `nu`-entanglement that blends the two approaches to attain strong long-term protection via uniform pointers and

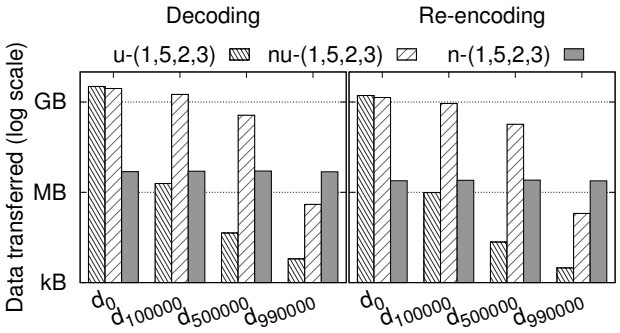


Figure 7.7: Bandwidth costs to delete a document in an archive with 10^6 documents. For each different STeP configuration (different fill pattern), we delete d_0 (the root of the history) and three other documents d_{100000} (old), d_{500000} (intermediate), d_{990000} (young).

fast short-term protection with a constant amount of temporary replication via normal pointers.

7.6 Evaluation

We evaluate RECAST's resilience to data loss, against active and passive adversaries. We also assess the cost of a document removal, which is, nevertheless, not offered as a functionality of RECAST, as discussed in Section 7.2.

7.6.1 Micro-benchmark: document removal

In Figure 7.7, we study the bandwidth cost to actually "delete" a document from the archive. In practice, the deletion of a target document is a three phase process: finding a (possibly minimal) complete set S of documents entangled with the target, decoding the documents in S and re-encoding these documents with other pointers.

The more accurate the first step is, the less expensive the actual deletion is. For this experiment, we compute the number of documents in the set S from the results shown in Figure 7.5. The cost of the last two steps depends on the RS code used in the STeP-archive: in Figure 7.7 we present results for a $(1, 5, 2, 3)$-archive with different entanglement strategies. The block size corresponds to the document size (1kB in the plot) and we compute 3 parity blocks out of the source and pointers using a RS(6,3) code. Hence decoding and re-encoding require fetching 6 and sending 3 blocks per document respectively (the source is not stored and the pointers to re-encode are chosen from the archive itself). The constant bandwidth required for normal entanglement is determined by the constant protection offered to documents. When using `nu`-entanglement on an archive of 1kB documents, erasing d_{100000} and d_{500000} from the archive requires the transfer of 2.7 GB and 0.55 GB respectively.

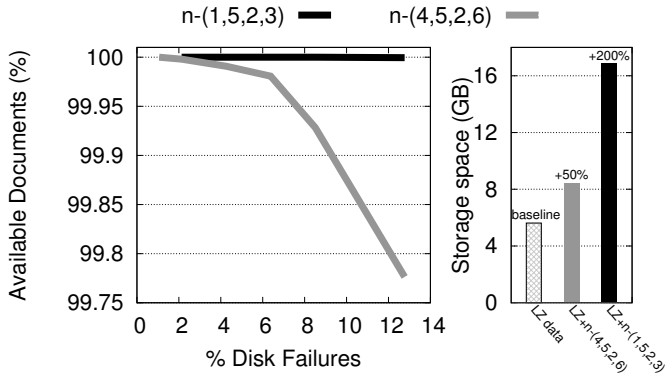


Figure 7.8: File availability as dependent on disk failures (on the left) using a certain storage space (on the right).

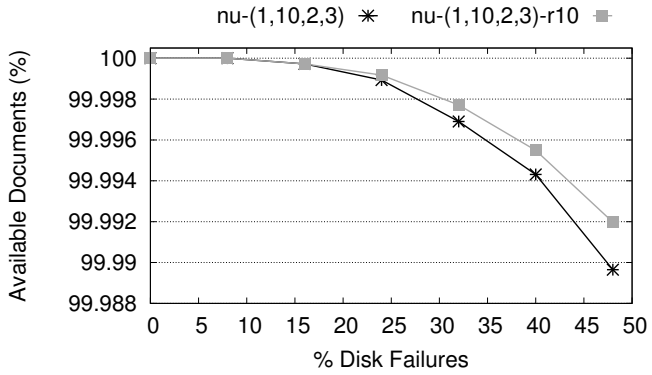


Figure 7.9: File availability as dependent on disk failures with and without temporary replication.

7.6.2 Macro-benchmark: passive attacks

We now evaluate the resilience of RECAST to passive attacks. We put under the label of passive attacks all threats to the system that neither leverage particular knowledge nor privileged access to it, e.g., nodes crashes or unavailable nodes for maintenance.

To evaluate the data loss RECAST undergoes when nodes fail we instrument simulations in the Internet Archive scenario proposed for the evaluation of DeepStore [119]. For the sake of comparison, we simulate RECAST with 188 nodes storing 196664 documents and we assume that the original data is compressed with the Lempel-Ziv (LZ) algorithm before archiving. We evaluate data availability after failures and storage space, respectively on Figure 7.8 left and right. In particular, we use two configurations for STeP with different storage overheads, namely 200% and a lighter 50%. The results show how parameters highly impact the system reliability in the passive attacker scenario and well prove the resistance of RECAST against random node failures: with a light-storage configuration, in the catastrophic event of about 13% of disks failing simultaneously, the data loss is less than 0.25%.

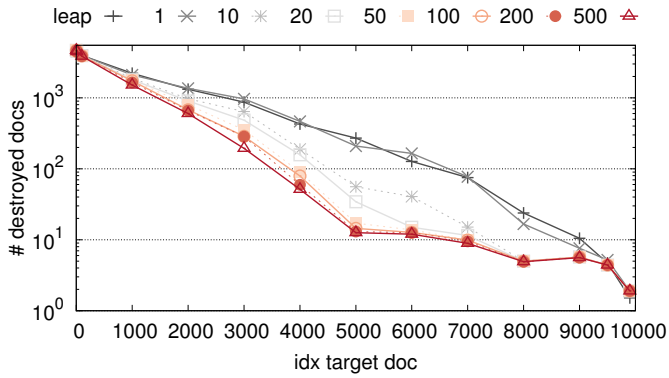


Figure 7.10: Number of corrupted documents to erase a target (x-axis) in a nu-$(1, 5, 2, 3)$-archive with 10^4 documents. We explore an increasing number (as indicated in the legend) of partial solutions at each step of the leaping attack.

In Figure 7.9, we evaluate the availability when introducing temporary replication. We simulate RECAST with 50 nodes storing 50000 documents, and configured with a $(1, 10, 2, 3)$-archive with standard deviation $\sigma = 100$. We simulate such system both without replication and by replicating the youngest 300 documents 10 times each. We randomly erase between 8% and 48% of nodes (x-axis), we trigger recursive reconstruction and record the percentage of available documents. We notice that temporary replication becomes useful when more than the 15% of disks fail, that is, in a non-catastrophic scenario the availability of the RECAST-archived documents is guaranteed by the `nu`-entanglement itself.

7.6.3 Macro-benchmark: active attacks

We move now to the evaluation of RECAST resiliency against an active attacker, i.e., an attacker that knows the metadata describing the system, has access to all its node and pro-actively exploits these abilities. We simulate such a powerful attacker via greedy heuristics which, as mentioned, are suboptimal. We improve them by exploring many partial solutions in the intermediate steps of the leaping attack. In practice, we perform much costlier computations which better approximate the unknown optimal solution. We show how the set of blocks to be deleted gets smaller in Figure 7.10. For the first document archived, increasing from 1 to 500 partial solutions at each step leads to an insignificant improvement (from 4704 to 4662) in the number of documents to be destroyed. Small improvements also apply to the tail of the archive. Opposite, the number of documents to be tampered to censor d_{2000} drops from 1359 to 601. Nevertheless, it should be noted that to halve the number of documents to be erased we multiplied by 500 the number of solutions that we expand and explore at every step. We also note that for the tail of the archive, it is sufficient to explore 10 solutions at each step of the greedy algorithm, while for the rest of the archive, in our simulation, a buffer size of 100 guarantees that the order of magnitude of the collateral damage is stable.

7.7 Summary

Archival systems are designed to support long-term storage of documents. As we turn into a "digital society", these systems become increasingly important. Archival storage can leverage the same mechanisms that classical data stores use, notably cryptography and redundancy, to protect against common types of attacks. It should however be resilient against more subtle attacks that would threaten the long-term integrity of the archived data, in particular offering strong protection against attackers that covertly tamper with the data or delete specific documents (i.e., censors).

In this chapter, we present RECAST, a novel anti-censorship archival system based on random data entanglement [71]. It exploits erasure codes to generate redundant blocks combining content from multiple documents, old and new, in order to protect them from both failures and malicious attacks. As opposed to prior work, RECAST allows efficient recursive repair while requiring censors to do an increasingly large amount of work over a large number of storage nodes as the archive scales, which is a highly desirable property.

RECAST uses a hybrid strategy for data entanglement designed to offer fast short-term and strong long-term protection to all the documents in the archive. This means that entanglement is performed in such a way that documents become more resilient as they stay longer in the system, and the level of interdependencies makes it quickly impossible to delete or tamper with a single document without causing collateral damage to a large number of other documents.

Chapter 8

Convolutional LPDC codes for distributed storage systems

8.1 Introduction

The requirements of large-scale distributed storage systems are often ill-suited to coding techniques developed in other settings. Erasure codes must have small length to store data objects upon arrival, high code rate to mitigate the storage overhead and low repair locality to cope with a small number of unavailable nodes efficiently.

Our contributions We study high rate convolutional LDPC codes of period 1 [120] for immutable distributed storage systems. Convolutional LDPC codes allow efficient local encoding and decoding of data objects using small constituent codes, while improving the reliability of these constituent codes by allowing message passing algorithms when local decoding is insufficient. Our construction sequentially archives data objects, each consisting of a single codeword. Each data object has s data blocks, is entangled with $t \triangleq s + p$ blocks already archived with which the code generates p parity blocks. Aided by mathematical analysis to avoid bad structures, we do an exhaustive search and provide the best constructions for $1 \leq s \leq 5$ and $p = 2$. As an example, optimal $\text{BSTEP}(5, 2)$ is more reliable than a $\text{RS}(20,8)$ code, with a complexity comparable to that of $\text{RS}(10,4)$ codes used by Facebook [121].

State of the art The performance of LDPC codes in the small length regime has not been ascertained. The finite-length analysis of LDPC codes on the binary erasure channel (BEC), presented in Section 4.5.3, provides ensemble results for the error probability together with a combinatorial characterization of decoding failures via message passing. Although the behaviour of individual codes is likely to concentrate around the ensemble average, the performance evaluation of code instances remains open. Moreover, degree distributions tailored to achieve capacity asymptotically lead to a significant decoding overhead

factor, when used with small lengths, that is, the number of blocks required to reconstruct the data is greater than the code dimension, see Section 4.5.1. Moreover, Reed-Solomon codes outperform LDPC codes in this regime when the download speed is slow. Small LDPC codes minimizing the decoding overhead are proposed in [122], but their performance in the presence of erasures is not analysed. The reliability of low repair bandwidth LDPC codes of length $n \geq 60$ is studied in [22], see Section 4.5.2, and an archival storage system based on Tornado codes of length 96 and rate $\frac{1}{2}$ is implemented in [70], see Section 4.5.5.

Ensemble-wise, coupling LDPC codes increases the decoding threshold to the maximum possible on the BEC [123]. On the AWGN channel, such gain of convolutional LDPC codes over the underlying LDPC block code is verified via simulations for $n \geq 145$ [124]. Implementation aspects of convolutional LDPC codes, e.g., systematic encoding and code termination, are discussed in [125], but besides an instance of length 128, simulations only cover codes with $n \geq 512$. Short to moderate length convolutional LDPC codes based on quasi-cyclic LDPC codes are built in [126], but again only with three-digit-number code lengths.

Outline The rest of the chapter is organised as follows. In Sections 8.2 and 8.3 we respectively introduce BSTEP(s,p) and define its structure. In Section 8.4 we compute the minimal erasures which we use for the reliability analysis of Section 8.5. We discuss extensions in Section 8.6 and conclude in Section 8.7.

8.2 Coding scheme: BSTEP(s,p)

We define BSTEP(s,p) to be a binary STeP-archive, see Section 4.6, with s source blocks, p parity blocks and $t \triangleq s+p$ pointer blocks. Let $\alpha \triangleq (\alpha_1, \ldots, \alpha_t)$ such that $\alpha_1 \neq 0$ and $\alpha_i < \alpha_j$ for all $1 \leq i < j \leq t$ be the entanglement strategy, and define

$$M(l) = \begin{cases} \ell + t & \text{if } 1 \leq \ell \leq t \\ \ell - t & \text{if } t < \ell \leq 2t \end{cases}. \tag{8.1}$$

The pointer column $1 \leq i \leq t$ is a α_{t-i+1}-shift of the data or parity column $M(i)$. Figure 8.1 shows BSTEP$(1,2)$.

Encoding works as follows: each data object is split into s data blocks, combined with $t = s+p$ pointer blocks (i.e., t blocks already archived) selected accordingly to $M(\cdot)$ and α, and encoded using a binary parity-check matrix H to generate p parity blocks. The binary entanglement parity-check matrix H has size $p \times 2t$ and minimum distance at least 2, i.e., no column of H is the zero column, see Proposition 2.2. The s source blocks and p parity blocks are then archived on the physical medium. The archive also includes a bootstrapping block (such as an all-zero block, not shown in Figure 8.1) for early blocks pointing before the beginning of the archive. Note that the choice of $M(\cdot)$ is not restrictive as changing $M(\cdot)$ corresponds to a permutation of the columns of H, as long as one pointer column maps to one source or parity column.

The scheme is systematic, thus data objects can be simply read from a fresh archive. To decode BSTEP(s,p) in the presence of erasures, we can leverage the reconstruction algorithm proposed in [71] and summarised in Section 4.6.1, which coincides with message passing in our binary setting. The recursive

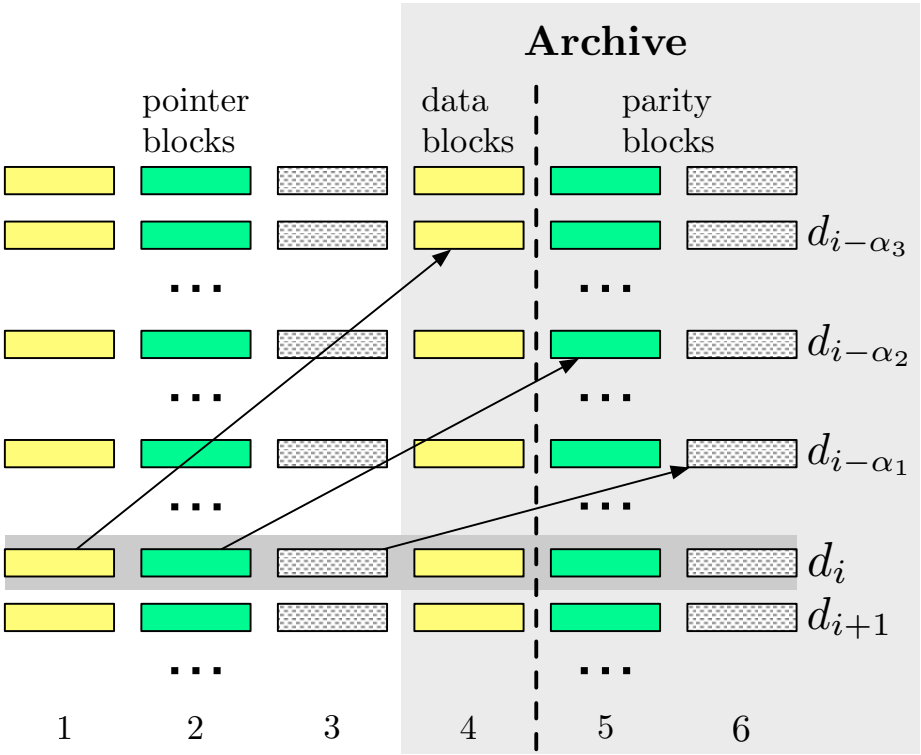


Figure 8.1: In bSTEP(1, 2), Column 1 is a α_1-shift of Column 4 ($M(1) = 4$ or equivalently $M(4) = 1$), Column 2 is a α_2-shift of Column 5 ($M(2) = 5$) and Column 3 is a α_3-shift of Column 6 ($M(3) = 6$). Each line in the figure represents the codeword corresponding to a data object.

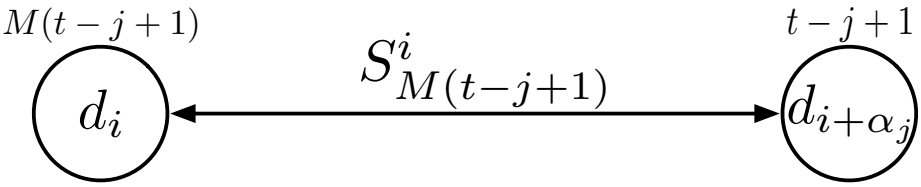
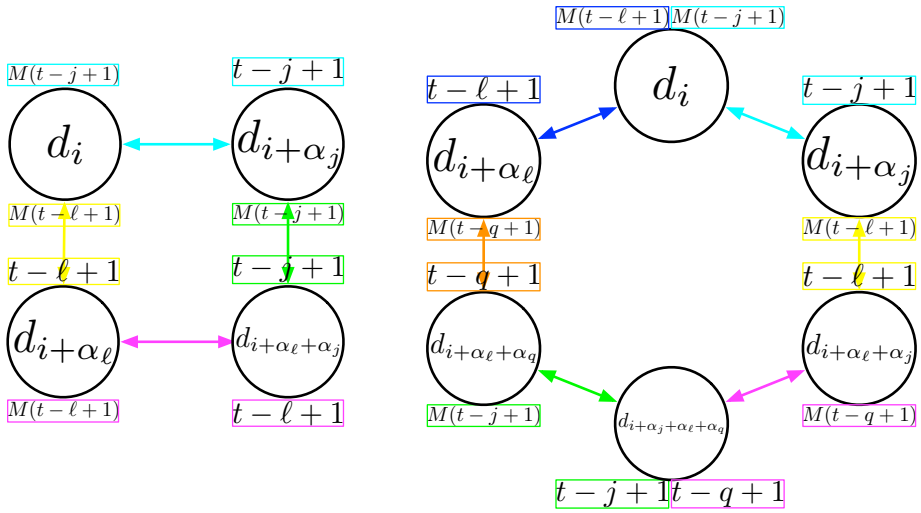


Figure 8.2: Two data objects of indices i and $i+\alpha_j$ share a block $S^i_{M(t-j-1)}$ which is at coordinate $M(t-j+1)$, $t-j+1$ in the codeword encoding the data object d_i, $d_{i+\alpha_j}$ respectively.



(a) Size 4.

(b) Size 6.

Figure 8.3: Cycle inducing a minimal erasure.

reconstruction starts at the length-$2t$ codeword level and enhances the repair capability of H thanks to the entanglement. Thus, the reliability of $\textsc{bstep}(s,p)$ depends on the choice of H and α, which we analyse in the next section, where we make extensive use of the fact that single erasures are repairable by H.

8.3 Selection of H and α

A *set of erasures* is a set of erased symbols, an *erasure pattern* is a set of erasures that result in some data loss, and a *minimal erasure* is an erasure pattern for which every erasure is necessary and sufficient for it to be an erasure pattern, see Section 3.4.4. Furthermore, in our setting:

- An *intrinsic erasure pattern* is caused by the choice of H, thus independent of the entanglement strategy α;
- An *incidental erasure pattern* is due to the particular entanglement strategy α.

We therefore impose conditions on H to get rid of small intrinsic erasure patterns, and remove incidental erasure patterns by stretching α. We now describe

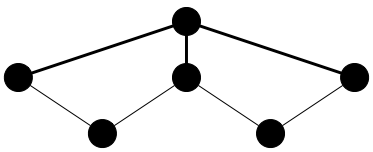


Figure 8.4: 3 erasures in a data object cause the size of the minimal erasure to be at least 7.

our strategy in more details.

Let B be the set of blocks forming a minimal erasure, and let $G(B)$ be the graph whose set of edges is B and whose vertices are the data objects hosting blocks in B. Let C be any cycle in $G(B)$. We fix an orientation for C and we define the length of edge e from data object d_{i_1} to data object d_{i_2} as $|e| = i_2 - i_1$, possibly negative. Let E_i be the set of edges of C of length $\pm\alpha_i$ for $1 \le i \le t$.

Lemma 2. *The minimal erasure B is* intrinsic *if and only if, for every cycle C in $G(B)$, $|E_i| \equiv 0 \pmod 2$ and $\sum_{e \in E_i} |e| = 0$ for all $1 \le i \le t$.*

We now derive conditions for small intrinsic erasure patterns to occur. We will later avoid these conditions in our search for the best parity-check matrices. Let $1 \le c_1, c_2 \le 2t$ be distinct coordinates of a codeword from H. We say that the pair $[c_1, c_2]$ is a minimal erasure for H when the blocks in position c_1 and c_2 cannot be reconstructed simultaneously by H.

Lemma 3. *An intrinsic minimal erasure of size 4 can only consist of a cycle of size 4, see Figure 8.3(a).*

Lemma 4. *An intrinsic minimal erasure of size 6 can only consist of a cycle of size 6, see Figure 8.3(b).*

Lemma 5. *Minimal erasures of size 3 and 5 are not intrinsic.*

Proof of Lemma 3. Block $S^i_{M(t-j+1)}$, see Figure 8.2, is:

- in coordinate $M(t-j+1)$ on data object of index i,
- in coordinate $t-j+1$ on data object of index $i+\alpha_j$.

A minimal erasure of size 4 spans 4 data objects (3 data objects do not share 4 pairs, and on 5 data objects a minimal erasure comprises at least 5 blocks). Such data objects are i, $i+\alpha_j$, $i+\alpha_\ell$, $i+\alpha_j+\alpha_\ell$ for a starting point i and $1 \le j < \ell \le t$. The set of erasures forms a minimal pattern if and only if H can repair none of the following pairs $[t-\ell+1, t-j+1]$ at $d_{i+\alpha_j+\alpha_\ell}$, $[M(t-\ell+1), t-j+1]$ at $d_{i+\alpha_j}$, $[t-\ell+1, M(t-j+1)]$ at $d_{i+\alpha_\ell}$, and $[M(t-\ell+1), M(t-j+1)]$ at d_i. □

Proof of Lemma 4. If a minimal erasure involves three erased blocks in a data object, then its size is at least 7, as we now show. Let us call i the index of the data object having 3 erased blocks. As $\alpha_\ell \ne \alpha_j$ for all $1 \le \ell, j \le t$, $\ell \ne j$, the three blocks point to three different data objects, say, $i+\alpha_{j_1}$, $i+\alpha_{j_2}$, $i+\alpha_{j_3}$, see Figure 8.4. If α is chosen properly, the three data objects $i+\alpha_{j_1}$, $i+\alpha_{j_2}$, $i+\alpha_{j_3}$ do not share any block. Data objects $i+\alpha_{j_1}$ and $i+\alpha_{j_2}$ both share a (different) block with data object $i+\alpha_{j_1}+\alpha_{j_2}$. Data objects $i+\alpha_{j_1}$ and $i+\alpha_{j_3}$ both share

a (different) block with data object $i+\alpha_{j_1}+\alpha_{j_3}$. Data objects $i+\alpha_{j_2}$ and $i+\alpha_{j_3}$ both share a (different) block with data object $i+\alpha_{j_2}+\alpha_{j_3}$. We need to involve at least two out of the three data objects of indices $i+\alpha_{j_1}+\alpha_{j_2}$, $i+\alpha_{j_1}+\alpha_{j_3}$, $i+\alpha_{j_2}+\alpha_{j_3}$ to have each vertex in the graph of degree at least 2. This leads to at least 7 blocks in the set of erasures.

As a consequence, a minimal erasure of size strictly smaller than 7 corresponds to a cycle. □

Proof of Lemma 5. The cycles of size 3 and 5 are a triangle and a pentagon respectively. They are not intrinsic from Lemma 2. □

8.4 Minimal erasures

To compute the size and multiplicity of the smallest minimal erasures of our BSTEP(s,p) scheme, we implement the Stopping Set Enumeration (SSE) algorithm [35]. We fix an arbitrary entanglement strategy $\alpha = (\alpha_1, \ldots, \alpha_t)$ as well as a parameter τ corresponding to the largest size of the smallest minimal erasures we expect to reach.

Let I_p be the $p \times p$ identity matrix. We test all the possible matrices $H = (h_1 \cdots h_{2t-2}\, I_p)$ with no minimal erasures of size 4. By carefully choosing the condition that guarantees that the cycle corresponding to a minimal erasure of size 4 (see Figure 8.3(a)) is disconnected, we simultaneously avoid some minimal erasures of size 6 (see Figure 8.3(b)). For each of the resulting matrices we run the SSE algorithm and compute the multiplicity n_i^H of the smallest intrinsic minimal erasure of size $1 \leq i \leq \tau$. Let σ_H be the smallest minimal erasure for BSTEP(s,p) entangled with H, i.e., $n_i^H = 0$ for all $1 \leq i < \sigma_H$ and $n_{\sigma_H}^H \geq 1$.

We of course want the smallest minimal erasure to be large and to have low multiplicity. Hence, we select H with the minimal erasure of size $\sigma = \max_H \sigma_H$ occurring with multiplicity $n_\sigma = \min_H n_\sigma^H$. Possibly after adjusting α, these matrices provide the best possible BSTEP(s,p) in terms of the size σ of the smallest minimal erasure and its multiplicity n_σ.

8.4.1 Results

We search for the best BSTEP(s,p) codes for $p = 2$ and $1 \leq s \leq 5$. The results are summarised in Table 8.1. We report the storage overhead, the size of the smallest minimal erasure σ and its multiplicity n_σ. We also report the number $\#H$ of parity-check matrices achieving the best pair σ, n_σ for a suitable α, and the smallest right degree (of the corresponding Tanner graph) of the optimal solutions. Table 8.1 also includes the smallest entanglement strategy and a parity-check matrix, not necessarily unique, achieving the best construction.

For BSTEP$(1,2)$ there is only one matrix reaching $\sigma = 12$ and $n_\sigma = 1$. For BSTEP$(2,2)$ we find eight matrices reaching $\sigma = 10$ and $n_\sigma = 1$. Out of those, four matrices have right degree 5 (for efficiency in repairing single erasures the lower the better). For BSTEP$(3,2)$ we find 12 matrices reaching $\sigma = 10$ and $n_\sigma = 2$, all having half check nodes of degree 6 and half of degree 7. For BSTEP$(4,2)$ we find 72 matrices reaching $\sigma = 9$ and $n_\sigma = 2$, having check nodes degree 7 or 8. For BSTEP$(5,2)$ we find 216 matrices reaching $\sigma = 8$ and $n_\sigma = 6$, having check nodes degree 9 or 10. Among them 96 have $\sigma = 8$, $n_\sigma = 6$ and $\mathsf{r} = 9$.

Storage overhead	s	α	r	σ	n_σ	$\#H$	H
200%	1	$(1,5,6)$	4	12	1	1	$\begin{pmatrix}1&1&0&1&1&0\\1&0&1&1&0&1\end{pmatrix}$
100%	2	$(1,4,6,11)$	5	10	1	8	$\begin{pmatrix}1&0&1&0&1&1&1&0\\1&1&0&1&1&0&0&1\end{pmatrix}$
67%	3	$(1,4,6,11,19)$	7*	10	2	12	$\begin{pmatrix}1&0&1&1&0&1&1&1&1&0\\1&1&0&1&1&1&0&0&0&1\end{pmatrix}$
50%	4	$(1,4,10,17,29,53)$	8*	9	2	72	$\begin{pmatrix}1&0&0&1&1&1&1&0&1&0&1&0\\1&1&1&0&0&1&1&1&0&1&0&1\end{pmatrix}$
40%	5	$(1,4,10,17,24,43,79)$	9	8	6	96	$\begin{pmatrix}0&0&1&1&1&1&1&1&0&0&1&1&1&0\\1&1&0&0&0&1&1&1&1&1&0&1&0&1\end{pmatrix}$

Table 8.1: Best constructions for BSTEP$(s,2)$ with $1 \le s \le 5$. We optimise for the size of the smallest minimal erasure σ and its multiplicity n_σ. We indicate with * right-irregular codes.

8.5 Reliability

8.5.1 Markov model

We can study the reliability of coding schemes for storage systems using Markov models, see Chapter 3. For the centrality of the reliability evaluation in this chapter, we recall the key features of the model.

A Markov model is characterized by mean time to failure (`mttf`), or equivalently drive failure rate $\lambda = \frac{1}{\texttt{mttf}}$, and mean time to repair (`mttr`), or equivalently drive rebuild rate $\mu = \frac{1}{\texttt{mttr}}$. The rebuild rate μ from state $i+1$ to state i is determined by the recovery rate of a single block as we assume that the recovery does not depend on the total number of available chunks [39]. The transition rate σ_i from state i to state $i+1$ is $\sigma_i = (n-i)\lambda p_i$, where p_i is the probability that the code can withstand one more node failure, given that it has already tolerated i failures. We compute such a conditional probability as $p_i = \frac{r(i+1)}{r(i)}$ where $r(i)$ is the fraction of repairable patterns of size i [42]. For non-MDS codes, we might go from state i directly to the DL state. We represent this with the probabilities $\delta_i = (n-i)\lambda(1-p_i)$.

Using the Markov model, we can compute the mean time to data loss (MTTDL) of a representative codeword by solving a linear system. Then, the MTTDL of a large system is obtained by dividing the MTTDL of a codeword by the number of codewords [42]. For $\text{BSTEP}(s,p)$, however, we must estimate the system-wide MTTDL directly since the archive consists of a unique convolutional codeword, as we explain next.

8.5.2 Approximation for $r(i)$

For a MDS code of length n and dimension k we have $r(i) = 1$ for $0 \leq i \leq n-k$ and $r(i) = 0$ for $n-k < i \leq n$. Since $\text{BSTEP}(s,p)$ archives are not MDS, the number of repairable patterns of size ranging from the minimum distance to the maximal fault tolerance is strictly bigger than 0 and strictly smaller than 1. To compute such numbers we resort to an approximation based on the idea that the smallest minimal erasures are dominant.

Let S_D be the set of fully entangled blocks when the archive holds D data objects. Let σ be the size of the smallest minimal erasure and $\beta \geq \sigma, B \leq |S_D|$. Let $\mathcal{N}_D(\beta, B)$ be the number of erasure patterns of size $B \geq \beta$ coming from minimal erasures of size β when the archive holds D data objects. Then $\mathcal{N}_D(\beta,\beta)$ is the number of minimal erasures of size β at D data objects and $\mathcal{N}_D(\beta,\beta) = n_\beta \cdot D$. The fraction of repairable patterns of size β is

$$r(\beta) = 1 - \frac{\sum_{i=\sigma}^{\beta} \mathcal{N}_D(i,\beta)}{\binom{|S_D|}{\beta}} = 1 - \frac{\mathcal{N}_D(\sigma,\beta) + \sum_{i=\sigma+1}^{\beta} \mathcal{N}_D(i,\beta)}{\binom{|S_D|}{\beta}}.$$

Like staircase codes, for which the dominant contribution to the error floor is due to minimal stall patterns [127], we can show that minimal erasures of size σ are dominant, i.e.,

$$r(\beta) \approx 1 - \frac{\mathcal{N}_D(\sigma,\beta)}{\binom{|S_D|}{\beta}}$$

3-REPL	Triple replication	[121]
RS(10,4)	Reed-Solomon code of dimension 10 and length 14 used in HDFS-RAID at Facebook	[121]
RS(6,3)	Reed-Solomon code of dimension 6 and length 9 used in GFS II at Google	[39]
WAS(6,4)	LRC of dimension 6 with 2 local and 2 global parities for Windows Azure Storage (WAS)	[41]
XORBAS(10,6)	LRC of dimension 10 with 4 RS parities and 2 local parities implemented in HDFS-Xorbas	[40]

Table 8.2: State-of-the-art-codes used for comparison.

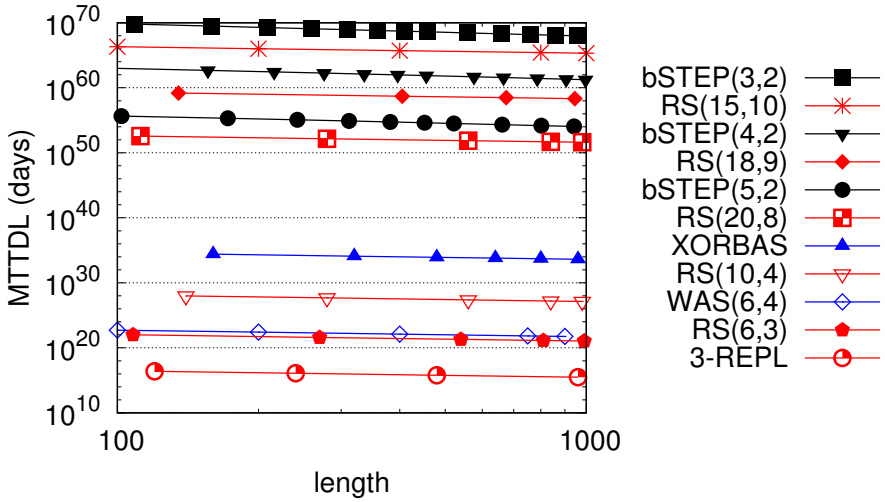


Figure 8.5: System MTTDL. On the x-axis we report the number of (fully entangled) blocks in the system. We refer to Table 8.2 for state-of-the-art erasure codes and indicate XORBAS(10,6) simply as XORBAS.

and that duplicates (non-minimal erasure patterns arising from different minimal erasures) are negligible, i.e,

$$r(\beta) \approx 1 - \frac{\mathcal{N}_D(\sigma,\sigma)\binom{|S_D|-\sigma}{\beta-\sigma}}{\binom{|S_D|}{\beta}} = \frac{n_\sigma \cdot D\binom{|S_D|-\sigma}{\beta-\sigma}}{\binom{|S_D|}{\beta}}. \tag{8.2}$$

Knowing the size of smallest minimal erasure σ and its multiplicity n_σ provides us with an approximation for $r(\beta)$ capturing the correct order of magnitude for the number of erasure patterns of size β, thus leading to an accurate estimate of $r(\beta)$ for $\sigma \leq \beta \leq |S_D|$.

8.5.3 Comparison between bSTEP(s,p) and other codes

We can build bSTEP(s,p) for any value of s and p greater than 1. As usual, greater fragmentation increases encoding and decoding complexity, however the increase is higher for bSTEP(s,p) than for a standard linear codes of dimension

s and redundancy p, as increasing s and p leads to a code of length $s + p + t = 2s + 2p$. Nevertheless, BSTEP(s, p) is a light XOR-based scheme, not requiring finite field arithmetic. Furthermore, in order to maximize the reliability for a fixed BSTEP(s, p), we need to stretch α, which impacts the tail of not-fully-entangled blocks at the end of the archive, see Section 8.6. For BSTEP$(s, 2)$ the repair locality is $\mathtt{r} \geq s + 2$, as we use an entanglement parity-check matrix H with $2s + 2p$ nonzero columns. The schemes detailed in Section 8.4.1 achieve the best σ and n_σ, but their repair locality increases with s.

With these tradeoffs in mind, we compare the MTTDL of BSTEP$(s, 2)$ against state-of-the-art erasure codes, summarised in Table 8.2, using the Markov model parameters from [40]. The results are presented in Figure 8.5. The mean time to failure of a disk $\mathtt{mttf} = \frac{1}{\lambda}$ is set at 4 years. For the 3-replication scheme, the block repair rate equals the cross-rack communication bandwidth over block size $\frac{1\text{Gbps}}{256\text{MB}}$ in [40], leading to a $\mathtt{mttr}$ of 2 seconds. We assume that all the codes repair at this rate, ignoring for example the effect of using heavy or light decoders. Nevertheless, the respective reliability order of the coding schemes meets the results in [40, 41].[1]

While we do not consider the numbers from Figure 8.5 valuable by themselves, see Section 3.4, we use them to compare the reliability of BSTEP$(s, 2)$ with respect to equivalent storage overhead schemes. In particular, at a 66% storage overhead, BSTEP$(3, 2)$ has the same reliability than a Reed-Solomon RS(15,10) code. At a 40% storage overhead, the reliability of BSTEP$(5, 2)$ is comparable to the reliability of RS(20,8), with the local complexity of RS(10,4) used in HDFS-RAID at Facebook.

8.6 Extensions and Discussion

8.6.1 Protecting the tail with replication

When D data objects are archived in BSTEP(s, p), the first $tD - \sum_{i=1}^{t} \alpha_i$ blocks are fully entangled, which leaves a tail of $\sum_{i=1}^{t} \alpha_i$ blocks that do not fully benefit from the protection of entanglement. In turn, minimal erasures of size smaller than σ comprising blocks from the tail arise. For example, for BSTEP$(5, 2)$ as in Table 8.1, the tail amounts to 178 blocks. We can secure these recent blocks by properly replicating them until they are fully entangled. Since the size of the tail is constant, the overhead cost of these additional replicas becomes negligible as the archive grows.

8.6.2 Increasing the number of pointers: bSTEP(s, p, t)

In this work, we set $t = s+p$. We can increase the fault tolerance of BSTEP(s, p) by relaxing this hypothesis and choosing $t > s+p$. Increasing t rises the complexity of encoding and decoding, increases the repair locality of the code, enlarges the tail of the archive, and complicates the selection of H and α by breaking the symmetry. However, the gain in fault tolerance can be significant. For example consider BSTEP$(2, 2)$, which has a single minimal erasure of size 10.

[1] The increasing reliability order is 3-REPL followed by RS(6,3) and WAS(6,4) in [41], and 3-REPL followed by RS(10,4) and XORBAS(10,6) in [40].

By increasing t from 4 to 5, the scheme loses its symmetry, and we cannot entangle columns as prescribed by $M(\cdot)$ in Equation (8.1). However, if we write s_i^j (p_i^j) for the i-th source (parity) block of the j-th data object, and if we entangle the columns as $s_1^{i-\alpha_5}$ $s_1^{i-\alpha_4}$ $s_2^{i-\alpha_3}$ $p_1^{i-\alpha_2}$ $p_2^{i-\alpha_1}$ s_1^i s_2^i p_1^i p_2^i, choosing $H = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \end{pmatrix}$ and $\alpha = (1, 7, 13, 49, 81)$, we can increase the size of the smallest minimal erasure from 10 to 14 (and multiplicity 1).

8.7 Summary

In this chapter, we focus on LDPC codes for distributed storage applications. As opposed to most work on these codes, we target small block lengths, for which the asymptotic results do not apply. Combining binary linear codes and data entanglement, we build convolutional LDPC codes that enable on-the-fly encoding of $s \geq 1$ source blocks. The LDPC structure also allows fast repair of a large number of erasures via message passing. We study and characterize minimal erasures to determine the best schemes in terms of erasure correction. Our Markov analysis reveals that, for the same storage overhead, state-of-the-art erasure codes require more fragmentation to attain the same reliability as our scheme.

Chapter 9

Conclusion

In this thesis, we consider several aspects of erasure coding for distributed storage systems. In what follows, we summarise our contributions and recap open problems together with the related future work.

Summary of contributions We first introduce the mathematical foundations of coding theory and present state-of-the-art coding techniques for distributed environments. The objective of these methods is to cope with failures so that stored content can be reliably retrieved from a network of individually unreliable nodes. For years the trend has been to privilege methods offering an optimal tradeoff between fault tolerance and storage overhead, but more recent techniques are tailored to optimize maintenance-oriented metrics. In particular, we quantify the impact of the repair locality on the bandwidth consumption for repair operations as well as on the overall reliability. We show that, as opposed to information locality, all-blocks repair locality can substantially decrease the bandwidth utilisation during repair. We also tackle the problem of block placement in fault-tolerant distributed systems. For reliability purposes, one is tempted to spread codeword blocks far from each other, on units unlikely to fail simultaneously. However, the more we disperse the blocks, the greater is the fetch latency to retrieve them. We evaluate several block placement heuristics taking into account different network topologies, targeting fast write/read of data object to/from the nodes comprising the systems, while ensuring fault tolerance. We dedicate the last two chapters of the thesis to the enrichment of the above coding techniques via data entanglement. In particular, we demonstrate that the combination of random data entanglement together with erasure coding provides anti-censorship properties to the system, and that structured entanglement combined with binary linear codes produces strongly reliable schemes.

Perspectives The work about entanglement calls for new challenges. RECAST, i.e., the system implementing random entanglement on top of erasure coding, exposes the metadata server as a single point of failure. Indeed, while it is very difficult to remove a single document thanks to data entanglement, if the metadata is damaged, the system is unable to identify and locate entangled blocks making reconstruction operations impossible. Thus, for RECAST to be a practical system, metadata must also be protected. This is currently being addressed by the systems team in Neuchâtel who recently submitted a solution

for the problem using Ethereum smart contracts. We could also use alternative coding techniques together with random entanglement. For example, the MDS erasure code can be replaced by a locally repairable code to improve the local repair performance and cope with single failures more efficiently.

An open challenge regarding structured entanglement on top of binary linear block codes is the study of how much the reliability can be improved by relaxing the hypothesis on the number of pointers t, i.e., to have $t > s + p$ entangled blocks per codeword. The approach rises many costs (it increases encoding and decoding complexity as well as the repair locality, it enlarges the tail of the archive, and it complicates the selection of the entanglement structure by breaking the symmetry) but can greatly enlarge the smallest minimal erasure, as we demonstrated. Moreover, because entanglement needs time to kick in, we proposed to replicate blocks belonging to the tail of not-fully-entangled documents. It remains an open problem to set an appropriate level of replication for such blocks, as well as to determine the real impact of the tail on the overall reliability. We envision a prototype implementation demonstrating that the on-the-fly encoding performance is competitive with respect to standard erasure codes and that reconstruction by message passing endows our scheme with a reconstruction efficiency comparable to that offered by long LDPC codes.

Appendix A

Publications

[Pub1] Roberta Barbi, Vitaly Buravlev, Claudio Antares Mezzina, Valerio Schiavoni.
Block placement strategies for fault-resilient distributed tuple spaces: an experimental study (Practical experience report).
In *IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS)*, pp. 67–82, 2017.

[Pub2] Roberta Barbi, Pascal Felber, Hugues Mercier, Valerio Schiavoni.
Worst-case, information and all-blocks locality in distributed storage systems: An explicit comparison.
In *IEEE 15th Canadian Workshop on Information Theory (CWIT)*, pp. 1–5, 2017.

[Pub3] Roberta Barbi, Dorian Burihabwa, Pascal Felber, Hugues Mercier, Valerio Schiavoni.
RECAST: Random Entanglement for Censorship-resistant Archival STorage.
In *48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 171–182, 2018.

[Pub4] Roberta Barbi, Pascal Felber, Laurent Hayez and Hugues Mercier.
Convolutional LPDC codes for Distributed Storage Systems.
Submitted to *IEEE International Symposium on Information Theory (ISIT)*, 2019.

Bibliography

[1] Accessed on December 3rd, 2018. [Online]. Available: https://www.domo.com/learn/data-never-sleeps-5

[2] Accessed on December 3rd, 2018. [Online]. Available: https://www.domo.com/learn/data-never-sleeps-6

[3] Accessed on October 16th, 2018. [Online]. Available: https://www.theguardian.com/commentisfree/2018/mar/28/all-the-data-facebook-google-has-on-you-privacy

[4] Accessed on October 16th, 2018. [Online]. Available: http://www.internetlivestats.com/twitter-statistics/

[5] Accessed on October 16th, 2018. [Online]. Available: http://thesocialskinny.com/100-social-media-statistics-for-2012/

[6] Accessed on October 16th, 2018. [Online]. Available: https://gizmodo.com/5937143/

[7] Accessed on October 16th, 2018. [Online]. Available: https://www.statista.com/statistics/264810/number-of-monthly-active-facebook-users-worldwide/

[8] Accessed on October 16th, 2018. [Online]. Available: https://www.statista.com/statistics/274564/monthly-active-twitter-users-in-the-united-states/

[9] Accessed on November 13th, 2018. [Online]. Available: https://hadoop.apache.org/docs

[10] D. Borthakur *et al.*, "HDFS architecture guide," *Hadoop Apache Project*, vol. 53, pp. 1–13, 2008.

[11] Accessed on December 3rd, 2018. [Online]. Available: https://blog.cloudera.com/blog/2009/02/the-small-files-problem/

[12] P. Elias, "Coding for two noisy channels," in *Information Theory, Third London Symposium*, vol. 67, 1955, pp. 61–76.

[13] F. J. MacWilliams and N. J. A. Sloane, *The theory of error-correcting codes*. Elsevier, 1977.

[14] W. C. Huffman and V. Pless, *Fundamentals of error-correcting codes*. Cambridge University Press, 2010.

[15] R. Lidl and H. Niederreiter, *Introduction to finite fields and their applications*. Cambridge University Press, 1994.

[16] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *Journal of the society for industrial and applied mathematics*, vol. 8, pp. 300–304, 1960.

[17] J. Bloemer, M. Kalfane, R. Karp, M. Karpinski, M. Luby, and D. Zuckerman, "An XOR-based erasure-resilient coding scheme," 1995.

[18] Accessed on November 6th, 2018. [Online]. Available: https://www1.icsi.berkeley.edu/~luby/

[19] D. J. MacKay and D. J. Mac Kay, *Information theory, inference and learning algorithms*. Cambridge University Press, 2003.

[20] R. G. Gallager, "Low-density parity-check codes," Ph.D. dissertation, MIT, 1963.

[21] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, D. A. Spielman, and V. Stemann, "Practical loss-resilient codes," in *ACM Symposium on Theory of Computing (STOC)*, 1997, pp. 150–159.

[22] H. Park, D. Lee, and J. Moon, "LDPC Code Design for Distributed Storage: Balancing Repair Bandwidth, Reliability and Storage Overhead," *IEEE Transactions on Communications*, 2017.

[23] J. S. Plank and M. G. Thomason, "A practical analysis of low-density parity-check erasure codes for wide-area storage applications," in *IEEE International Conference on Dependable Systems and Networks (DSN)*, 2004, pp. 115–124.

[24] C. Di, D. Proietti, I. E. Telatar, T. J. Richardson, and R. L. Urbanke, "Finite-length analysis of low-density parity-check codes on the binary erasure channel," *IEEE Transactions on Information theory*, vol. 48, no. 6, pp. 1570–1579, 2002.

[25] A. Orlitsky, K. Viswanathan, and J. Zhang, "Stopping set distribution of ldpc code ensembles," *IEEE Transactions on Information Theory*, vol. 51, no. 3, pp. 929–953, 2005.

[26] A. Orlitsky, R. Urbanke, K. Viswanathan, and J. Zhang, "Stopping sets and the girth of Tanner graphs," in *IEEE International Symposium on Information Theory (ISIT)*, 2002, p. 2.

[27] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. A. Spielman, "Efficient erasure correcting codes," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 569–584, 2001.

[28] A. Shokrollahi, "LDPC codes: An introduction," Digital Fountain Inc., Tech. Rep., 2003.

[29] V. Guruswami, "Iterative decoding of low-density parity check codes (a survey)," *arXiv preprint cs/0610022*, 2006.

[30] M. A. Shokrollahi, "Capacity-achieving sequences," in *Codes, Systems, and Graphical Models*. Springer, 2001, pp. 153–166.

[31] K. M. Krishnan and P. Shankar, "Computing the stopping distance of a Tanner graph is NP-hard," *IEEE transactions on Information Theory*, vol. 53, no. 6, pp. 2278–2280, 2007.

[32] K. M. Krishnan and L. S. Chandran, "Hardness of approximation results for the problem of finding the stopping distance in Tanner graphs," in *International Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*. Springer, 2006, pp. 69–80.

[33] A. McGregor and O. Milenkovic, "On the hardness of approximating stopping and trapping sets," *IEEE Transactions on Information Theory*, vol. 56, no. 4, pp. 1640–1650, 2010.

[34] C.-C. Wang, S. R. Kulkarni, and H. V. Poor, "Finding all small error-prone substructures in LDPC codes," *IEEE Transactions on Information Theory*, vol. 55, no. 5, pp. 1976–1999, 2009.

[35] E. Rosnes and Ø. Ytrehus, "An efficient algorithm to find all small-size stopping sets of low-density parity-check matrices," *IEEE Transactions on Information Theory*, vol. 55, no. 9, pp. 4167–4178, 2009.

[36] X. Zhang and P. H. Siegel, "Efficient algorithms to find all small error-prone substructures in LDPC codes," in *IEEE Global Telecommunications Conference (GLOBECOM)*, 2011, pp. 1–6.

[37] A. Sarıduman, A. E. Pusane, and Z. C. Taşkın, "An integer programming-based search technique for error-prone structures of LDPC codes," *International Journal of Electronics and Communications*, vol. 68, no. 11, pp. 1097–1105, 2014.

[38] K. M. Greenan, "Reliability and power-efficiency in erasure-coded storage systems," Ph.D. dissertation, University of California, Santa Cruz, 2009.

[39] D. Ford, F. Labelle, F. I. Popovici, M. Stokely, V.-A. Truong, L. Barroso, C. Grimes, and S. Quinlan, "Availability in globally distributed storage systems," in *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2010, pp. 1–7.

[40] M. Sathiamoorthy, M. Asteris, D. Papailiopoulos, A. G. Dimakis, R. Vadali, S. Chen, and D. Borthakur, "XORing elephants: novel erasure codes for big data," in *Proceedings of the VLDB Endowment*, vol. 6, 2013, pp. 325–336.

[41] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, and S. Yekhanin, "Erasure Coding in Windows Azure Storage," in *USENIX Annual Technical Conference*, 2012, pp. 15–26.

[42] J. L. Hafner and K. Rao, "Notes on reliability models for non-MDS erasure codes," *IBM Res. rep. RJ10391*, 2006.

[43] S. I. Resnick, *Adventures in stochastic processes*. Springer Science & Business Media, 2013.

[44] S. Nath, H. Yu, P. B. Gibbons, and S. Seshan, "Subtleties in tolerating correlated failures in wide-area storage systems," in *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, vol. 6, 2006, pp. 225–238.

[45] R. Rodrigues and B. Liskov, "High availability in DHTs: Erasure coding vs. replication," in *International Workshop on Peer-to-Peer Systems (IPTPS)*. Springer, 2005, pp. 226–239.

[46] A. G. Dimakis, P. B. Godfrey, Y. Wu, M. J. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," *IEEE Transactions on Information Theory*, vol. 56, no. 9, pp. 4539–4551, 2010.

[47] R. Bhagwan, S. Savage, and G. M. Voelker, "Understanding availability," in *International Workshop on Peer-to-Peer Systems (IPTPS)*. Springer, 2003, pp. 256–267.

[48] K. M. Greenan, J. S. Plank, J. J. Wylie *et al.*, "Mean Time to Meaningless: MTTDL, Markov Models, and Storage System Reliability," in *HotStorage*, 2010, pp. 1–5.

[49] Accessed on January 17th, 2018. [Online]. Available: http://www.kaymgee.com/KevinGreenan/Software.html

[50] K. M. Greenan, E. L. Miller, and J. J. Wylie, "Reliability of flat XOR-based erasure codes on heterogeneous devices," in *IEEE International Conference on Dependable Systems and Networks (DSN)*, 2008, pp. 147–156.

[51] K. Rashmi, P. Nakkiran, J. Wang, N. B. Shah, and K. Ramchandran, "Having Your Cake and Eating It Too: Jointly Optimal Erasure Codes for I/O, Storage, and Network-bandwidth," in *USENIX Conference on File and Storage Technologies (FAST)*, 2015, pp. 81–94.

[52] N. B. Shah, K. V. Rashmi, P. V. Kumar, and K. Ramchandran, "Distributed storage codes with repair-by-transfer and nonachievability of interior points on the storage-bandwidth tradeoff," *IEEE Transactions on Information Theory*, vol. 58, no. 3, pp. 1837–1852, 2012.

[53] A. G. Dimakis, K. Ramchandran, Y. Wu, and C. Suh, "A survey on network codes for distributed storage," *Proceedings of the IEEE*, pp. 476–489, 2011.

[54] V. Guruswami and M. Wootters, "Repairing Reed-Solomon Codes," *IEEE Transactions on Information Theory*, vol. 63, pp. 5684–5698, 2017.

[55] K. V. Rashmi, N. B. Shah, and P. V. Kumar, "Optimal exact-regenerating codes for distributed storage at the MSR and MBR points via a product-matrix construction," *IEEE Transactions on Information Theory*, vol. 57, no. 8, pp. 5227–5239, 2011.

[56] D. S. Papailiopoulos and A. G. Dimakis, "Locally Repairable Codes," *IEEE Transactions on Information Theory*, vol. 60, no. 10, pp. 5843–5855, 2014.

[57] C. Huang, M. Chen, and J. Li, "Pyramid Codes: Flexible Schemes to Trade Space for Access Efficiency in Reliable Data Storage Systems," *ACM Transaction on Storage*, vol. 9, pp. 1–28, 2013.

[58] I. Tamo and A. Barg, "A family of optimal locally recoverable codes," *IEEE Transactions on Information Theory*, vol. 60, no. 8, pp. 4661–4676, 2014.

[59] I. Tamo, D. S. Papailiopoulos, and A. G. Dimakis, "Optimal locally repairable codes and connections to matroid theory," *IEEE Transactions on Information Theory*, vol. 62, pp. 6661–6671, 2016.

[60] L. Pamies-Juarez, F. Blagojevic, R. Mateescu, C. Guyot, E. E. Gad, and Z. Bandic, "Opening the Chrysalis: On the Real Repair Performance of MSR Codes," in *USENIX Conference on File and Storage Technologies (FAST)*, 2016, pp. 81–94.

[61] P. Gopalan, C. Huang, H. Simitci, and S. Yekhanin, "On the locality of codeword symbols," *IEEE Transactions on Information Theory*, vol. 58, no. 11, pp. 6925–6934, 2012.

[62] N. Prakash, G. M. Kamath, V. Lalitha, and P. V. Kumar, "Optimal linear codes with a local-error-correction property," in *IEEE International Symposium on Information Theory (ISIT)*, 2012, pp. 2776–2780.

[63] F. Oggier and A. Datta, "Self-repairing homomorphic codes for distributed storage systems," in *IEEE International Conference on Computer Communications (INFOCOM)*, 2011, pp. 1215–1223.

[64] S. Jiekak, A.-M. Kermarrec, N. Le Scouarnec, G. Straub, and A. Van Kempen, "Regenerating codes: A system perspective," *ACM SIGOPS Operating Systems Review*, vol. 47, no. 2, pp. 23–32, 2013.

[65] N. B. Shah, K. Rashmi, P. V. Kumar, and K. Ramchandran, "Interference alignment in regenerating codes for distributed storage: Necessity and code constructions," *IEEE Transactions on Information Theory*, vol. 58, no. 4, pp. 2134–2158, 2012.

[66] Accessed on February 10th, 2019. [Online]. Available: https://github.com/rashmikv/repair-by-transfer-product-matrix-codes

[67] K. Rashmi, N. B. Shah, D. Gu, H. Kuang, D. Borthakur, and K. Ramchandran, "A "Hitchhiker's" Guide to Fast and Efficient Data Reconstruction in Erasure-coded Data Centers," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 331–342, 2014.

[68] K. Rashmi, N. B. Shah, and K. Ramchandran, "A piggybacking design framework for read-and download-efficient distributed storage codes," in *IEEE International Symposium on Information Theory (ISIT)*, 2013, pp. 331–335.

[69] X.-Y. Hu, E. Eleftheriou, and D.-M. Arnold, "Regular and irregular progressive edge-growth tanner graphs," *IEEE Transactions on Information Theory*, vol. 51, no. 1, pp. 386–398, 2005.

[70] M. Woitaszek and H. M. Tufo, "Tornado codes for maid archival storage," in *Conference on Mass Storage Systems and Technologies (MSST)*, 2007, pp. 221–226.

[71] H. Mercier, M. Augier, and A. K. Lenstra, "STeP-Archival: Storage Integrity and Tamper Resistance Using Data Entanglement," *IEEE Transactions on Information Theory*, vol. 64, no. 6, pp. 4233–4258, 2018.

[72] K. V. Rashmi, N. B. Shah, D. Gu, H. Kuang, D. Borthakur, and K. Ramchandran, "A Solution to the Network Challenges of Data Recovery in Erasure-coded Distributed Storage Systems: A Study on the Facebook Warehouse Cluster," in *HotStorage*, 2013.

[73] LRC Erasure Coding in Windows Storage Spaces, accessed on February 10th, 2019. [Online]. Available: http://research.microsoft.com/en-us/um/people/chengh/slides

[74] Reed-Solomon open-source implementation, accessed on February 10th, 2019. [Online]. Available: https://github.com/tomerfiliba/reedsolomon

[75] F. Mattoussi, V. Roca, and B. Sayadi, "Complexity comparison of the use of Vandermonde versus Hankel matrices to build systematic MDS Reed-Solomon codes," in *IEEE International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, 2012, pp. 344–348.

[76] F. Oggier, A. Datta *et al.*, "Coding techniques for repairability in networked distributed storage systems," *Foundations and Trends in Communications and Information Theory*, vol. 9, no. 4, pp. 383–466, 2013.

[77] M. Bakkaloglu, J. J. Wylie, C. Wang, and G. R. Ganger, "On correlated failures in survivable storage systems," Carnegie Mellon University, Tech. Rep., 2002. [Online]. Available: http://repository.cmu.edu/pdl/107/

[78] D. Gelernter and N. Carriero, "Coordination languages and their significance," *Communications of the ACM*, vol. 35, pp. 97–107, 1992.

[79] L. I. Patterson, R. S. Turner, and R. M. Hyatt, "Construction of a fault-tolerant distributed tuple-space," in *ACM/SIGAPP Symposium on Applied computing: states of the art and practice (SAC)*, 1993, pp. 279–285.

[80] A. N. Bessani, M. Correia, J. da Silva Fraga, and L. C. Lung, "An efficient byzantine-resilient tuple space," *IEEE Transactions on Computers*, vol. 58, no. 8, pp. 1080–1094, 2009.

[81] A. N. Bessani, E. P. Alchieri, M. Correia, and J. S. Fraga, "DepSpace: a Byzantine fault-tolerant coordination service," in *ACM SIGOPS EuroSys European Conference on Computer Systems*, vol. 42, no. 4, 2008, pp. 163–176.

[82] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop Distributed File System," in *IEEE Mass storage systems and technologies (MSST)*, 2010, pp. 1–10.

[83] T. White, *Hadoop: The definitive guide*. O'Reilly Media Inc., 2012.

[84] M. Y. Eltabakh, Y. Tian, F. Özcan, R. Gemulla, A. Krettek, and J. McPherson, "CoHadoop: flexible data placement and its exploitation in Hadoop," *Proceedings of the VLDB Endowment*, vol. 4, no. 9, pp. 575–585, 2011.

[85] H. Jin, X. Yang, X.-H. Sun, and I. Raicu, "ADAPT: Availability-Aware MapReduce Data Placement for Non-dedicated Distributed Computing," in *IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2012, pp. 516–525.

[86] D. Gelernter, "Generative communication in Linda," *ACM Transactions on Programming Languages and Systems*, vol. 7, no. 1, pp. 80–112, 1985.

[87] N. Carriero and D. Gelernter, *How to Write Parallel Programs: A First Course*. MIT Press, 1990.

[88] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The Many Faces of Publish/Subscribe," *ACM Computing Surveys*, vol. 35, no. 2, pp. 114–131, 2003.

[89] V. Buravlev, R. De Nicola, and C. A. Mezzina, "Tuple spaces implementations and their efficiency," in *International Conference on Coordination Models and Languages (COORDINATION)*. Springer, 2016, pp. 51–66.

[90] P. Institute, "2013 cost of data center outages," 2013.

[91] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An introduction to statistical learning*. Springer, 2013, vol. 6.

[92] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, no. 5439, pp. 509–512, 1999.

[93] S. Knight, H. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The internet topology zoo," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 9, pp. 1765–1775, 2011.

[94] C. C. Robusto, "The Cosine-Haversine Formula," *The American Mathematical Monthly*, vol. 64, no. 1, pp. 38–40, 1957.

[95] Intel ISA-L. Accessed on February 10th, 2019. [Online]. Available: https://github.com/01org/isa-l

[96] A. D. R. Marc Waldman and L. F. Cranor, "Publius: A robust, tamper-evident, censorship-resistant, web publishing system," in *USENIX Security Symposium*, 2000, pp. 59–72.

[97] R. Dingledine, M. J. Freedman, and D. Molnar, "The Free Haven Project: Distributed Anonymous Storage Service," *Designing Privacy Enhancing Technologies*, pp. 67–95, 2001.

[98] A. Stubblefield and D. S. Wallach, "Dagster: Censorship-Resistant Rublishing Without Replication," Rice University, Tech. Rep. TR01-380, 2002.

[99] M. Waldman and D. Mazières, "Tangler: A Censorship-resistant Publishing System Based on Document Entanglements," in *ACM Conference on Computer and Communications Security (CCS)*, 2001, pp. 126–135.

[100] Accessed on February 10th, 2019. [Online]. Available: https://www.dropbox.com/help/security/legal-requests

[101] R. Anderson, "The Eternity Service," in *International Conference on the Theory and Applications of Cryptography (PRAGOCRYPT)*, vol. 96, 1996, pp. 242–252.

[102] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong, "Freenet: A Distributed Anonymous Information Storage and Retrieval System," *Designing Privacy Enhancing Technologies*, pp. 46–66, 2001.

[103] R. Kotla, L. Alvisi, and M. Dahlin, "SafeStore: A Durable and Practical Storage System," in *USENIX Annual Technical Conference*, 2007, pp. 129–142.

[104] L. L. You, K. T. Pollack, and D. D. Long, "Deep Store: an archival storage system architecture," in *IEEE International Conference on Data Engineering (ICDE)*, 2005, pp. 804–815.

[105] I. Clarke, O. Sandberg, M. Toseland, and V. Verendel, "Private Communication Through a Network of Trusted Connections: The Dark Freenet," *Network*, 2010.

[106] Freenet. Accessed on February 10th, 2019. [Online]. Available: https://freenetproject.org

[107] S. Rhea, P. Eaton, D. Geels, H. Weatherspoon, B. Zhao, and J. Kubiatowicz, "Pond: The OceanStore Prototype," in *USENIX Conference on File and Storage Technologies (FAST)*, 2003, pp. 1–14.

[108] OceanStore. [Online]. Available: http://www.oceanstore.net

[109] M. W. Storer, K. M. Greenan, E. L. Miller, and K. Voruganti, "POTSHARDS: Secure Long-Term Storage Without Encryption," in *USENIX Annual Technical Conference*, 2008, pp. 1–14.

[110] Publius. Accessed on February 10th, 2019. [Online]. Available: http://www.cs.nyu.edu/~waldman/publius

[111] RECAST Source Code. Accessed on February 10th, 2019. [Online]. Available: https://github.com/safecloud-project/recast

[112] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, pp. 612–613, 1979.

[113] R. Geambasu, A. A. Levy, T. Kohno, A. Krishnamurthy, and H. M. Levy, "Comet: An active distributed key-value store," in *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2010, pp. 323–336.

[114] S. Hand and T. Roscoe, "Mnemosyne: Peer-to-Peer Steganographic Storage," in *International Workshop on Peer-to-Peer Systems (IPTPS)*. Springer, 2002, pp. 130–140.

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

[116] L. L. You, K. T. Pollack, D. D. Long, and K. Gopinath, "PRESIDIO: A Framework for Efficient Archival Data Storage," *ACM Transactions on Storage (TOS)*, vol. 7, pp. 1–60, 2011.

[117] P. Maniatis, M. Roussopoulos, T. J. Giuli, D. S. Rosenthal, and M. Baker, "The LOCKSS peer-to-peer digital preservation system," *ACM Transactions on Computer Systems (TOCS)*, vol. 23, pp. 2–50, 2005.

[118] J. C. Frank, S. M. Frank, L. A. Thurlow, T. M. Kroeger, E. L. Miller, and D. D. Long, "Percival: A searchable secret-split datastore," in *Symposium on Mass Storage Systems and Technologies (MSST)*, 2015, pp. 1–12.

[119] D. Bhagwat, K. Pollack, D. D. Long, T. Schwarz, E. L. Miller, and J.-F. Pâris, "Providing high reliability in a minimum redundancy archival storage system," in *IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2006, pp. 413–421.

[120] A. J. Felstrom and K. S. Zigangirov, "Time-varying periodic convolutional codes with low-density parity-check matrix," *IEEE Transactions on Information Theory*, pp. 2181–2191, 1999.

[121] Facebook. Accessed on November 1st, 2018. [Online]. Available: https://code.fb.com/core-data/saving-capacity-with-hdfs-raid

[122] J. S. Plank, A. L. Buchsbaum, R. L. Collins, and M. G. Thomason, "Small parity-check erasure codes-exploration and observations," in *IEEE International Conference on Dependable Systems and Networks (DSN)*, 2005, pp. 326–335.

[123] S. Kudekar, T. J. Richardson, and R. L. Urbanke, "Threshold saturation via spatial coupling: Why convolutional LDPC ensembles perform so well over the BEC," *IEEE Transactions on Information Theory*, pp. 803–834, 2011.

[124] A. E. Pusane, R. Smarandache, P. O. Vontobel, and D. J. Costello, "Deriving good LDPC convolutional codes from LDPC block codes," *IEEE Transactions on Information Theory*, pp. 835–857, 2011.

[125] A. E. Pusane, A. J. Feltstrom, A. Sridharan, M. Lentmaier, K. S. Zigangirov, and D. J. Costello, "Implementation aspects of LDPC convolutional codes," *IEEE Transactions on Communications*, pp. 1060–1069, 2008.

[126] R. M. Tanner, D. Sridhara, A. Sridharan, T. E. Fuja, and D. J. Costello, "LDPC block and convolutional codes based on circulant matrices," *IEEE Transactions on Information Theory*, pp. 2966–2984, 2004.

[127] B. P. Smith, A. Farhood, A. Hunt, F. R. Kschischang, and J. Lodge, "Staircase codes: FEC for 100 Gb/s OTN," *IEEE Journal of Lightwave Technology*, vol. 30, no. 1, pp. 110–117, 2012.