Coding Techniques for Repairability in Networked Distributed Storage Systems

Frédérique Oggier
Division of Mathematical Sciences
School of Physical and Mathematical Sciences
Nanyang Technological University
Singapore
E-mail:frederique@ntu.edu.sg

Anwitaman Datta
Division of Computer Science
School of Computer Engineering
Nanyang Technological University
Singapore
E-mail:anwitaman@ntu.edu.sg

September 18, 2012

Coding Techniques for Repairability in Networked Distributed Storage Systems

Frédérique Oggier and Anwitaman Datta

Keywords

Distributed Storage Systems, Erasure Codes, Repair

Abstract

This manuscript comprises a tutorial on traditional erasure codes and their applications to networked distributed storage systems (NDSS), followed by a survey of novel code families tailor made for better repairability in NDSS.

Preface

Portions of this survey, particularly the second part, was originally written as personal notes when we started to work on this topic, as an attempt to understand the big picture. The big picture was accordingly summarized at a very high level in a short survey [6]. The tutorial part on NDSS and coding theory was added later, together with one code construction that we proposed, when these personal notes became lecture notes that were provided for the Open Phd program at Warsaw University and presented at a tutorial in ICDCN 2012. The current version is an updated version of these lecture notes, including technical details and taking into account some recent developments, as well as providing background context to make the manuscript self-contained.

Contents

Part I

Background

1

Introduction

When communicating over an erasure channel, a transmitter typically adds redundancy to the message to be sent, so that the intended recipient can reconstruct the original message despite the loss of parts of the transmitted data. The mapping from the original message to its redundant version is referred to as encoding, and the challenge is to design an efficient coding scheme, that provides good protection against erasures at a low overhead.

Analogous problems arise in the context of data storage. Damages to the physical storage medium may make some bits/bytes unreadable and redundancy is needed to protect the stored data. For instance, a compact disc (CD) can often tolerate scratches thanks to the presence of a suitable coding technique, called Reed-Solomon codes [33]. Another example at the other end of the size spectrum of storage systems is a large-scale distributed system such as a data-center or a peer-to-peer system with many storage devices, some of which may fail or become inaccessible due to network problems. Redundancy is again needed for fault tolerance, so that the aggregate data stored in the system can be retrieved. Though coding is a way of handling failures in the aforementioned scenarios, the design of a good code naturally depends on the peculiarities of the setting considered - thus, codes for magnetic medium, solid state devices, CD, disk arrays or distributed systems may aim for distinct desirable properties.

The most commonly deployed multi-storage device systems are RAID (Redundant Array of Independent/Inexpensive Disks) systems [29], which store the data across multiple disks, some of which containing the actual information, while the others provide fault-tolerance by storing redundancy. Furthermore, distributing the data over multiple storage disks may also help increase the throughput of reading data, thanks to the parallelization of disk accesses. RAID systems traditionally put the multiple storage disks within a single computing unit, making the internal distribution transparent both logically as well as physically for the end users. Currently, typical RAID configurations allow for two failures within a RAID unit, though configurations tolerating more failures have also been studied.

The idea of distributing data across multiple disks has been naturally extended to multiple storage nodes which are interconnected over a network, as we witness in data-centers, and some peer-to-peer (P2P) storage systems. We call such systems networked distributed storage systems (NDSS), where the word "networked" insists on the importance of the network interconnect. It is worth recalling that the individual storage nodes in an NDSS may themselves be comprised of multi-disk RAID systems, whose storage disks may themselves employ some redundancy scheme for fault-tolerance of their physical medium. Thus, while redundancy is present at several layers of a large storage system, this survey only looks at redundancy through coding techniques at the highest level of abstraction, namely for NDSS - and do so in a manner agnostic of the lower layer details.

At the NDSS level, data stored in individual storage nodes may become unavailable due to various reasons. As pointed out earlier, either a storage node or the communication link to this node may fail, but these are not the only cases. In peer-to-peer settings, a user operating a storage node may just decide to make it offline temporarily, or leave the system permanently. Irrespective of the nature of the failure, redundancy is needed to ensure data availability. Depending on the nature of failure, the lost redundancy may also need to be replenished in order to ensure long term data durability. The simplest form of coding, namely replication, has been and still is a popular way to ensure redundancy in NDSS, due to its simplicity. However, given the sheer volume of data that needs to be stored and the overheads of replication, there has been in recent years an immense interest in using coding for NDSS among major commercial players such as Google and Microsoft [18] to name a few, an interest which has also been mirrored in the academic world.

The aim of this survey is to look at coding techniques for NDSS, which aim at achieving (1) fault tolerance efficiently and (2) good repairability characteristics to replenish the lost redundancy, and ensure data durability over time. We will like to make the following disclaimer about the scope of this survey. There are many other criteria (than repair) that may guide the design of codes for NDSS. There are also many other kind of performance issues (than repair) that still need to be studied for many of the codes that we summarize in this survey. We will however confine our discussions mainly to codes providing good repairability. Also, while we have tried to provide an overview of the most prominent code techniques representing different points in the code design space, our treatment of the subject is by no means exhaustive. We have both deliberately as well as out of our ignorance given the rapid pace of developments in the area, left out many works.

This survey is organized in two parts. The first part gives an overview of some basic concepts related to networked distributed storage systems (NDSS) and provides a quick introduction to classical coding theory, concluding with a discussion of the pros and cons of using classical erasure codes for NDSS. Such a discussion leads us to the second part, where several new families of codes tailor made for NDSS repairability are described and reviewed. Since it is impossible to keep track of every single code construction proposed, we instead identify prominent design choices, which are described and illustrated respectively in Chapter 6 for a network coding approach, in Chapter 7 for combining two layers of erasure codes, and in Chapter 8 for codes aiming at local repairability.

2

Networked Distributed Storage Systems

Networked distributed storage systems (NDSS) refer to systems that store data in a distributed manner across a computer network, where the interconnect plays an important role. NDSS are commonplace nowadays, they operate in several environments such as peer-to-peer (P2P) systems and data centers that comprise the backbone infrastructure of cloud computing. They consequently cater to very different applications and workloads.

In P2P systems, the storage servers are run by different users. P2P backup systems (e.g. the former version of Wuala [37]) are formed by swarms of tens to hundreds of nodes for individual files or directories, but may distribute such swarms arbitrarily out of hundreds of thousands of peers. P2P systems are geographically distributed and connected through an arbitrary topology. Individual P2P nodes may frequently go offline and come back online (known as temporary churn) unpredictably, subject to the whims of individual storage node owners.

In contrast, data centers comprise of tens of thousands of nodes, while individual clusters such as that of Google File System (GFS) are formed out of hundreds up to thousands of nodes. Individual data objects are spread over tens of nodes from within a cluster. The nature of applications varies a lot, and may not only be storage and input/output (I/O), i.e. write/read intensive (e.g., file storage and video streaming), but also computation intensive (e.g., Hadoop tasks [14]). Data center interconnects have well defined topologies and storage nodes are either collocated or distributed across a very few geographic regions. They use dedicated infrastructures with relatively infrequent temporary outages. However, nodes may also be turned off/on temporarily in a planned manner, for instance for maintenance and upgrades or saving energy .

Despite their relatively different size, very particular topology and network dynamics, both data centers and peer-to-peer storage systems however share common characteristics: (1) the volume of data to be stored is immense and hence it is distributed over many nodes, (2) failures are a norm rather than an exception, and there is a need for fault-tolerance strategies, (3) this same fault-tolerance has to be ensured over time, requiring in turn efficient maintenance mechanisms, (4) apart the distinguishing commonality among NDSS that interconnect plays an important role (hence the qualifying term 'networked') in the distribution of data across many storage nodes. We next elaborate these three first issues.

2.1 Scaling-up vs Scaling-out

P2P storage systems are naturally distributed. However 'scaling-out' is an inevitable design choice in data centers. The term scaling-out refers to the fact that many logical storage units (or simply 'storage nodes') are interconnected together to increase the overall capacity of the system. We note that a logical storage unit in itself may be internally implemented by composing several storage components (hard disks), for instance as in RAID systems [29].

As opposed to scaling-out, a hypothetical alternative would be to 'scale-up', that is building a single piece of special hardware with adequate storage capacity. However, the demand for storage is astonishing. For instance, back in 2010, Facebook was reportedly storing over 260 billion images, which translated to over 20 petabytes of data [2]. Users were then uploading one billion new photos (60 terabytes) each week. If at all a single piece of hardware could be built to store all that data, not only would it be prohibitively expensive, but when many of the stored data would be accessed at the same time, the limited input/output (I/O) resources of the hardware would be a significant bottleneck. Facebook served over one million images per second at peak demand in 2010. Furthermore, as the amount of data grows over time, one would need to frequently move to a new device with much larger capacity, and in the case of a failure, all the data would be lost. Scaling-out on the contrary has already proven to be a feasible option since the system can grow in an organic manner by incrementing the number of commodity storage nodes over time, while only a small subset of the system fails and gets replaced at any specific time.

2.2 Redundancy and Storage Overhead

With the number of its components increasing (storage nodes, but also routers, network, power supply, cooling, etc.), an NDSS ends up having a significant (even if small) subset of these components not functioning properly at almost any time instance. Thus, fault-tolerance to make the overall system and its services transparent from the underlying faults is essential. This is achieved by the addition of redundancy.

Redundancy is achieved in many different manners. For instance, in data centers, auxiliary batteries are used for power supply to racks. Multiple (typically two) switches are provided per rack for connectivity. Likewise the interconnect itself has multiple point-to-point routing options to tolerate faults and avoid congestions. Similarly, the data is replicated to deal with the outage of the storage nodes themselves, typically in storage nodes in different racks to tolerate the failure of a complete rack. In some environments, the data is further redundantly stored in multiple data centers so that it remains available even if a whole data center becomes unavailable. Amazon AWS introduced the concept of 'availability zones' where data centers across different availability zones utilize independent physical infrastructure, such as power or water supply, so that the faults can be decoupled. The data centers can in turn also be distributed across multiple geographic regions, in order to tolerate geographically localized catastrophes.

In a blog from O'Reilly,[1] a further level of redundancy is suggested for end users of cloud services, namely that of using different service providers, and hence insulating oneself from the failings - technical or commercial - of the individual service providers.

We note that realizing most of the above kinds of redundancy inherently assumes storing the data itself redundantly. Replication is a simple way of doing so. A simple rule of thumb is *3-way replication* (a more general variation being *r-way replication*), where the rationale is that if one of the replica becomes unavailable, one can recreate a new replica using the remaining two, with the optimism that not both of the remaining replicas fail while the replenishment of redundancy is being carried out. This brings up two interesting issues pertaining to distributed storage systems: how to trade redundancy and fault tolerance, which we discuss below, and how to handle the maintenance mechanisms if we move away from replication, which is the topic of the next section.

Clearly, the more redundancy is used, the more fault-tolerant the storage system becomes, but there is a price to pay: redundancy increases the overheads of the storage infrastructure. The cost for such an infrastructure should be estimated not only in terms of the hardware, but also of real estate, operation and maintenance of a data center. A US Environmental Protection Agency report of 2007[2] indicates that the US used 61 billion kilowatt-hours of power for data centers and servers in 2006. That is 1.5 percent of the US electricity use, and it costs the companies that paid those bills more than $4.5 billion. Data to be stored is not reducing over time, and a study sponsored by the information storage company *EMC* estimated that the world's data is more than doubling ev-

[1] http://broadcast.oreilly.com/2011/04/the-aws-outage-the-clouds-shining-moment.html

[2] http://arstechnica.com/old/content/2007/08/epa-power-usage-in-data-centers-could-double-by-2011.ars

ery two years, reaching 1.8 zettabytes (1 zettabyte = 10^{21} bytes) of data to be stored in 2011.[3]

Given the volume of data concerned and the associated cost of storing it reliably, a first important issue is, "is there any other strategy which can achieve redundancy at least as effectively as replication of the data, but more efficiently in terms of storage overheads?"

Coding techniques (see next chapter), long studied in the context of robust communication over unreliable channels, readily provides a solution to this issue. Not surprisingly, major industry players like Google and Microsoft have adopted the use of erasure codes in the new Google File System and Windows Azure [18] respectively. Other storage services like CleverSafe [5] and Wuala [37] have been using erasure codes in their systems for longer.

2.3 Maintenance of Redundancy

We noticed above that the rationale behind 3-way replication is that in case of a failure involving a copy, one of the two copies left can be used to recreate the missing one. This illustrates the need for recreation of lost redundancy. If a new copy is not obtained, the next failure will let the data with no fault tolerance at all, and this data will eventually become unavailable or lost.

In order to maintain a desirable amount of redundancy in the system over time, three important functionalities are required. Foremost, **failures need to be detected correctly**. Delay in detection naturally delays the repair process, inadvertently leaving the system vulnerable. Repairs triggered due to false positives, i.e., concluding that a node has failed when it in fact has not (for instance if communication is delayed due to network congestion) may however aggravate the situation. They may cause further congestions and cascading failures (for instance, due to positive feedback based amplification of congestion). Monitoring and detecting failures properly in large scale systems is thus an issue of its own

[3] `http://www.emc.com/about/news/press/2011/20110628-01.htm`

right, and is beyond the scope of our study.

Once some failures are detected, the next question is **whether and when to carry out repairs**. In peer-to-peer systems, nodes frequently go offline to come back online later, and while a relatively higher amount of initial redundancy is essential to deal with such temporary 'failures', repairs can be avoided by waiting. Eventually, if repairs need to be carried out, the cost of multiple repairs are also likewise amortized. In data-center settings, a more proactive approach to repair may be the norm, nonetheless, there are circumstances where repairs are either delayed in order to avoid positive feedbacks,[4] or multiple faults may accumulate due to temporally correlated failures which occur faster than the repair process. The associated repair time and load at the live nodes facilitating the repairs are other important characteristics. In summary, whether and when to trigger repairs is a system design choice, however either as a feature to tolerate voluntarily delayed repair, or as a safety mechanisms in case of correlated failures, the ability to repair multiple faults simultaneously is desirable.

The final functionality is **the repair process itself**. That is the primary focus of this survey. Suppose that we have appropriate mechanisms to detect failures and trigger repairs, how is repair performed? In the case of replication, copies of the data are downloaded from live nodes. What happens if replicas are replaced by erasure codes is less obvious. Traditional erasure codes were designed for robust communication, where the objective is to retrieve the original message. From the perspective of storage systems, this is equivalent to *data retrieval* or *data reconstruction* for accessing the stored object. Therefore as long as adequate redundancy is present in the system, from an end-user or application perspective, it is adequate. Thus a long believed mantra when applying codes for storage has been that '*the storage device is the erasure channel*'. Such a simplification however ignores the maintenance

[4] Positive feedback induced cascading failures have been witnessed in practice, for instance the case Amazon Elastic Block Store's outage in April 2011 `http://storagemojo.com/2011/04/29/amazons-ebs-outage/`

process in NDSS for long term reliability.

This realization has led to a renewed interest in designing codes tailor-made for NDSS, looking at better/more efficient strategies to carry out the redundancy replenishment. In order to pursue this discussion in more precise terms, the next chapter recalls a few basic concepts from erasure codes, which will then allow us to analyze concretely how these erasure codes coming from communication can be used for storage applications.

3

Coding Preliminaries

This chapter presents some basic concepts and techniques from classical coding theory, such as maximal distance separable (MDS) codes, and Reed-Solomon codes, so as to discuss erasure codes for storage applications in precise terms. Whenever needed, we will introduce the necessary mathematical tools. In particular, we will describe how to construct finite fields. For now, let us denote the binary alphabet $\{0, 1\}$ by $\mathbb{F}_2$. The index 2 refers to the cardinality of the set, while the letter $\mathbb{F}$ stands for field. It basically means that our usual arithmetic works: addition, subtraction, multiplication which is commutative, and division by a non-zero element. Since we only have 0 and 1, operations are performed modulo 2: $0+0 = 0$, $0 + 1 = 1 + 0 = 1$, $1 + 1 = 0$, for the addition (or XOR) and $0 \cdot 1 = 1 \cdot 0 = 0$, $0 \cdot 0 = 0$ and $1 \cdot 1 = 1$, for the multiplication. By $\mathbf{u} \in \mathbb{F}_2^k$ we mean a k-dimensional row vector $\mathbf{u} = [u_1, \dots, u_k]$ with coefficients in $\mathbb{F}_2$.

3.1 Generator and Parity Check Matrix

A *code* consists of a map that sends a vector $\mathbf{u} \in \mathbb{F}_2^k$ to a vector $\mathbf{x} \in \mathbb{F}_2^n$ where $n > k$. We will only be interested in *linear codes*, that is when the map from $\mathbf{u}$ to $\mathbf{x}$ is linear, which means that $\mathbf{x}$ can be written as vector matrix multiplication:

$$\mathbf{u}G = \mathbf{x}$$

where G is a $k \times n$ matrix with coefficients in $\mathbb{F}_2$ called *generator matrix* of the code. We usually refer to $\mathbf{u}$ as a vector of information symbols, and to $\mathbf{x}$ as a codeword. The set of codewords is called *codebook*, and we will denote a code or its corresponding codebook by $\mathcal{C}$. It is standard to write a (n, k) code, to emphasize the parameters of the code $\mathcal{C}$, where n is called the length of the code, and k its dimension.

Definition 1. The ratio k/n is called the *rate* of an (n, k) code.

The rate of a code is the proportion of the transmitted data which is useful.

Example 1. The *repetition code* takes one information symbol and repeats it n times. For example, a $(3, 1)$ repetition code can be written as

$$u \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} = [u, u, u]$$

where u is either 0 or 1. Its rate is 1/3.

Example 2. Another simple linear code, sometimes called *single parity check code*, consists of appending one bit of parity, that is: take $\mathbf{u} = [u_1, \ldots, u_k]$ and add 1 bit given by the sum $u_1 + \ldots + u_k$. This $(k + 1, k)$ code can be written, if $k = 2$, as

$$[u_1, u_2] \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} = [u_1, u_2, u_1 + u_2]$$

where u_1 and u_2 are either 0 or 1. Its rate is 2/3.

By looking at these two examples, we notice that they have something in common. Both codewords actually contain the original information symbols, namely

$$[u, u, u]$$

contains u and

$$[u_1, u_2, u_1 + u_2]$$

contains $[u_1, u_2]$. This happens exactly when the generator matrix G is of the form

$$G = \begin{bmatrix} \mathbf{I}_k & A \end{bmatrix} \tag{3.1}$$

for some $k \times (n-k)$ matrix A and $\mathbf{I}_k$ is the identity matrix

$$\mathbf{I}_k = \begin{bmatrix} 1 & & \\ & \ddots & \\ & & 1 \end{bmatrix}.$$

In this case, the code is said to be *systematic.*

We have seen above how, given a vector $\mathbf{u}$ of information symbols, to create a codeword $\mathbf{x}$. The reverse question would be: given a vector $\mathbf{x} \in \mathbb{F}_2^n$, how to recognize whether it is a codeword from a given code?

Example 3. Let us take again the repetition code of Example 1. It is obvious that if we see a codeword of the form $\mathbf{x} = [u, \dots, u]$, then it is a codeword from the repetition code. This is not a mathematically satisfying answer though! In most cases, things will not be that obvious. Thus let us try to see what conditions uniquely determine a codeword $\mathbf{x} = [x_1, x_2, x_3] = [u, u, u] \in \mathbb{F}_2^3$ from the $(3, 1)$ repetition code. We notice that if we sum x_1 and x_3, or x_2 and x_3, we get 0 both times, and in fact this is enough to characterize our codeword. Indeed: if $\mathbf{x} = [x_1, x_2, x_3] = [u, u, u]$, then $x_1 + x_3 = 2u = 0$ and $x_2 + x_3 = 2u = 0$. Conversely, take any $\mathbf{x} = [x_1, x_2, x_3]$ such that $x_1 + x_3 = x_2 + x_3 = 0$. That $x_1 + x_3 = 0$ implies that $x_1 = -x_3 = x_3$ and similarly we get that $x_2 = x_3$ and it must be that $\mathbf{x} = [x_1, x_1, x_1]$. We can rewrite these conditions in a matrix form:

$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

In general, we can associate to any linear (n,k) code $\mathcal{C}$ an $(n-k)\times n$ matrix H called *parity check matrix*, with the property that

$$H\mathbf{x}^T = \mathbf{0}$$

whenever $\mathbf{x}$ belongs to the code and $\mathbf{x}^T$ denotes the transpose of $\mathbf{x}$. Let us try to understand why this is the case. Recall that

$$\mathbf{x}^T = (\mathbf{u}G)^T = \left(\mathbf{u}\begin{bmatrix}\mathbf{I}_k & A\end{bmatrix}\right)^T = \begin{bmatrix}\mathbf{I}_k \\ A^T\end{bmatrix}\mathbf{u}^T$$

where A is a $k\times(n-k)$ matrix so that

$$\begin{bmatrix}-A^T & \mathbf{I}_{n-k}\end{bmatrix}\mathbf{x}^T = \begin{bmatrix}-A^T & \mathbf{I}_{n-k}\end{bmatrix}\begin{bmatrix}\mathbf{I}_k \\ A^T\end{bmatrix}\mathbf{u}^T = \mathbf{0}$$

and the parity check matrix H can in fact be defined as

$$H = \begin{bmatrix}-A^T & \mathbf{I}_{n-k}\end{bmatrix}. \tag{3.2}$$

Example 4. Consider the following (7,4) *Hamming code*, whose 3×7 parity check matrix contains as columns all binary vectors of length 3:

$$H = \begin{bmatrix}1&0&0&1&1&0&1\\0&1&0&1&0&1&1\\0&0&1&0&1&1&1\end{bmatrix}.$$

Using (3.2) and (3.1), we know that its generator matrix G is given by

$$G = \begin{bmatrix}1&0&0&0&1&1&0\\0&1&0&0&1&0&1\\0&0&1&0&0&1&1\\0&0&0&1&1&1&1\end{bmatrix}.$$

3.2 Minimum Distance and Singleton Bound

We have seen that a linear (n,k) code consists of taking k information symbols, and adding $n-k$ symbols. These $n-k$ symbols are there for redundancy. Suppose we want to transmit the symbol u through an erasure channel, that is a communication channel that

Fig. 1: Richard Hamming (1915-1998)

will output either the input with no error or an erasure. If only u is sent, either the receiver gets u, in which case communication is perfect, or he receives nothing. By using an $(n, 1)$ repetition code, the receiver will be able to understand that u was sent as long as there are no more than $n - 1$ erasures.

Note that some communication channels introduce errors instead of erasures. This case will not be treated here, hence the codes we deal with are in fact *erasure codes*.

Our next question is thus: given a linear code with parameters (n, k), what is the best maximum possible number of erasures of arbitrary symbols a code can tolerate?

To answer this, we need the notion of *Hamming distance*, named in honor of Richard Hamming (see Fig. 1), as was the Hamming code in the above example.

Definition 2. Given two vectors $\mathbf{x}$ and $\mathbf{y}$, the *Hamming distance* between $\mathbf{x}$ and $\mathbf{y}$ is the number of coefficients in which $\mathbf{x}$ and $\mathbf{y}$ differ, which is denoted by $d(\mathbf{x}, \mathbf{y})$.

Example 5. If $\mathbf{x} = [1, 0, 0, 1]$ and $\mathbf{y} = [0, 0, 0, 1]$, then $d(\mathbf{x}, \mathbf{y}) = 1$.

Definition 3. Given a vector $\mathbf{x}$, the Hamming weight of $\mathbf{x}$ is the number of non-zero coefficients of $\mathbf{x}$, denoted by $wt(\mathbf{x})$.

Note that

$$d(\mathbf{x},\mathbf{y}) = wt(\mathbf{x}-\mathbf{y}).$$

Example 6. If $\mathbf{x} = [1,0,0,1]$ and $\mathbf{y} = [0,0,0,1]$, then $wt(\mathbf{x}) = 2$, $wt(\mathbf{y}) = 1$ and $\mathbf{x}-\mathbf{y} = [1,0,0,0]$ so that $wt(\mathbf{x}-\mathbf{y}) = d(\mathbf{x},\mathbf{y}) = 1$.

Definition 4. The *minimum (Hamming) distance* d_H of a code $\mathcal{C}$ is the minimum Hamming distance between any two distinct codewords, namely

$$d_H(\mathcal{C}) = \min_{\mathbf{x}\neq\mathbf{y}\in\mathcal{C}} d(\mathbf{x},\mathbf{y}) = \min_{\mathbf{x}\neq\mathbf{y}\in\mathcal{C}} wt(\mathbf{x}-\mathbf{y}).$$

When the minimum distance of a code needs to be emphasized, d_H is sometimes written explicitly as a code parameter, namely we speak of an (n,k,d) code $\mathcal{C}$ to mention that $d_H(\mathcal{C}) = d$.

In fact, when $\mathcal{C}$ is a linear code, it is not needed to check every pair of codewords, it suffices to compute

$$d_H(\mathcal{C}) = \min_{\mathbf{x}\neq\mathbf{0},\mathbf{x}\in\mathcal{C}} wt(\mathbf{x}),$$

since the difference of any two codewords is again a codeword. Let us go back to our two examples.

Examples 7. Consider the three codes we have seen so far.

(1) For the repetition code of Example 1, there are two codewords $[0,0,0]$ and $[1,1,1]$ and thus $d_H(\mathcal{C}) = 3$.
(2) For the code of Example 2, codewords are of the form (u_1,u_2,u_1+u_2), there are thus 4 of them: $[0,0,0],[0,1,1],[1,0,1],[1,1,0]$, and $d_H(\mathcal{C}) = 2$.
(3) For the Hamming code of Example 4, we need to compute the generic form of a codeword first:

$$[u_1,u_2,u_3,u_4]\begin{bmatrix}1&0&0&0&1&1&0\\0&1&0&0&1&0&1\\0&0&1&0&0&1&1\\0&0&0&1&1&1&1\end{bmatrix}$$

$$= [u_1,u_2,u_3,u_4,u_1+u_2+u_4,u_1+u_3+u_4,u_2+u_3+u_4],$$

from which we see that $d_H(\mathcal{C}) = 3$ (it is achieved by picking for example $u_1 = 1$, $u_2 = u_3 = u_4 = 0$, and having only 2 non-zero coefficients is not possible).

We can now link the capability of a code to tolerate erasures to its minimum distance. Suppose a codeword $\mathbf{x} = [x_1, \dots, x_n]$ of length n suffers from erasures at arbitrary positions. How many can be recovered? The codeword can be recovered as long as there is no doubt which of them was sent, that is, even if some coordinates are erased, the codewords are still distinguishable.

Example 8. Consider the single parity check code, which contains as codewords

$$[0, 0, 0], [0, 1, 1], [1, 0, 1], [1, 1, 0].$$

If say one coefficient is erased, we get

$$[0, *, 0], [0, *, 1], [1, *, 1], [1, *, 0] \text{ or } [*, 0, 0], [*, 1, 1], [*, 0, 1], [*, 1, 0]$$

or

$$[0, 0, *], [0, 1, *], [1, 0, *], [1, 1, *].$$

In all the three cases, there is no doubt which codeword was sent. Of course, if two coefficients are erased, then for example we have

$$[*, *, 0], [*, *, 1], [*, *, 1], [*, *, 0]$$

and if we get $[*, *, 1]$, there is no way to know whether $[0, 1, 1]$ or $[1, 0, 1]$ was sent.

If a code $\mathcal{C}$ has minimum distance $d_H(\mathcal{C}) = d$, then it can support $d - 1$ erasures. The reason is as mentioned above: since every two codewords differ in at least d positions, as long as not more than $d - 1$ positions are erased, it is possible to distinguish them.

The minimum distance is related to the parity check matrix as explained below.

Theorem 1. If H is the parity check matrix of a code $\mathcal{C}$ of length n, then $d_H(\mathcal{C}) = d$ if and only if every $d-1$ columns of H are linearly independent and some d columns are linearly dependent.

Proof. There is a codeword $\mathbf{x}$ of weight $wt(\mathbf{x}) = d$ in $\mathcal{C}$ if and only if

$$H\mathbf{x}^T = \mathbf{0}$$

for $\mathbf{x}$ of weight d, if and only if d columns of H are linearly dependent. $\square$

The *Singleton bound* (due to Richard Collom Singleton) gives the best possible minimum distance once n and k are given.

Theorem 2. (The Singleton bound.) Let $\mathcal{C}$ be an (n, k, d) linear code. Then

$$n - k \geq d - 1,$$

that is

$$d \leq n - k + 1.$$

Proof. The rank of H is $n-k$, and by definition, this the maximum number of linearly independent columns of H. $\square$

Corollary 1. An (n, k) code reaching the Singleton bound can recover from up to $n - k$ erasures.

Proof. An erasure code $\mathcal{C}$ of minimum distance $d_H(\mathcal{C})$ can recover from up to $d_H(\mathcal{C}) - 1$ erasures, and a code reaching the Singleton bound satisfies $d_H(\mathcal{C}) = n - k + 1$. $\square$

Definition 5. A code achieving the Singleton bound is called *maximum distance separable (MDS).*

Because MDS codes are providing the best erasure protection given k and n, they are often the preferred erasure codes for both communication and storage applications. Let us see some examples of MDS codes.

Examples 9. Among the examples seen so far, only two codes have the MDS property.

(1) The repetition code $(n, 1)$ is a MDS code. Indeed $n-k+1 = n$ which is the minimum distance of the code (recall that this code has only 2 codewords, one with n ones and one with n zeroes).
(2) The single parity check code $(k+1, k)$ is also a MDS code. In this case $n-k+1 = (k+1)-k+1 = 2$ which is the minimum distance of the code. This can also be checked explicitly by looking at the 4 possible codewords.
(3) The $(7, 4)$ Hamming code is not MDS, since $d_H(\mathcal{C}) = 3$, while $n-k+1 = 7-4+1 = 4$. Note that $d_H(\mathcal{C})$ can be computed from Theorem 1, since the parity check matrix of the (7,4) Hamming code contains as columns all the binary vectors of length 3. Indeed, every $d-1 = 2$ columns are linearly independent, and there exists $d = 3$ columns which are linearly dependent.

Are there any other MDS codes than the two above? In fact, as long as we keep only 0 and 1 as coefficients of the code parity check matrix, the answer is no. Or more precisely:

Proposition 1. A binary $(n, k, n-k+1)$ linear code cannot exist for $k \neq 1$ and $k \neq n-1$.

Proof. By Theorem 1, we know that an $(n, k, n-k+1)$ code has a parity check matrix with $n-k$ rows, n columns and the property that every $n-k$ columns are linearly independent. Let us try to build such a matrix. We can get $n-k$ linearly independent columns of course, for example we can use the identity matrix $\mathbf{I}_{n-k}$. Now we add one more column, and this column must have the property that out of all the $n-k+1$ columns, every choice of $n-k$ gives linearly independent columns. If we choose the identity matrix, it is easy to see that we can add one column containing only ones: indeed, for any choice of $n-k-1$ columns among the $n-k$ first columns, one gets exactly one row with only zeroes, and thus the whole one vector will always be linearly independent. Now let us see that once we have those $n-k+1$ vectors, we cannot add any more column to

$$\begin{bmatrix}\mathbf{I}_{n-k} & \mathbf{1}\end{bmatrix}.$$

This follows from the fact that we actually had no choice when picking the whole 1 vector. Any other vector, that is having at least 0 zero, can be obtained from $n-1$ unit vectors. This shows that $n = n-k+1$, that is $k=1$. A similar argument works if the identity matrix is replaced by any $(n-k)\times(n-k)$ full rank matrix. The case $k=n-1$ corresponds to a $1 \times n$ matrix H. □

From the proof of the above proposition, we understand that the reason behind the lack of binary MDS codes is that we do not have enough possible choices of "linear combinations" when we only work with binary coefficients. Thus, to have (hopefully) more choices of MDS codes, we need to work with another set of coefficients than just 0 and 1.

3.3 Finite Fields and Reed-Solomon Codes

Consider the set $\mathbb{F}_2[X]$ of polynomials in X with coefficients that are either 0 or 1. Now given a polynomial $p(X) \in \mathbb{F}_2[X]$, we can look at the set denoted by $\mathbb{F}_2[X]/(p(X))$ of polynomials in $\mathbb{F}_2[X]$ modulo $p(X)$. The notion "modulo a polynomial" is the same as "modulo an integer". For it to work, we need the Euclidean division, that is for $f(X) \in \mathbb{F}_2[X]$, we write

$$f(X) = p(X)q(X) + r(X)$$

where the degree of $r(X)$ is smaller than that of $p(X)$, and we say that $f(X)$ is congruent to $r(X)$ modulo $p(X)$. $\mathbb{F}_2[X]/(p(X))$ contains all the possible remainders $r(X)$ where the arithmetic is dictated by the choice of $p(X)$. The arithmetic in $\mathbb{F}_2[X]/(p(X))$ is close to "usual", meaning that we can add and multiply two polynomials modulo $p(X)$. However it is not clear, and in fact not true in general, that any non-zero polynomial is invertible.

Examples 10. These two examples illustrate scenarios when $\mathbb{F}_2[X]/(p(X))$ is not and is a field respectively.

(1) Take $p(X) = X^2 + 1$, and look at $\mathbb{F}_2[X]/(p(X))$. We first notice that polynomials in $\mathbb{F}_2[X]/(p(X))$ have a degree which is at most 1, because $p(X)$ is of degree 2. The possible choices are thus $\{0, 1, X, X+1\}$. We also know that $X^2+1 = 0$, that is $X^2+1 = (X+1)(X-1) = (X+1)^2 = 0$, showing some anomaly with respect to usual arithmetic, since a non-zero element squared becomes 0. In fact, this is enough to show that $\mathbb{F}_2[X]/(p(X))$ cannot be a field: if $(X+1)$ were invertible, then $(X+1)^2 = 0$ could be multiplied on both sides by $(X+1)^{-1}$, showing that $X+1 = 0$, which leads to a contradiction.

(2) Instead, take $p(X) = X^2 + X + 1$, and look again at $\mathbb{F}_2[X]/(p(X))$. As a set, we still have $\{0, 1, X, X+1\}$ except that this time $X^2 + X + 1 = 0$, that is $X^2 = X + 1$. Now every non-zero element is invertible, because $X(X+1) = X^2 + X = 1$.

The key ingredient is that if $p(X)$ is irreducible, that is, it cannot be factored into a product of polynomials of lower degree, then $\mathbb{F}_2[X]/(p(X))$ will indeed have a field structure, thus it is called a *finite field*: finite, because its cardinality is finite, it is in fact 2^s where s is the degree of $p(X)$. There is something we can understand from the above examples. If the polynomial $p(X)$ is reducible, then it can be written as $p_1(X)p_2(X)$ for some non-zero polynomials, and since $p(X) = 0$ modulo $p(X)$, then $p_1(X)p_2(X) = 0$ modulo $p(X)$. If $p_1(X)$ were invertible, then we would get by multiplying both sides of the equation by $p_1(X)^{-1}$ that $p_2(X) = 0$, a contradiction.

Roughly speaking, when we say that $p(X) = 0$ in $\mathbb{F}_2[X]/(p(X))$, this means that $p(X)$ has a zero in $\mathbb{F}_2[X]/(p(X))$, say w, and it is more convenient to describe $\mathbb{F}_2[X]/(p(X))$ in terms of this element w than keeping the polynomial notation. A more formal way to deal with finite fields can be found in about any abstract algebra textbook!

Fig. 2: Irving S. Reed (1923-) and Gustave Solomon (1930-1996)

Example 11. Take again $p(X) = X^2 + X + 1$ and call w a root of $p(X)$, such that $p(w) = 0$, that is $w^2 + w + 1 = 0$. Then the above construction gives a finite field with 4 elements, denoted by $\mathbb{F}_4 = \{0, 1, w, w + 1\}$.

Let $\mathbb{F}_q$ denote some finite field (when $q = 2$, we get back to $\mathbb{F}_2 = \{0, 1\}$). It is now possible to read again the previous section, and define everything in $\mathbb{F}_q$: generator matrix, parity check matrix, minimum distance. The advantage of moving away from $\mathbb{F}_2$ is that we can now present a new class of MDS codes, incidentally one of the most famous ones, the class of *Reed-Solomon codes* [33].

Let $\mathbf{u} = [u_1, u_2, \ldots, u_k]$ be the information symbols and let $\alpha_1, \ldots, \alpha_n$ be n distinct values in some finite field $\mathbb{F}_q$. Consider the polynomial

$$f(X) = u_1 + u_2X + \ldots + u_kX^{k-1}.$$

Codewords are obtained by evaluating f at each α_i, that is

$$[f(\alpha_1), f(\alpha_2), \ldots, f(\alpha_n)].$$

We have that $k < n$ and n can be at most the size of $\mathbb{F}_q$.

Example 12. As a toy example, we can take $\mathbb{F}_4$ as the chosen finite field, given by $\{0, 1, w, w + 1\}$ with $w^2 = w + 1$. Since the size of the field is 4, we can pick say $n = 4$ and $k = 2$. Then

$$f(X) = u_1 + u_2X \in \mathbb{F}_2[X]$$

and a codeword looks like

$$[f(0), f(1), f(w), f(w+1)] = [u_1, u_1 + u_2, u_1 + u_2w, u_1 + u_2 + u_2w].$$

This can be expressed in terms of generator matrix as well:

$$[u_1, u_2] = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & w & w+1 \end{bmatrix}.$$

We conclude by noticing that Reed-Solomon codes are maximum distance separable (MDS) codes. The argument is based on Lagrange interpolation: a codeword from a Reed-Solomon code is formed by n points of a polynomial of degree $k-1$. Such a polynomial is fully determined by k points or more. This means that if at most $n-k$ points are lost, there are still k left to reconstruct the polynomial, no matter which set of k points is left.

Now armed with the notions of an (n, k) MDS erasure code, and having in mind examples of MDS codes such as the repetition code, the single parity check code, and Reed-Solomon codes, we are ready to go back to NDSS, and study how MDS codes are used in that context.

4

Erasure codes for NDSS

In the following, we study how MDS erasure codes can be used to store a single object in a networked distributed storage system (NDSS). Consider a system comprising of N nodes, of which $n < N$ nodes are used to store a single object in a distributed manner. The distribution is done using an (n, k) MDS erasure code (see Definition 5), and replication is included since it can be seen as a repetition code. We ignore non-MDS erasure codes at this point, because from a pure storage-efficiency and fault-tolerance perspective, non-MDS codes are sub-optimal (recall the Singleton bound from Theorem 2). We will however see in the second part of this survey that the use of non-MDS codes may be justified because of better repairability, and study some such codes in later chapters.

Thus, in this chapter, by an (n, k) code we always mean an MDS code. Now consider a single data object, cut into k pieces, and which are mapped to n encoded fragments using an (n, k) code. Every encoded piece for an object is stored in a distinct node. The data can be retrieved by accessing the encoded fragments from any k of these n nodes, and carrying out a reconstruction (decoding). In practice a single object may first be partitioned in smaller parts (depending on the size of the object and code parameters), and the whole object's availability will depend on the availability of all the parts. In the following we ignore such intricacies. One may interpret our study as regarding the availability of one specific 'part' in such a setting.

Definition 6. The *storage overhead* induced by the use of an (n, k) code is given by n/k. Note that this is the reciprocal of the rate of a code (see Definition 1). The storage overhead is also known in the literature as the *stretch* of a code.

We have previously argued that the rationale for using erasure codes is their supposedly better fault-tolerance than replication, for the same storage overhead. This can be quantified using the notion of *static resilience*. The static resilience of a system is defined as the amount of fault-tolerance it achieves given an initial level of redundancy, if no further remedial actions (such as, restoring redundancy) are taken.

4.1 Static Resilience

Consider a simplistic scenario where node failures are i.i.d. (that is independent and identically distributed). Let μ be the probability that an individual storage node is available, or alternatively, let $f = 1 - \mu$ be the failure probability of individual storage nodes. The actual number of independent node failures is then binomially distributed. Hence the probability that an object stored using an (n, k) MDS erasure code remains available A_{MDS} is equal to the probability that at least k nodes are available, i.e.,

$$A_{MDS} = \sum_{j=k}^{n} \binom{n}{j} \mu^j (1-\mu)^{n-j} = \sum_{j=k}^{n} \binom{n}{j} f^{n-j} (1-f)^j. \qquad (4.1)$$

In contrast, with r-way replication, the probability A_{rep} that the object remains available is

$$A_{rep} = 1 - f^r = 1 - (1-\mu)^r. \qquad (4.2)$$

Note that the static resilience for r-way replication can be viewed as a special case of the (n, k) MDS erasure code scheme, with $n = r$

and $k = 1$, since replication is essentially a trivial MDS erasure code, namely the repetition code.

Example 13. If the probability of failure of individual nodes is $f = 0.1$, then for the same storage overhead of 3, corresponding to $r = 3$ for replication and to a $(9, 3)$ erasure code, the probabilities of losing an object are

$$1 - A_{rep} = 10^{-3}$$

for replication, using (4.2) and

$$1 - A_{MDS} \sim 3 \cdot 10^{-6}$$

for the $(9, 3)$ erasure code using (4.1), respectively.

This example illustrates that an erasure code can be used to achieve significantly better resilience w.r.to replication, for an equivalent storage overhead.

An interesting question is, do erasure codes always provide better fault-tolerance than replication? This was investigated in [22], whose analysis and results are summarized next.

In order to compare both strategies subject to an equivalent storage overhead, let us write $n = rk$ where $r = n/k$ is the storage overhead (see Definition 6). Now recall from (4.1) that

$$A_{MDS} = P(X \geq k)$$

where X is the random variable representing the number of online storage nodes, and is binomially distributed ($X \sim B(n, \mu)$) with mean $rk\mu$ and variance $rk\mu(1 - \mu)$. By rewriting

$$A_{MDS} = P(X \geq k) = P(X \geq rk\mu + (k - rk\mu))$$

we can apply Chebyshev's inequality

$$P(X - E[X] \geq t) \leq \frac{Var[X]}{t^2 + Var[X]}$$

when $\mu < 1/r$ to determine the following upper bound:

$$A_{MDS} \leq \frac{rk\mu(1 - \mu)}{(k - rk\mu)^2 + rk\mu(1 - \mu)} = \frac{\mu(1 - \mu)}{rk(1/r - \mu)^2 + \mu(1 - \mu)}.$$

Likewise, when $\mu > 1/r$, one can similarly derive another bound [22]:

$$1 - A_{MDS} \leq \frac{\mu(1-\mu)}{rk(1/r - \mu)^2 + \mu(1-\mu)}.$$

This analysis indicates that there are two distinct regimes. (1) When the availability of storage nodes μ is smaller than the code's rate, i.e, $\mu < k/n$, then the larger k, the lower the object availability, and hence using replication, i.e., $k = 1$ is preferable. (2) On the contrary, when $\mu \geq k/n$, the larger the value of k, the better the system's resilience for any specific storage overhead.

While such a study reveals that erasure codes do not necessarily provide better resilience than replication under all circumstances, under diverse practical settings and considerations, erasure codes have been found to be beneficial. It is important to note that the static resilience analysis ignores the effect of maintenance mechanisms that replenish lost redundancy, which essentially yields a larger effective μ. Thus another metric, namely *mean time to data loss*, taking into account the dynamic processes of redundancy loss as well as replenishment, is also often studied in the literature. This metric will hardly be discussed in this survey, since the codes that we present have not (yet) been analyzed with respect to mean time to data loss.

4.2 Choice of Code Parameters

Another interesting conclusion one may draw from the above analysis is that the larger the value of the code parameter k, the better the system's resilience. However, there are many potential drawbacks of using a large k. Foremost, larger k implies much more meta-information for the different encoded pieces (that need to be kept up to date in a dynamic environment). It also implies that more nodes need to be contacted when reconstructing (decoding) the original data object. Downloading data from many different nodes in parallel during an access request may however also be profitable. These are some of the several possible system design implications which need to be considered when choosing the code parameters in practice.

Typical values of n and k thus depend on system considerations: for data centers, the number of temporary failures is relatively low, thus small (n, k) values such as $(9, 6)$ [18] or $(13, 10)$ [15] (with a respective storage overhead of $9/6 = 1.5$ and $13/10 = 1.3$) are known to have been used. In P2P systems, larger parameters like $(517, 100)$ are desirable (claimed to have been used in Wuala) to guarantee availability since nodes frequently go temporarily offline. Large values of n also allow a better margin for deferred repairs in P2P systems, thus either avoiding repairs altogether in presence of temporary churn, as well as amortizing the repair cost when repairs are actually carried out [3].

4.3 Maintenance

Overall, in realistic scenarios (where the probability of individual storage nodes being available μ is not too small) we saw that using erasure codes is advantageous in terms of fault-tolerance and storage overhead with respect to replication.

There is however a catch. Though erasure codes provide a good fault-tolerance as far as data reconstruction is concerned, they were not designed to *repair* subsets of arbitrary encoded blocks efficiently. When a data block encoded by an MDS erasure code is lost and has to be recreated, one would typically first need data equivalent in amount to recreate the whole object in one place (either by storing in advance a full copy of the data, or else by downloading an adequate number of encoded blocks), even in order to recreate a single encoded block. This in turn means that the number of nodes that are contacted for repair (also called *fan-in*) is at least k. This strategy is a waste of communication bandwidth if only one encoded fragment is needed, though its cost is amortized if repairs are deferred, and multiple repairs are carried out together. Even in the case of deferred repair, the node which needs to acquire adequate data and restore each encoded fragment remains a bottleneck, which might delay the repairs. A slower repair process can in turn adversely affect the long term data durability.

Finally, note that the above is a theoretical view point on maintenance, and the fact that many commercial deployments actually claim to use some variation of Reed-Solomon codes show that there are existing engineering solutions to mitigate some of the indicated drawbacks.

Part II

Codes with Better Repairability

5 Code Design for Repairability

The use of MDS erasure codes for fault tolerance in networked distributed storage systems (NDSS), where node availability is not too low, has been motivated in the first part of this survey. This is based on the premise that these erasure codes require a smaller storage overhead than replication for the same or even better fault tolerance.

However, an important drawback of MDS erasure codes is their lack of compatibility with NDSS maintenance requirements. This deficiency has triggered in the recent years a lot of research activity around the design of new classes of erasure codes, better suited for NDSS maintenance needs. Before discussing some of these novel coding techniques, we will like to outline the various aspects that are considered to characterize "better" repairability in the existing literature surveyed.

It is well understood that the original redundancy of a storage system needs to be replenished for data survivability on the long run. A naive repair strategy for an (n, k) MDS code is: one live node must obtain a whole copy of the object (or some data which contains all the original object information), from which the lost encoded pieces can be recreated before being distributed to other live nodes in the system.

Based on the understanding of the repair process for MDS erasure codes, several quantitative and qualitative conclusions can be drawn: (1) the **communication cost** of repair is heavy: in particular if no node keeps a whole replica, then k pieces of data needs to be downloaded for repairing even a single lost piece, (2) this in turn means that k **live nodes or more are actually require to respond**, while these nodes are likely to give priority to other tasks they need to complete at the same time, (3) which may further accentuates the issue of the **time needed to actually complete a repair**, which can be particularly critical in the event of correlated failures. One may also argue about (4) high **computational cost** for the node that needs to retrieve and encode the object again, and the fact that (5) this single node becomes a bottleneck for the whole repair process.

These drawbacks of the traditional repair mechanism used for MDS erasure codes shed light on some desirable repairability properties for codes used in the context of NDSS. In addition to continue to provide good fault tolerance (static resilience) for low storage overhead, such codes should, during the repair process exhibit one or preferably several of the following features:

(1) reduction in overall **data transfer** over the network (denoted as γ henceforth),
(2) reduction in the **number of nodes contacted**, i.e. *repair fan-in* (denoted as d [1]),
(3) reduction in the amount of data that needs to be read from the live nodes, i.e. **disk I/O**,
(4) possibility to **repair multiple faults**,
(5) possibility to **distribute** the repair load and **parallelize** the repair process and
(6) reduction in the **time** to complete repairs.

There are likely other desirable properties directly relevant to

[1] Note that we use a similar notation for the distance d_H of a code. Hopefully, the context of usage helps avoid any confusion.

the repair process itself (for instance, computational and system design complexity), and there are other consequences such as read, write and update efficiency, etc. that also are affected by the choice of the code design. Many of these additional issues will be mainly ignored in our survey, since for most of the codes that we will study, these aspects remain open for further investigation.

The above list will serve us as a guideline to categorize the code constructions presented next. In Chapter 6, we recall an approach based on network coding, whose goal is to minimize the data transfer during repair. In Chapter 7, four code constructions - Hierarchical codes, Pyramid codes, local reconstruction codes, and redundantly grouped codes - which have in common to combine two layers of erasure codes are described. Though each of them has an agenda of its own, they also share one salient aim, that of reducing the repair fan-in d, i.e. the number of nodes contacted to perform a repair. The codes presented in Chapter 8 push this idea to its limit, and ask for codes having a repair fan-in as small as possible. We note that a small fan-in might in turn result in reducing the repair bandwidth. In fact, how the desirable repair properties are intertwined is in itself an interesting question to address. We will see as we described the code constructions that there are many trade-offs, and as one might expect, it is not possible to achieve everything at the same time.

5.1 Notations and Assumptions

Let us fix some notation and terminology before we proceed. We consider a network of N nodes, and one data object is stored across n of these N nodes. This (data) object, or file, is of length B symbols (symbols being a unit of choice, such as bits, bytes, etc). We use the simplifying assumption that only one object is stored, unless stated otherwise explicitly. Each node is assumed to have the same storage capacity of α symbols.

Nodes that participate in the repair process are sometimes called *newcomers* in the literature. These could be nodes that are indeed newly added to the system, or may be nodes that already existed previously, but were not storing any data corresponding to the specific object whose redundancy replenishment is being carried out. A newcomer is assumed to connect to d nodes which store encoded parts of the object in question (sometimes also called *live nodes*), and download some data from each of these d nodes. Every link has a download capacity of β symbols, and a newcomer is thus downloading at most $d\beta$ amount of data in total from these live nodes.

When coding is used for redundancy, the file of length B is cut into k fragments of length B/k each, that we will denote by $\mathbf{u} = (u_1, \ldots, u_k)$ (to keep the same notation as in the previous chapter). Note that this splitting of the object into k to perform the encoding is different from potentially slicing the object first in case its size is too big.

Encoding generates a codeword $\mathbf{x} = (x_1, \ldots, x_n)$ with $n > k$. Each of the x_i is assigned to a different node to be stored.

Most commonly used notations:

(1) N is the total number of nodes in the network.
(2) n is the number of nodes storing one object.
(3) The size of a data object is B (symbols).
(4) An (n, k) code is used to encode the object.
(5) α is the storage capacity of every node.
(6) d is the repair fan-in,
(7) β is the amount of data downloaded from a live node by a newcomer participating in the repair.
(8) γ is the bandwidth cost to perform a repair.

6

Network Codes on Codes

The repair of failed nodes in networked distributed storage systems (NDSS) is a process which involves transmission of data from live nodes to nodes where the lost redundancy is being replenished (which will also be referred to as *newcomers*, as explained in the previous chapter). From that perspective, a natural question is whether network coding techniques [1, 38, 21], which are known to increase the throughput of a network by allowing intermediate nodes to encode their incoming packets, can be leveraged upon in order to improve the transfer of useful information during repair. In [9] (and references therein), the possible benefit of employing network coding techniques for replenishing redundancy is affirmed by determining how it can minimize the repair traffic for the case of a single failure. The main result of [9] is a nice trade-off between the node storage capacity α and the repair bandwidth γ, while ensuring that the original amount of fault-tolerance is restored. The first part of this chapter presents a more general storage capacity versus repair bandwidth trade-off, mainly based on the works [20] and [35], which generalize the result of [9] to the case of multiple faults repaired simultaneously.

Numerous codes have been proposed in the literature, which take achieving these storage-bandwidth curves as code design criterion. We will discuss some of these in the second half of the chapter, under the name of "network codes on codes", even though some of them might not really perform any network coding operations, in that the coding operations involved are trivial. Many such code constructions available in the literature are deliberately not reported here, both because the emphasis of this survey is not to exhaustively enumerate all codes, but instead to capture various flavors of code designs, network coding based techniques being only one of them, and also because surveys specialized on this particular type of coding for NDSS exist, and the interested reader may refer to one of them, e.g. [10].

6.1 Network Coding and Information Flow

A key concept in network coding theory is that of *information flow*: the network is seen as a directed graph, where data flows from the sources to the sinks via vertices and edges of this graph.

Similarly in the storage context, one data object to be stored flows from a source to some n storage nodes, then to some other storage nodes during the repair process, and finally to a sink (or data collector) which contacts κ[1] nodes to access the object. To capture the notion of storage capacity in the information flow, each node is abstracted as a logical pair of two nodes $(x_{\text{in}}, x_{\text{out}})$ [9], connected with an edge of capacity α: $x_{\text{in}} \overset{\alpha}{\to} x_{\text{out}}$. To repair t faults simultaneously, $t \geq 2$, t newcomers each contacts a possibly different set of d live nodes, and downloads β amount of data from each of the d nodes from its chosen set. The newcomers can then collaborate [20, 34] by exchanging the downloaded information among each other. To model this collaborative scenario, each node is now abstracted as a triple $(x_{\text{in}}, x_{\text{coord}}, x_{\text{out}})$ [20] instead of a pair, where the edge between x_{in} and x_{coord} represents the amount of data that the node temporarily stores (which is at least the storage

[1] The original papers use the notation k, however we decided to use κ since k is a code parameter, and we will see later that sometimes κ is equal to k while sometimes it is not.

capacity α), while the edge between x_{coord} and x_{out} is as before, i.e., the storage capacity α:

$$x_{\text{in}} \overset{\geq\alpha}{\rightarrow} x_{\text{coord}} \overset{\alpha}{\rightarrow} x_{\text{out}}.$$

Collaboration among the t nodes participating in the repair is modeled by edges of weight β' among the respective x_{in} and x_{coord} (see Fig. 3). In particular, each of the t nodes has $t-1$ incoming edges. The total amount of bandwidth needed per repair is thus

$$\gamma = d\beta + (t-1)\beta' \tag{6.1}$$

If $t = 1$, there is no collaboration, and $\gamma = d\beta$.

6.2 A Min-Cut Bound

A necessary condition for a data collector to recover a stored object despite several rounds of failures/repairs can be seen on the information flow graph considered above: it is needed that an amount of information at least the size of the stored object circulates from the source to the data collector. According to the max-flow min cut-theorem, any *min-cut* in the information graph should be at least equal to the object size. Let us now see how a min-cut bound can be derived [20].

We assume that repairs have been triggered g times, each at a threshold of t faults. The data collector contacts κ live nodes to download the data object it wants. Of these live nodes, a group of $u_i \leq t$ nodes had participated in the ith repair generation, $i = 0, \ldots, g-1$, thus $u_0 + \ldots + u_{g-1} = \kappa$.

The network nodes are partitioned into a set U and its complementary set $\bar{U}$. A cut is given by a set of edges, with one end connected to a node in U and the other end to a node in $\bar{U}$. We will pick one particular cut, as we now explain. The cut we choose is such that the source belongs to U, while $\bar{U}$ contains the data collector. Any new node $(x_{\text{in}}, x_{\text{coord}}, x_{\text{out}})$ joining one repair generation connects to d live nodes. We assume that the x_{out} part of any such node belongs to $\bar{U}$ and when the first repair is triggered, these are the only nodes in $\bar{U}$ along with the data collector.

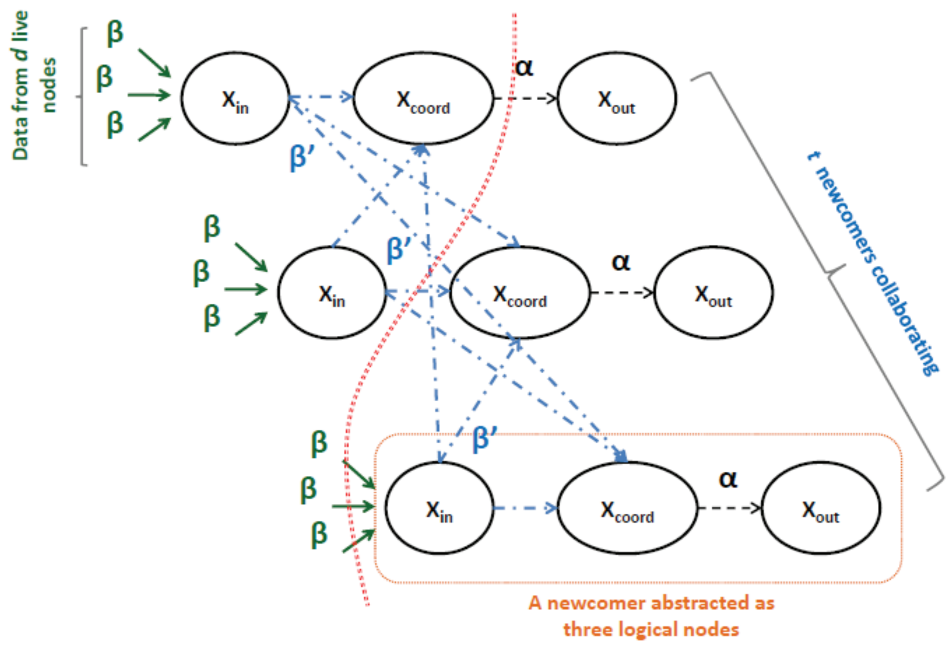
Data from *d* live nodes
β
β'
α
Xin
Xcoord
Xout
t newcomers collaborating
A newcomer abstracted as three logical nodes

Fig. 3: Three different cuts are illustrated: the storage capacity link, the $x_{\text{in}} \to x_{\text{coord}}$ link, or the download links are cut.

When the first round of repairs is triggered ($g = 0$), u_0 nodes join as a group. All the u_0 x_{out} nodes are in $\bar{U}$, and the x_{in} nodes can be either in U, say m of them, or in $\bar{U}$, for the remaining $u_0 - m$.

Case 1. For each of the m x_{in} nodes in U, either the corresponding x_{coord} node is in U or it is in $\bar{U}$. If it is in U, then the contribution to the cut is α, coming from the storage link $x_{\text{coord}} \overset{\alpha}{\to} x_{\text{out}}$. If the x_{coord} node is instead in $\bar{U}$, the cut involves $x_{\text{in}} \overset{\geq\alpha}{\to} x_{\text{out}}$. Since we do not know how many x_{coord} nodes are on either side, a lower bound is determined when all the m x_{in} nodes are in U, for a total cut of $m\alpha$.

Case 2. We now look at the $u_0 - m$ x_{in} nodes in $\bar{U}$. Since this is the first group of newcomers joining, those x_{in} nodes in $\bar{U}$ need to connect to existing nodes in U, leading to a cut of $d\beta$. Similarly, every x_{coord} node needs to connect to $t-1$ x_{in} nodes involved in this repair, but since $u_0 - m$ are already in $\bar{U}$ (in fact $u_0 - m - 1$ others), the cut will contain only $(t-1) - (u_0 + m - 1)$ of the t edges.

The total contribution $c_0(m)$ of this stage of repair in the min-cut is

$$c_0(m) \geq m\alpha + (u_0 - m)[d\beta + (t - u_0 + m)\beta'].$$

Since the function $c_0(m)$ is concave on the interval $[0, u_0]$, it has its minima (or minimum) on its boundary, namely in $m = 0$ and $m = u_0$, and $c_0(m)$ has either one global minimum in one of the two points, or two minima if both points give the same value, so that

$$c_0(m) \geq \min(c_0(u_0), c_0(0)) = \min(u_0\alpha, u_0[d\beta + (t - u_0)\beta']).$$

The process is the same for the group of newcomers joining the second repair generation. All the x_{out} nodes are in $\bar{U}$. Irrespectively of where their corresponding x_{coord} are, the m x_{in} nodes in U contribute at least $m\alpha$ to the cut. The $u_1 - m$ x_{in} nodes in $\bar{U}$ contribute $(u_1 - m)(d - u_0)\beta$, since they could connect to the u_0 nodes from the first group already in $\bar{U}$. Their corresponding x_{coord} nodes only connect to nodes from the second group, and thus as before, their contribution is $(t - u_1 + m)\beta'$, which gives

$$\begin{aligned} c_1(m) &\geq m\alpha + (u_1 - m)[(d - u_0)\beta + (t - u_1 + m)\beta'] \\ &\geq \min(u_1\alpha, u_1[(d - u_0)\beta + (t - u_1)\beta']). \end{aligned}$$

Iterating for all the groups of repairs gives the following min-cut bound.

Proposition 2. A min-cut bound between the source and a data collector is

$$\text{mincut}(\text{S}, \text{DC}) \geq \sum_{i=0}^{g-1} u_i \min(\alpha, [d - \sum_{j=0}^{i-1} u_j]\beta + (t - u_i)\beta')$$

where g is the number of repair generations, u_i is the size of each newcomer group, and $\kappa = \sum_{i=0}^{g-1} u_i$ with $1 \leq u_i \leq t$.

For the data collector to be able to retrieve the object, it is needed that the amount of information that flows from the source through the network is at least the size B of the data, that is

$$\sum_{i=0}^{g-1} u_i \min(\alpha, [d - \sum_{j=0}^{i-1} u_j]\beta + (t - u_i)\beta') \geq B.$$

Definition 7. We call any coding strategy that allows the data collector to retrieve the object by contacting any κ nodes and that satisfies

$$\sum_{i=0}^{g-1} u_i \min(\alpha, [d - \sum_{j=0}^{i-1} u_j]\beta + (t - u_i)\beta') = B \tag{6.2}$$

a *Collaborative regenerating code* ([20] used the term *coordinated regenerating codes* while [34] chose *cooperative regenerating codes*).

Note that the authors of [17] who proposed *mutually cooperative recovery* also computed a min-cut bound in the context of collaboration among newcomers, but they consider only a particular case of the above, when (1) $\beta = \beta'$, (2) the new nodes automatically contact all the live nodes.

When $t = 1$ failure, $u_i = 1$ for all i, thus $t - u_i = 0$ and $g = \kappa$, which gives

$$\text{mincut}(\text{S}, \text{DC}) \geq \sum_{i=0}^{\kappa-1} \min(\alpha, [d - \sum_{j=0}^{i-1} 1]\beta)$$

and we get the following corollary.

Corollary 2. [9] A min-cut bound between the source and a data collector is

$$\text{mincut}(\text{S}, \text{DC}) \geq \sum_{i=0}^{\kappa-1} \min\{(d - i)\beta, \alpha\}.$$

The goal is now, given n, κ, d, t, to find the optimal trade-off between the storage cost α and the repair cost γ (6.1) subject to the min-cut constraint (6.2). This is a highly non-trivial optimization problem, which is solved in closed form expression in [35]. We report the solution below, before discussing important particular cases, that of the boundary points [20, 35].

Let $C(d, \kappa, t)$ be the closure of all admissible operating points achieved by Collaborative regenerating codes. To shorten the notation, define

$$\begin{aligned} D_j &= \kappa(2d - 2\kappa + 2j + t - 1) - j(j-1) \\ D'_l &= \kappa(d + t(l+1) - \kappa) - t^2 l(l+1)/2 \\ \Delta_j &= \lfloor j/t \rfloor t^2 + (j - \lfloor j/t \rfloor t)^2 \\ \mu_j &= \frac{j(d-\kappa) + (j^2 + \Delta_j)/2}{jt - \Delta_j}, \ \Delta_j < jt, \end{aligned}$$

and $\mu_j = \infty$ if $\Delta_j = jt$.

Theorem 3. We have that $C(d, \kappa, t)$ is the convex hull of (that is, the smallest convex set containing) the union of the following sets of points:

$$\left\{ \left(\frac{2d+t-1}{D_j}, \frac{2(d-\kappa+j)+t-1}{D_j} \right), \ j = 2, \ldots, \kappa - 1, d \leq (r-1)\mu(j) \right\},$$

$$\left\{ \left(\frac{d+t-1}{D'_{\lfloor j/t \rfloor}}, \frac{d + t(\lfloor j/t \rfloor + 1) - \kappa}{D'_{\lfloor j/t \rfloor}} \right), \ j = 2, \ldots, \kappa - 1, d > (r-1)\mu(j) \right\},$$

where for both sets, it is defined that when $r = 1$, $0 \cdot \infty = \infty$,

$$\left\{ \left(\frac{d+t-1}{\kappa(d+t-\kappa)} + c, \frac{1}{\kappa} \right), \ c \geq 0 \right\},$$

and

$$\left\{ \left(\frac{2d+t-1}{\kappa(2d+t-\kappa)}, \frac{2d+t-1}{\kappa(2d+t-\kappa)} + c \right), \ c \geq 0 \right\}.$$

Note that the trade-off points are given normalized by the size B of the object.

When $t = 1$ (and remembering the convention $0 \cdot \infty = \infty$), the above theorem gives as particular case the optimal storage bandwidth trade-off curve computed in [9].

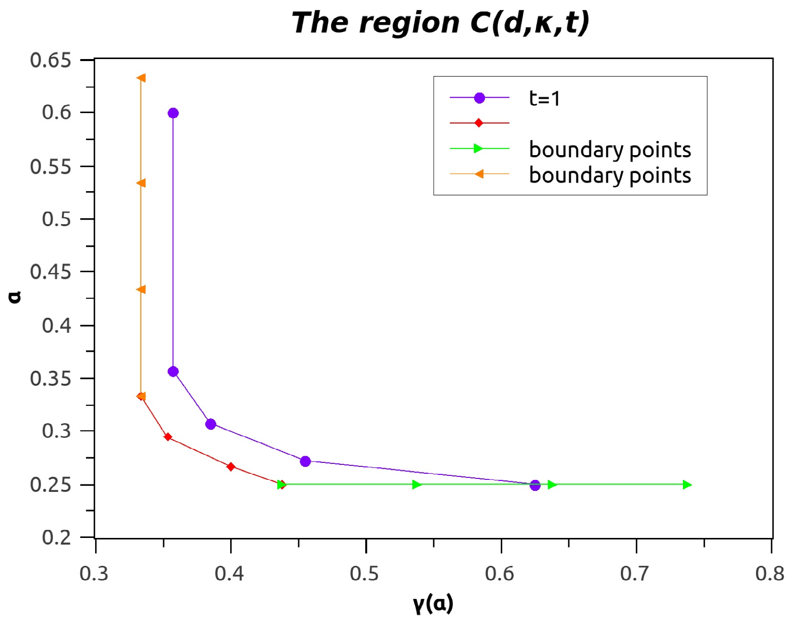

Fig. 4: The closure of all admissible operating points achieved by Collaborative regenerating codes: $C(d, \kappa, t)$

6.3 Minimum Storage and Repair Bandwidth Points

Two extreme cases of (6.2) can be identified. The highest contribution from the term $[d-\sum u_j]\beta$ comes when there is no contribution from $\beta'(t-u_i)$, that is, $u_i = t$ for all i, in which case $g = \kappa/t$, yielding

$$\sum_{i=0}^{\kappa/t-1} t \min((d-it)\beta, \alpha) = B. \tag{6.3}$$

Conversely, the highest contribution in $\beta'(t-u_i)$ is required when $[d-\sum u_j]\beta$ is minimized, that is $u_i = 1$ for all i, and $g = \kappa$:

$$\sum_{i=0}^{\kappa-1} \min((d-i)\beta + (t-1)\beta', \alpha) = B. \tag{6.4}$$

The minimum storage (MSR) point : the minimum storage per node is $\alpha = B/\kappa$, given that, by definition of κ, the data should be retrieved from any choice of κ nodes. Recall that when encoding an object with an (n, k) code, each encoded fragment, stored at a distinct node, has size B/k. Thus at the MSR point, $\kappa = k$, and the requirement that a data collector should be able to recover an object by contacting $\kappa = k$ nodes translates into saying that an MDS code (see Definition 5) is used.

Now from (6.3), we have a sum of κ/t terms, each of size at most tB/κ. Since the total must be at least B, each term in the sum must be tB/κ. Thus for every i

$$t \min((d - it)\beta, \alpha) = \frac{Bt}{\kappa}$$

and

$$(d - it)\beta \geq \frac{B}{\kappa},$$

in particular the smallest β is obtained for $i = \kappa/t - 1$, that is

$$\beta = \frac{B}{\kappa} \frac{1}{d - \kappa + t}.$$

The same computation with (6.4) gives

$$\beta' = \frac{B}{\kappa} \frac{1}{d - \kappa + t}.$$

Thus at the minimum storage point, we have (given that $\kappa = k$)

$$\boxed{\alpha = \frac{B}{\kappa}, \ \beta = \beta' = \frac{B}{\kappa} \frac{1}{d - \kappa + t}, \ \gamma = \frac{B}{\kappa} \frac{d + t - 1}{d + t - \kappa}}. \tag{6.5}$$

Note that this expression is also obtained by setting $c = 0$ in Theorem 3.

The denominators $d - \kappa + t$ have to be strictly positive for these expressions to make sense, namely $d - \kappa \geq 1 - t$, which in the particular case $t = 1$ implies that $d - \kappa \geq 0$ and the repair fan-in d must be at least the number κ of nodes contacted by the data collector for object retrieval. There is in fact a more fundamental tension between d and κ that appears here. Because the underlying (n,k) erasure code is MDS, the repair fan-in d must satisfy that $d \geq k = \kappa$. Indeed, if d were smaller than k, then it would mean that some of the encoded fragments stored at different nodes were linear combinations of each others, in which case, the data collector could not possibly reconstruct the data out of a set of k nodes which includes those $d+1$ nodes.

The minimum bandwidth point : to minimize γ alone, let α tend to infinity in (6.3), thus

$$\sum_{i=0}^{\kappa/t-1} t(d-it)\beta \geq B$$

that is

$$\begin{aligned}\beta &= \frac{B}{t}\frac{1}{d(\kappa/t) - t\sum_{i=0}^{\kappa/t-1} i} \\ &= \frac{B}{\kappa}\frac{2}{2d-\kappa+t}.\end{aligned}$$

From (6.4) we similarly compute that

$$\beta' = \frac{B}{\kappa}\frac{1}{2d-\kappa+t}.$$

Since α must be greater or equal to both $(d-i)\beta + (t-1)\beta'$ and $(d-it)\beta$ for every i, we get that

$$\alpha = d\beta + (t-1)\beta'.$$

To summarize, at the minimum bandwidth point, we have

$$\boxed{\alpha = \frac{B}{\kappa}\frac{2d+t-1}{2d-\kappa+t},\ \beta' = \frac{B}{\kappa}\frac{1}{2d-\kappa+t},\ \beta = 2\beta',\ \gamma = \alpha}. \quad (6.6)$$

This expression is also obtained by setting $c = 0$ in Theorem 3.

6.4 Examples of Regenerating Codes

In the previous section, Collaborative regenerating codes were defined as codes whose parameters fit in the trade-off curve between the storage overhead α and the repair bandwidth $\gamma = d\beta+(t-1)\beta'$, or satisfy (6.2). When $t = 1$, we drop the term "collaborative" and call them *Regenerating codes*, as they were firstly introduced [9].

The above analysis provides bounds on the best that can be achieved, but not the codes themselves. Most importantly, it does not specify what the repair process should look like. While it is easier for the system to keep track of the system state when a recreated fragment is "bit by bit" identical to what was lost, in fact it is also possible to replace the missed data by some other data, which plays the same role as far as maintaining the amount of redundancy is concerned. The former is called *exact repair* [36], while the latter form of repair has been referred to as *functional repair*[2].

We will present three illustrative constructions of Regenerating codes, two of them [31] are actually for the case of one failure $t = 1$, one at the minimum bandwidth point, and one at the minimum storage point. The third construction [34] is a Collaborative regenerating code at the minimum storage point. We chose these examples to illustrate different design points, and we are by no means trying to be exhaustive here. For a better overview of existing Regenerating codes, the interested readers may refer to existing surveys that discuss exclusively this type of codes, e.g. [10].

Construction I [31]: one failure at the minimum storage point. Consider the minimum storage regeneration (MSR) point for $t = 1$ failure, which from (6.5) is given by

$$(\alpha_{MSR}, \beta_{MSR}) = \left(\frac{B}{\kappa}, \frac{B}{\kappa(d-\kappa+1)}\right).$$

If we fix $\beta_{MSR} = 1$ which is the smallest unity of data that can be downloaded (we normalize with respect to β), we get

$$B = \kappa(d-\kappa+1)$$

[2] While the phrase 'functional repair' was coined in [36], we will like to note that introducing non-identical substitute for lost redundancy has been contemplated in the literature for other kinds of codes also, for instance for rateless codes.

and thus

$$\alpha_{MSR} = d - \kappa + 1, \beta_{MSR} = 1.$$

By fixing $d = \kappa + 1$, we further have

$$B = 2\kappa, \ \alpha_{MSR} = 2.$$

Encoding. Let $\mathbb{F}_q$ be some finite field. Partition the object $\mathbf{o} = (o_1, \ldots, o_B) \in \mathbb{F}_q^B$ into two vectors $\mathbf{o}_1 = (o_1, \ldots, o_\kappa)$ and $\mathbf{o}_2 = (o_{\kappa+1}, \ldots, o_B)$ (recall that $B = 2\kappa$ thus both vectors have the same size). The ith node will store $\alpha = 2$ symbols, given by

$$(\mathbf{o}_1 p_i^T, \mathbf{o}_2 p_i^T + \mathbf{o}_1 v_i^T)$$

where p_i and v_i are row vectors which define the encoding, $i = 1, \ldots, n$. It is asked that the vectors p_i^T are the columns of the generator matrix G' of an (n, k) MDS code (see Definition 5):

$$G' = \begin{bmatrix} p_1^T & \ldots & p_n^T \end{bmatrix}.$$

The encoding of the whole code, which is an $(2n, 2\kappa)$ linear code, is then given by

$$[\mathbf{o}_1, \mathbf{o}_2] \begin{bmatrix} G' & V \\ \mathbf{0} & G' \end{bmatrix} \tag{6.7}$$

where $V = [v_1^T, \ldots, v_n^T]$ is some matrix.

It might be confusing to see the use a $(2n, 2k)$ code, while one would expect to see an (n, k) code here. What is actually happening is that each node stores 2 symbols, thus we do have a code which maps k symbols of length 2, to n symbols of length 2 as well. It is however easier to design a code on $2k$ symbols, by splitting every symbol of length 2 into two.

Object retrieval. To retrieve the object from κ nodes, a data collector downloads the 2 symbols from κ nodes, and then obtains

$$(\mathbf{o}_1 p_i^T, \mathbf{o}_2 p_i^T + \mathbf{o}_1 v_i^T)$$

for a set of κ indices. Because G' is the generator matrix of an (n, κ) MDS code, $\mathbf{o}_1$ can be recovered. Once $\mathbf{o}_1$ is known, and assuming that all the v_i are known, the data collector is left with

$\mathbf{o}_2 p_i^T$, which can be recovered similarly as $\mathbf{o}_1$ since G' corresponds to a MDS code.

Repair of a failure. Suppose one node, say the jth node fails, thus $(\mathbf{o}_1 p_j^T, \mathbf{o}_2 p_j^T + \mathbf{o}_1 v_j^T)$ is lost. A newcomer downloads $\beta = 1$ symbol from $d = \kappa + 1$ nodes (w.l.o.g we label these d nodes from 1 to $d = \kappa + 1$), and this downloaded symbol is of the form

$$w_i = a_i(\mathbf{o}_1 p_i^T) + (\mathbf{o}_2 p_i^T + \mathbf{o}_1 v_i^T),\ i = 1, \dots, d,$$

for some $a_i \in \mathbb{F}_q$ to be defined later. Now the newcomer computes two new symbols out of these d w_i, by creating a linear combination of them:

$$(\sum_{i=1}^{d} \delta_i w_i, \sum_{i=1}^{d} \rho_i w_i),$$

and ρ_i and δ_i are chosen respectively such that

$$\sum_{i=1}^{d} \rho_i p_i^T = p_j^T,\ \sum_{i=1}^{d} \delta_i p_i^T = \mathbf{0}.$$

The key point here is that both equations are sets of κ linear equations (the p_i have length κ) in $d = \kappa + 1$ unknowns (either δ_i or ρ_i) and thus have a non-trivial solution in δ_i, i.e., such that the δ_i are not all zero. Now

$$\begin{aligned}
\sum_{i=1}^{d} \delta_i w_i &= \sum_{i=1}^{d} \delta_i [a_i(\mathbf{o}_1 p_i^T) + (\mathbf{o}_2 p_i^T + \mathbf{o}_1 v_i^T)] \\
&= \mathbf{o}_1 \sum_{i=1}^{d} \delta_i a_i p_i^T + \mathbf{o}_2 \sum_{i=1}^{d} \delta_i p_i^T + \sum_{i=1}^{d} \delta_i \mathbf{o}_1 v_i^T \\
&= \mathbf{o}_1 \sum_{i=1}^{d} \delta_i [a_i p_i^T + v_i^T]
\end{aligned}$$

and it is enough to pick a_i (again we have κ linear equations in $\kappa + 1$ unknowns) such that

$$\sum_{i=1}^{d} \delta_i [a_i p_i^T + v_i^T] = p_j^T$$

to repair the first symbol $\mathbf{o}_1 p_j^T$. Similarly

$$\begin{aligned}
\sum_{i=1}^{d} \rho_i w_i &= \sum_{i=1}^{d} \rho_i [a_i(\mathbf{o}_1 p_i^T) + (\mathbf{o}_2 p_i^T + \mathbf{o}_1 v_i^T)] \\
&= \mathbf{o}_1 \sum_{i=1}^{d} \rho_i a_i p_i^T + \mathbf{o}_2 \sum_{i=1}^{d} \rho_i p_i^T + \sum_{i=1}^{d} \rho_i \mathbf{o}_1 v_i^T \\
&= \mathbf{o}_1 \sum_{i=1}^{d} \rho_i [a_i p_i^T + v_i^T] + \mathbf{o}_2 p_j^T \\
&= \mathbf{o}_1 {v'_j}^T + \mathbf{o}_2 p_j^T
\end{aligned}$$

where

$${v'_j}^T = \sum_{i=1}^{d} \rho_i [a_i p_i^T + v_i^T]$$

is some vector (which is not important as long as it is known, as seen in (6.7)). Since $(\mathbf{o}_1 p_j^T, \mathbf{o}_2 p_j^T + \mathbf{o}_1 v_j^T)$ has been replaced with $(\mathbf{o}_1 p_j^T, \mathbf{o}_2 p_j^T + \mathbf{o}_1 {v'_j}^T)$ with v'_j being possibly different from v_j, this construction provides functional repair.

Example 14. Take $n = 5$ nodes, $\kappa = 3$ and $d = \kappa + 1 = 4$, so that $B = 2\kappa = 6$. We denote the object $\mathbf{o} = (o_1, \ldots, o_6) \in \mathbb{F}_q^6$ with $\mathbf{o}_1 = (o_1, o_2, o_3)$ and $\mathbf{o}_2 = (o_4, o_5, o_6)$. The encoding is thus done by

$$\begin{aligned}
&[\mathbf{o}_1, \mathbf{o}_2] \begin{bmatrix} p_1^T & \ldots & p_5^T & v_1^T & \ldots & v_5^T \\ \mathbf{0} & \ldots & \mathbf{0} & p_1^T & \ldots & p_5^T \end{bmatrix} \\
&= [\mathbf{o}_1 p_1^T, \ldots, \mathbf{o}_1 p_5^T, \mathbf{o}_1 v_1^T + \mathbf{o}_2 p_1^T, \ldots, \mathbf{o}_1 v_5^T + \mathbf{o}_2 p_5^T].
\end{aligned}$$

To get a suitable $(5, 3)$ MDS code, we can take a Reed-Solomon code (see Section 3.3). To be sure that a Reed-Solomon code exists, it is enough to pick a finite field with the right size, that is $|\mathbb{F}_q| \geq 5$. Let us say we pick $\mathbb{F}_8 = \{0, 1, w, w+1, w^2, w^2+1, w^2+w, w^2+w+1\}$ and $w^3 = w + 1$. The encoding of the Reed-Solomon code is done by evaluating the polynomial $f(X) = o_1 + o_2 X + o_3 X^2$ in say

$1, w, w+1, w^2, w^2+1$, yielding

$$[o_1, o_2, o_3]\begin{bmatrix}1 & 1 & 1 & 1 & 1\\ 1 & w & w+1 & w^2 & w^2+1\\ 1 & w^2 & (w+1)^2 & w^4 & (w^2+1)^2\end{bmatrix}.$$

Construction II [31]: one failure at the minimum bandwidth point. Consider the minimum repair bandwidth (MRB) point for $t = 1$ failure, which from (6.6) is given by

$$(\alpha_{MRB}, \beta_{MRB}) = \left(\frac{2Bd}{2\kappa d - \kappa^2 + \kappa}, \frac{2B}{2\kappa d - \kappa^2 + \kappa}\right).$$

If we fix again $\beta_{MRB} = 1$, we get

$$B = \kappa d - \frac{\kappa^2 - \kappa}{2}$$

and thus

$$\alpha_{MRB} = d, \beta_{MRB} = 1.$$

Now the the biggest gain in bandwidth occurs when $d = n-1$, in which case, the choice of $\beta = 1$, $d = n-1$ and κ determine the size B of the object:

$$B = \kappa(n-1) - \frac{\kappa^2 - \kappa}{2}.$$

Encoding. Let $\mathbf{o} = (o_1, \ldots, o_B) \in \mathbb{F}_q^B$ be the object to be stored. Every node stores $\alpha = d = n-1$ encoded pieces, computed as follows (see Fig. 5). Create a fully connected acyclic graph that connect all the n nodes, thus containing $n(n-1)/2$ vertices. Now to each of the edges, associate a vector v_j, $j = 1, \ldots, n(n-1)/2$, and give to each node $n-1$ encoded pieces of the form $\mathbf{o}v_j^T$, where the v_j's are the $n-1$ vectors associated to each edge connected to this node. The vectors v_j, $j = 1, \ldots, n(n-1)/2$, must satisfy the following conditions.

Object retrieval. For the data collector to be able to retrieve the data out of κ nodes, we must have that out of any choice of κ nodes, it gets $\kappa(n-1)$ encoded pieces which yield a system of $\kappa(n-1)$ equations to be solved for $B = \kappa(n-1) - \frac{\kappa^2-\kappa}{2}$ unknowns.

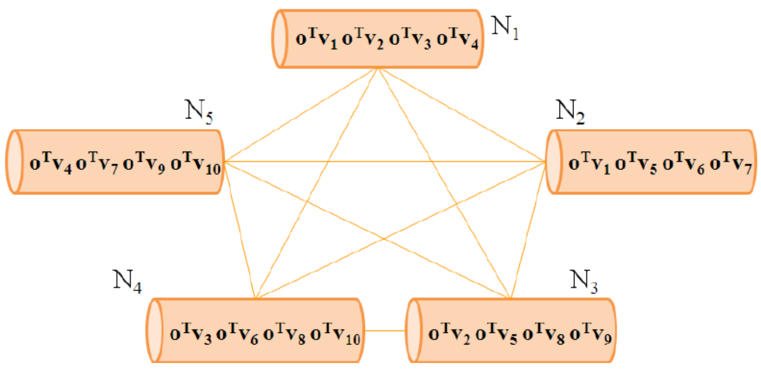
N1
o^T v1 o^T v2 o^T v3 o^T v4
N5
o^T v4 o^T v7 o^T v9 o^T v10
N2
o^T v1 o^T v5 o^T v6 o^T v7
N4
o^T v3 o^T v6 o^T v8 o^T v10
N3
o^T v2 o^T v5 o^T v8 o^T v9

Fig. 5: A Regenerating code at MBR.

Repair of a failure. If the ith node fails, a newcomer can get its data by downloading one fragment from each of the $n-1$ remaining nodes, since by the code construction, whenever a vertex connects two nodes, they share a common encoded fragment of the form $\mathbf{o}v_j^T$.

Example 15. Consider the example shown in Fig. 5 with $n = 5$ and $\kappa = 3$, so that $d = n - 1 = 4$. The number of nodes is thus $n = 5$, while the number of vertices is $n(n-1)/2 = 10$, for an object of size $B = 9$. If we write the encoding as that of a linear erasure code, we get

$$[o_1, \ldots, o_B] \begin{bmatrix} v_1^T & v_2^T & \ldots & v_{10}^T \end{bmatrix}$$

and we can choose the generator matrix $G = [v_1^T, \ldots, v_{10}^T]$ to be that of the single parity check code $(10, 9)$ (see Example 2), that is

$$G = \begin{bmatrix} \mathbf{I}_9 & \mathbf{1}^T \end{bmatrix}$$

where $\mathbf{1} = [1, 1, 1, 1, 1, 1, 1, 1, 1]$. By contacting $\kappa = 3$ nodes, the data collector gets $\kappa(n-1) = 3 \cdot 4 = 12$ encoded pieces which yield a system of 12 equations, out of them 3 are redundant. Thus, in fact 9 independent equations need to be solved for $B = 9$ unknowns.

We make one more remark here: the code parameters are $(B, n(n-1)/2)$, and in the above example the chosen (10,9) code is in fact MDS. However, as mentioned for the previous code, what is needed is that when we look at the code where the encoded fragments are grouped together, the resulting code is MDS. This is in theory less restrictive than asking the $(B, n(n-1)/2)$ code to be MDS, though it is not clear how such a code should be computed. On the other hand, it is indeed enough to pick the $(B, n(n-1)/2)$ code to be MDS.

Construction III [34]: t failures at the minimum storage point. Consider the (n, k) Reed-Solomon code which is defined over the finite field $\mathbb{F}_q$ with $q \geq n$. Suppose that the size B of the object $\mathbf{o}$ to be stored is $B = tk$, so that $\mathbf{o} = (o_{11}, \ldots, o_{1k}, \ldots, o_{t1}, \ldots, o_{tk})$ can be written as the matrix

$$\mathbf{O} = \begin{bmatrix} o_{11} & \ldots & o_{1k} \\ & & \\ o_{t1} & \ldots, & o_{tk} \end{bmatrix}, \; o_{ij} \in \mathbb{F}_q.$$

Note that the object is cut into a number of pieces which depends on the number t of failures after which a repair is triggered, with $k < n - t$. We only consider the regime $\kappa = k = d$.

Encoding. The generator matrix G of the Reed-Solomon code is a $k \times n$ Vandermonde matrix whose columns are denoted by $\mathbf{g}_i$, $i = 1, \ldots, n$. Every node is assumed to know G, and the ith node stores the t-dimensional column vector $\mathbf{O}\mathbf{g}_i$, whose rows represent t different pieces, forming its encoded data.

Object retrieval. Any choice of k nodes $i_1, \ldots, i_k$ clearly allows to retrieve $\mathbf{o}$ since we get

$$\mathbf{O}[\mathbf{g}_{i_1}, \ldots, \mathbf{g}_{i_k}]$$

where the matrix formed by any k columns of G is a Vandermonde matrix and is thus invertible.

Repair of t failures. Let us now assume that t nodes become unavailable, and t new nodes join. Let us call the t new nodes as nodes 1 to t w.l.o.g. The ith newcomer asks $[o_{i1}, \ldots, o_{ik}]\mathbf{g}_j$ for any choice $j_1, \ldots, j_k$ of k nodes among the live nodes, to obtain

$$[o_{i1}, \ldots, o_{ik}][\mathbf{g}_{j_1}, \ldots, \mathbf{g}_{j_k}],$$

from which $[o_{i1}, \ldots, o_{ik}]$ is decoded, by inverting the matrix formed by these k columns of G. This furthers allows the ith node to compute $[o_{i1}, \ldots, o_{ik}]\mathbf{g}_j$ for any j, which is then sent to the jth newcomer. All newcomers do similar computations and likewise deliver the missing pieces to the other newcomers, hence completing the collaborative regeneration process.

Example 16. Consider the $(7, 3)$ Reed-Solomon code which is defined over the finite field $\mathbb{F}_8 = \{0, 1, w, w^2, w^3, w^4, w^5, w^6, w^7\}$ with $w^3 = w + 1$. Suppose that the object $\mathbf{o}$ is to be stored in $n = 7$ nodes, while expecting to deal with $t = 2$ failures. First, represent the object as $\mathbf{o} = (o_{11}, o_{12}, o_{13}, o_{21}, o_{22}, o_{23})$ with o_{ij} in $\mathbb{F}_8$. The generator matrix G of the Reed-Solomon code is given by:

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ w & w^2 & 1+w & w+w^2 & 1+w+w^2 & 1+w^2 & 1 \\ w^2 & w+w^2 & 1+w^2 & w & 1+w & 1+w+w^2 & 1 \end{bmatrix}.$$

Now create a matrix $\mathbf{O}$ as follows:

$$\mathbf{O} = \begin{bmatrix} o_{11} & o_{12} & o_{13} \\ o_{21} & o_{22}, & o_{23} \end{bmatrix}.$$

The ith node stores $\mathbf{O}\mathbf{g}_i$ where $\mathbf{g}_i$ denotes the ith column of G, for example, node 1 stores the two units

$$\mathbf{O} \begin{bmatrix} 1 \\ w \\ w^2 \end{bmatrix} = \begin{bmatrix} o_{11} + o_{12}w + o_{13}w^2 \\ o_{21} + o_{22}w + o_{23}w^2 \end{bmatrix}.$$

Any choice of $k = 3$ nodes i_1, i_2, i_3 clearly allows to retrieve $\mathbf{o}$ since we get

$$\mathbf{O}[\mathbf{g}_{i_1}, \mathbf{g}_{i_2}, \mathbf{g}_{i_3}]$$

where the matrix formed by any 3 columns of G is a Vandermonde matrix and is thus invertible.

7

Codes on Codes

The codes proposed in the previous chapter can be seen as a combination of an erasure code and a network code.

Other families of codes have been proposed, which combine two erasure codes instead. Examples include Hierarchical Codes [11], Pyramid Codes [19] and Local Reconstruction Codes [18] which we will present next. While all of them have their own set of design objectives to fulfill, they all have in common better repairability. We will end this chapter by describing redundantly grouped codes [7], which use codes on codes as well, but differ from all the others by the fact that it encodes multiple objects together.

We start by recalling the definition of product codes, a classical technique introduced by Elias [12] to combine two erasure codes.

7.1 Product Codes

Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be two systematic codes with parameters (n_1, k_1) and (n_2, k_2) respectively, and generator matrices

$$G_1 = [\mathbf{I}_{k_1}\ A_1],\ G_2 = [\mathbf{I}_{k_2}\ A_2],$$

If U is a $k_2 \times k_1$ matrix containing the information symbols of

$k_2 \times k_1$ data bits	$k_2 \times (n_1 - k_1)$ parity bits for $\mathcal{C}_1$
$(n_2 - k_2) \times k_1$ parity bits for $\mathcal{C}_2$	$(n_2 - k_2) \times (n_1 - k_1)$ "parity of parity"

Table 7.1: The general form of a codeword from the product code of $\mathcal{C}_1$ and $\mathcal{C}_2$.

the data to be encoded, then the *product code* $\mathcal{C} = \mathcal{C}_1 \times \mathcal{C}_2$ of $\mathcal{C}_1$ and $\mathcal{C}_2$ is described by the $n_2 \times n_1$ codewords of the form

$$X = \begin{bmatrix} U & UA_1 \\ {A_2}^T U & {A_2}^T U A_1 \end{bmatrix},$$

where each row is a codeword from $\mathcal{C}_1$, since

$$X = \begin{bmatrix} U \\ {A_2}^T U \end{bmatrix} [\mathbf{I}_{k_1} \; A_1]$$

and similarly each column is a codeword from $\mathcal{C}_2$:

$$X = \begin{bmatrix} \mathbf{I}_{k_2} \\ {A_2}^T \end{bmatrix} [U \; UA_1].$$

The product code $\mathcal{C}$ of $\mathcal{C}_1$ and $\mathcal{C}_2$ has parameters $(n_1 n_2, k_1 k_2)$, and in fact, its minimum Hamming distance $d_H(\mathcal{C})$ can be computed from $d_H(\mathcal{C}_1)$ and $d_H(\mathcal{C}_2)$.

Theorem 4. The minimum Hamming distance of the product code $\mathcal{C} = \mathcal{C}_1 \times \mathcal{C}_2$ is

$$d_H(\mathcal{C}) = d_H(\mathcal{C}_1) d_H(\mathcal{C}_2).$$

Proof. Since every row of the codeword X belongs to $\mathcal{C}_1$, every row has at least $d_H(\mathcal{C}_1)$ nonzero elements. Similarly, every column belongs to $\mathcal{C}_2$, and thus has at least $d_H(\mathcal{C}_2)$ nonzero elements, which shows that

$$d_H(\mathcal{C}) \geq d_H(\mathcal{C}_1) d_H(\mathcal{C}_2).$$

Choose now a codeword $\mathbf{x}$ from $\mathcal{C}_1$ with weight $d_H(\mathcal{C}_1)$. Then a codeword X in which all the columns corresponding to zeros in $\mathbf{x}$ are all-zero columns, and all the other columns are a codeword of weight $d_H(\mathcal{C}_2)$ in $\mathcal{C}_2$ will always belong to the product code $\mathcal{C}$ of $\mathcal{C}_1$ and $\mathcal{C}_2$, and will have weight $d_H(\mathcal{C}_1)d_H(\mathcal{C}_2)$. This concludes the proof. □

It follows from the above theorem that a product code is not MDS.

Corollary 3. The product code $\mathcal{C} = \mathcal{C}_1 \times \mathcal{C}_2$ cannot be a MDS code.

Proof. An (n_1n_2, k_1k_2) code $\mathcal{C}$ is MDS if and only if $d_H(\mathcal{C}) = n_1n_2 - k_1k_2 + 1$. In the best case, when both $\mathcal{C}_1$ and $\mathcal{C}_2$ are themselves MDS, we get that

$$d_H(\mathcal{C}) = d_H(\mathcal{C}_1)d_H(\mathcal{C}_2) = (n_1 - k_1 + 1)(n_2 - k_2 + 1)$$

and it is enough to show that

$$(n_1 - k_1 + 1)(n_2 - k_2 + 1) < n_1n_2 - k_1k_2 + 1,$$

which is equivalent to

$$-n_1k_2 + n_1 - k_1n_2 + 2k_1k_2 - k_1 + n_2 - k_2 < 0.$$

But

$$\begin{aligned} & -n_1k_2 + n_1 - k_1n_2 + 2k_1k_2 - k_1 + n_2 - k_2 \\ = \; & k_1(k_2 - 1) + k_2(k_1 - 1) - n_1(k_2 - 1) - n_2(k_1 - 1) \\ = \; & (k_2 - 1)(k_1 - n_1) + (k_1 - 1)(k_2 - n_2) < 0 \end{aligned}$$

whenever $k_1, k_2 > 1$ and $k_1 < n_1, k_2 < n_2$. □

Example 17. The simplest example of product code is the *parity check code*. If both $\mathcal{C}_1$ and $\mathcal{C}_2$ are single parity check codes, with generator matrix

$$G_1 = [\mathbf{I}_{k_1}\ \mathbf{1}],\ G_2 = [\mathbf{I}_{k_2}\ \mathbf{1}]$$

where **1** is the whole 1 column vector of the suitable dimension, then

$$X = \begin{bmatrix} u_{11} & \dots & u_{1k_2} & \sum_{i=1}^{k_2} u_{1i} \\ & & & \\ u_{k_1 1} & \dots & u_{k_1 k_2} & \sum_{i=1}^{k_2} u_{k_1 i} \\ \sum_{i=1}^{k_1} u_{i1} & & \sum_{i=1}^{k_1} u_{ik_2} & \sum_{j=1}^{k_2}\sum_{i=1}^{k_1} u_{ij} \end{bmatrix} = (x_{ij}).$$

It is easy to see in this parity check code that if an encoded block x_{ij} is erased, then it is possible to recompute it using the remaining $n_1 - 1$ blocks from the ith row, or the $n_2 - 1$ blocks from the jth column, provided that the remaining blocks of the row or column respectively are still available. This example shows that it is possible to carry out a repair without having to access the whole original data. In fact, such a localization of a repair is feasible because a product code is not maximum distance separable (as shown in Corollary 3), and hence there are linear dependencies among a smaller subset of encoded pieces.

Product codes (and their generalization to higher dimensional product codes) have been used in storage media such as compact discs, thanks to their resilience against both random and bursty erasures [4, sec. 10.1]. Indeed despite their poor minimum distance, product codes have a structure that allows to correct large numbers of errors [4, sec. 10.1]. They have not been studied in their generality, as far as we know, in context of distributed storage systems. However, the codes that we will present next can be viewed as variations of the basic idea of employing two codes (codes on codes) in order to achieve a certain degree of localized repairs.

7.2 Hierarchical Codes

Hierarchical codes [11] can be seen as a *bottom-up* approach of applying codes on codes. Let us first elaborate this with an example.

Example 18. Take the $(3, 2)$ binary single parity check code, that maps

$$[u_1, u_2] \mapsto [u_1, u_2, u_1 + u_2].$$

Now take two copies $\mathcal{C}_1$ and $\mathcal{C}_2$ of this code, namely

$$\mathcal{C}_1 : [u_1, u_2] \mapsto [u_1, u_2, u_1 + u_2],\ \mathcal{C}_2 : [u_3, u_4] \mapsto [u_3, u_4, u_3 + u_4].$$

We now combine these two codes to obtain a $(7, 4)$ code $\mathcal{C}$ as follows:

$$[u_1, u_2, u_3, u_4] \mapsto [u_1, u_2, u_1 + u_2, u_3, u_4, u_3 + u_4, u_1 + u_2 + u_3 + u_4].$$

If we look at the obtained codeword in Example 18, we see that the 3rd encoded coefficient $u_1 + u_2$ corresponds to the parity bit obtained from the the first code, and the 6th coefficient $u_3 + u_4$ to that of the second code. These two bits act as *local redundancy* which can be used to localize the repair of an erasure. For instance, in case u_2 and u_3 both fail, they can be recovered independently using $u_1, u_1 + u_2$ and $u_4, u_3 + u_4$ respectively. The last coefficient $u_1+u_2+u_3+u_4$ is a parity bit computed from the all 4 information symbols, and provides some *global redundancy.*

We remark that the above example has some similarities with product codes: in fact, Example 17 with $k_1 = k_2 = 2$ gives

$$X = \begin{bmatrix} u_{11} & u_{12} & u_{11} + u_{12} \\ u_{21} & u_{22} & u_{21} + u_{22} \\ \sout{u_{11} + u_{21}} & \sout{u_{12} + u_{22}} & u_{11} + u_{12} + u_{21} + u_{22} \end{bmatrix}$$

and the above Hierarchical code is a product code combining two parity codes where two coefficients ($u_{11} + u_{21}$ and $u_{12} + u_{22}$) have been deleted.

This example can now be generalized in different ways (in which case, the resulting Hierarchical code may not fit the product code framework anymore).:

(1) Instead of calculating a simple parity bit for the local redundancy, more sophisticated codes could be used, and multiple bits of local redundancy may be created.
(2) Likewise, instead of using a single parity check code to determine the global redundancy, more sophisticated codes could again be used to create multiple bits of global redundancy.

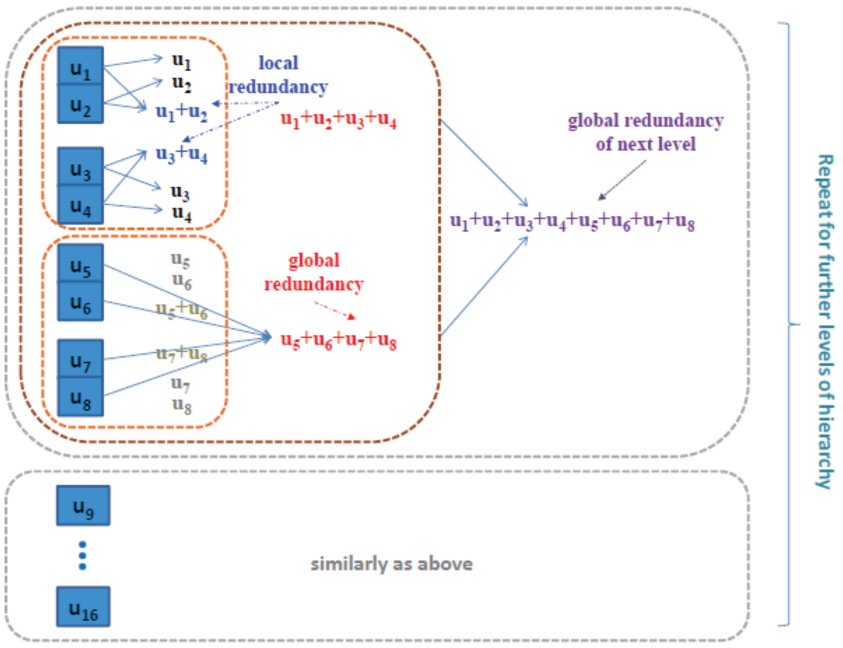

Fig. 6: Hierarchical code example

(3) The process may be iterated multiple times, possibly by using different codes at each iteration. A simple continuation of Example 18 is shown in Figure 6.

(4) Finally, instead of combining two code copies $\mathcal{C}_1$, $\mathcal{C}_2$, one could combine an arbitrary number L of code copies $\mathcal{C}_1, \ldots, \mathcal{C}_L$ at a particular iteration.

This more general setting, where multiple layers can further be built on top of each other in a similar manner, gives rise to Hierarchical codes.

Definition 8. Let $\mathcal{C}_1, \ldots, \mathcal{C}_L$ be L copies of an (n, k) code, $i = 1, \ldots, L$, which map

$$[u_{i,1}, \ldots, u_{i,k}] \mapsto [x_{i,1}, \ldots, x_{i,n}].$$

The construction which consists of iteratively creating a new code on top of $\mathcal{C}_1, \dots, \mathcal{C}_L$, by taking the Lk coefficients and add, on top of the local redundancy (of $(Ln - k)$ coefficients) that already exists, another layer of global redundancy, is referred to as *Hierarchical codes.*

The motivation behind Hierarchical codes is to have a repair fan-in d which is smaller than k, which is the fan-in when using a classical erasure code (say a Reed-Solomon code). This construction leads to the idea of "local" versus "global" redundancy, where local refers to redundancy computed from a small subset of the information symbols. When few errors occur, only local redundancy is used to repair, as illustrated in Example 18, where u_2 and u_3 can be recovered independently using $u_1, u_1 + u_2$ and $u_4, u_3 + u_4$ respectively. However, it is also easy to see that different encoded blocks have different importance. To continue with Example 18, if u_1 and u_2 fail simultaneously, then they can not be reconstructed anymore. If instead u_2 and $u_1 + u_2$ are lost simultaneously, we see that they cannot be rebuilt in parallel, and instead $u_1 + u_2$ needs to be reconstructed first using $u_3 + u_4$ and the global redundancy $u_1 + u_2 + u_3 + u_4$, following which, u_2 can also be repaired.

For this very simple case, we see that even though it provides a way for low-overhead repairs, the fault-tolerance of the Hierarchical code is worse than that of a $(7, 4)$ Hamming code (see Example 4), which in itself is not even an MDS code. There is unfortunately little theory to understand the fault-tolerance of Hierarchical codes (and it seems that both their general definition together with the asymmetric role that different encoded fragments play make such an analysis quite difficult), though one can refer to [11] for some simulation results.

A natural question to ponder is, can we achieve similar localized repairability, while ensuring a good fault tolerance. Pyramid codes [19], which incidentally were proposed before Hierarchical codes have some analogous properties of local and global redundancies, and yet has a structure which makes analytical study of their fault tolerance more tractable. We will thus discuss them next.

7.3 Pyramid and Local Reconstruction Codes

Let us start by explaining the motivating example given in [19]. Take an $(11, 8)$ MDS code, say a Reed-Solomon code (this is possible as long as the size of the finite field used is $|\mathbb{F}_q| \geq 11$) with generator matrix G, so that:

$$[u_1, \dots, u_8]G = [x_1, \dots, x_{11}],$$

where the codeword can be written, in systematic form, as

$$[x_1, \dots, x_{11}] = [u_1, \dots, u_8, c_1, c_2, c_3].$$

A Pyramid code can be built from this base code, by retaining the systematic pieces, and two of the non-systematic pieces (without loss of generality, lets say, c_2, c_3).

Additionally, split the information symbols into two groups $u_1, \dots, u_4$ and $u_5, \dots, u_8$, and compute some more redundancy coefficients for each of the two groups, which is done by picking a first symbol $c_{1,1}$ corresponding to c_1 with $[u_1, \dots, u_4, \mathbf{0}]$, that is

$$c_{1,1} = [u_1, \dots, u_4, \mathbf{0}]G,$$

and $c_{1,2}$ corresponding to c_1 with $[\mathbf{0}, u_5, \dots, u_8]$:

$$c_{1,2} = [\mathbf{0}, u_5, \dots, u_8]G.$$

This results in a $(12, 8)$ code, which in systematic form looks like

$$[u_1, \dots, u_8]G = [u_1, \dots, u_8, c_{1,1}, c_{1,2}, c_2, c_3]$$

where $c_{1,1} + c_{1,2}$ is equal to the original code's c_1:

$$c_{1,1} + c_{1,2} = [u_1, \dots, u_8]G = c_1.$$

We notice that Hierarchical codes are similar to this construction, in that both codes have this notion of “local” redundancy computed from subsets of the information symbols, and “global redundancy” which comes from potentially all the information symbols. However, the design of Pyramid code may be seen as

$\underbrace{u_1 \dots u_{k_1}}_{S_1}$	$\dots$	S_L	$\underbrace{c_{n-m_1} \dots c_n}_{\text{global redundancy}}$	$\underbrace{c_{11} \dots c_{m_0 1}}_{\text{redundancy for } S_1}$	$\dots$	$\underbrace{c_{1L} \dots c_{m_0 L}}_{\text{redundancy for } S_L}$

Fig. 7: A Pyramid codeword.

a *top-down* approach, in that a larger code is reused to build the smaller codes, in contrast to Hierarchical code, where smaller codes are assembled together to form a bigger code. Similarly to Hierarchical codes, if few erasures occur in Pyramid codes, local redundancy can be used to repair them, thus reducing the fan-in (or number of nodes contacted to repair). It is worth noting that while Pyramid codes have such localized repairability property, they were originally designed to enable fault-tolerant and efficient access (degraded reads). The fault-tolerance of Pyramid codes has been analyzed, and will be discussed subsequently in Proposition 3.

More generally, suppose that we start with an (n, k) MDS code, and split the k information symbols $u_1, \dots, u_k$ into L groups $S_1, \dots, S_L$, of size $|S_i| = k_i$, $i = 1, \dots, L$, with $k = k_1 + \dots + k_L$. Set

$$m = n - k$$

that is m is the number of redundancy symbols of the MDS code. Out of these m symbols, keep m_1 of them, and computes $m_0 = m - m_1$ new "local" redundancy symbols for each group S_l (thus for a total of $m_0 L$ such symbols), denoted by $c_{j,l}$, $j = 1, \dots, m_0$, where $c_{j,l}$ is computed as c_j (in the original MDS code) by setting S_1 to S_L to zero but for S_l. A codeword obtained in this fashion defines a *Pyramid code* (see Fig. 7).

The fault-tolerance of the original (n, k) code can be used to understand the fault-tolerance [19] of the derivative Pyramid code.

Proposition 3. A Pyramid code constructed from an (n, k) MDS code can recover arbitrary $m = n - k$ erasures, and each group itself is a $(k_l + m_0, k_l)$ MDS code.

Proof. First note that a Pyramid code constructed from an (n, k)

MDS code encodes k information symbols into a codeword of length $m_1 + m_0L$, where m_1 is the number of redundancy coefficients kept from the original MDS code which forms the global redundancy, while there are L blocks of m_0 coefficients for the local redundancy.

Let us consider an arbitrary failure pattern with m erasures. Now these erasures can occur either in the local redundancy symbols, say r of them, or in the global redundancy symbols and original data, for $m - r$ of them.

Case 1. When $r \geq m_0$, that is, the number of erasures affecting the local redundancy symbols is more than the number of local redundancy symbols for one group S_l.
In this case, assume that all the local redundancy symbols have been erased. From the perspective of the original MDS code, it had m redundancy symbols, and m_1 are still there, while all the rest is lost, which means for this MDS code that it experienced m_0 erasures. Together with the $m - r$ erasures in the rest of the data, the total number of erasures for this MDS code is

$$m_0 + m - r \leq m = n - k$$

since $r \geq m_0$, showing that the original MDS code will recover from these failures.

Case 2. When $r < m_0$, then the number of erasures in the local redundancy symbols is less than the number of local redundancy symbols for one group S_l.

Recall that the knowledge of all $c_{j,l}$ for the same j yields the symbol c_j of the original MDS code. Thus the worst that can happen is that the r erasures affect different j, which means that out of the m_0 c_j, r are erased, and from the point of view of the original MDS code, the total of erasures is

$$m - r + r = m$$

and recovery is possible. This proves the first part of the statement.

For the second part, (i.e., each group itself is a $(k_l + m_0, k_l)$ MDS code) suppose that one group S_l corresponds to a $(k_k + m_0, k_l)$ code which is however not MDS. Then this code must fail to recover some erasure pattern with m_0 failures. Now consider the case where all data blocks are set to zero, but for S_l. Then S_l together with the m_1 global redundancy symbols is equivalent to the original MDS code. Even if these m_1 symbols are erased, the original MDS code can recover the data even if m_0 erasures occur because $m_0 + m_1 = m$ erasures in total. Thus S_l can be recovered, a contradiction. □

A multi-hierarchical extension of basic Pyramid codes exists, where the groups S_i are separated in smaller groups, and the process of keeping some global redundancy blocks, and creating local ones is iterated, which in fact explains the name *Pyramid*. Generalized Pyramid codes, which allow the data blocks to overlap, were also presented in [19]. We let the interested reader refer this paper for further details.

We conclude this section by mentioning that particular examples of Pyramid codes, called *Local Reconstruction Codes*, are in fact used in Microsoft Azure [18]. The main particularity of LRC is that the coefficients of the generator matrix are optimized to belong to a field of small size ($\mathbb{F}_{2^4}$), unlike the originally proposed Pyramid codes.

Example 19. A LRC which encodes $k = 6$ data blocks into 2 local parities and 2 global parities is optimized in [18]. A codeword then has the form

$$(x_0, x_1, x_2, y_0, y_1, y_2, p_x, p_y, p_0, p_1)$$

where the x_i and y_i form the data object, p_x and p_y are the 2 local parities formed respectively from the x_i and y_i, while p_0 and p_1 are the global parities.

7.4 Cross-object Coding

All the coding techniques we have seen so far in this survey, as well as those which will follow address the repairability problem at the granularity of isolated objects that are stored using erasure coding. However, the idea of product code (see Section 7.1) can in fact also be harnessed by carrying out codes over codes, but across multiple objects [7] as shown in Figure 8.

Consider m^1 objects $O_1, \ldots, O_m$ to be stored. For $j = 1, \ldots, m$, object O_j is erasure encoded into n encoded pieces $e_{j1}, \ldots, e_{jn}$, all of them to be stored in mn distinct storage nodes. Additionally, *parity groups* formed by m encoded pieces (with one encoded piece chosen from each of the m objects) can be created, together with a parity piece (or xor), where w.l.o.g, a parity group is of the form $e_{1l}, \ldots, e_{ml}$ for $l = 1, \ldots, n$, and the parity piece p_l is $p_l = e_{1l} + \ldots + e_{ml}$. The parity pieces are then stored in other n distinct storage nodes. Such an additional redundancy is akin to RAID-4.

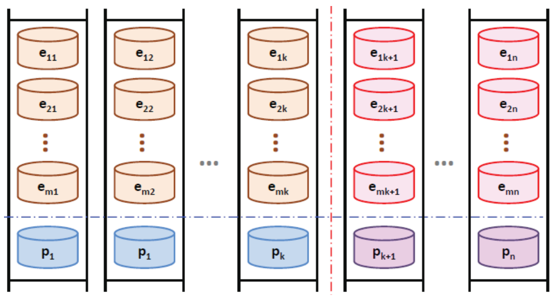

Fig. 8: Redundantly grouped coding: an horizontal layer of coding is performed on each object, using an (n, k) code, while a parity bit is computed vertically across m objects where m is a design parameter

.

This code design, called *Redundantly grouped coding* is similar to a two-dimensional product code in that the coding is done both horizontally and vertically. The design objectives are here somewhat different, namely: (i) the horizontal layer of coding primarily achieves fault-tolerance by using an (n, k) erasure coding of individual objects, while (ii) the vertical single parity check code mainly enables cheap repairs (by choosing a suitable m) by creat-

[1] Note that m has sometimes been used for $n - k$ in the previous chapters, the value of m here is unrelated.

ing RAID-4 like parity of the erasure encoded pieces from different objects.

The number of objects m that are cross-coded indeed determines the fan-in for repairing isolated failures independently of the code parameters n and k. If $m < k$, it can be shown that the probability that more than one failure occurs per column is small, and thus repair using the parity bit is often enough - resulting in cheaper repairs, while relatively infrequently repairs may have to be performed using the (n, k) code. The choice of m determines trade-offs between repairability, fault-tolerance and storage overheads which have been formally analyzed in [7]. Somewhat surprisingly, the analysis demonstrates that for many practical parameter choices, this cross-object coding achieves better repairability while retaining equivalent fault-tolerance as maximum distance separable erasure codes incurring equivalent storage overhead.

Such a strategy also leads to other practical concerns as well as opportunities, such as the issues of object deletion or updates, which need further rigorous investigation before considering them as a practical option.

8
Locally Repairable Codes

The codes proposed in the context of network coding (see Chapter 6) were aiming at reducing the repair bandwidth, and can be seen as the combination of an MDS code and a network code. Hierarchical and Pyramid codes (see Chapter 7) instead tried to reduce the repair degree or fan-in (i.e., the number of nodes needed to be contacted to repair) by using "erasure codes on top of erasure codes". In this chapter, we present some recent families of codes [23, 25, 24, 32] which minimize the repair fan-in d, trying to achieve $d = 2$ or 3. Forcing the repair degree to be small has advantages in terms of repair time and bandwidth, however, it might affect other code parameters (such as its rate, or storage overhead). Recent works start to address these trade-offs [13, 16]. We will present some of the results of [16] after elaborating a few specific instances of locally repairable codes.

The term "locally repairable" is inspired from [13] where the repair degree d of a node is called the "locality d" of a codeword coordinate, and is reminiscent of *locally decodable* and *locally correctable* codes, which are well established topics of study in theoretical computer science. Self-repairing codes [23, 25] were to our knowledge the first (n, k) codes designed to achieve $d = 2$ per repair for up to $\frac{n-1}{2}$ simultaneous failures. Another family of locally repairable codes, namely Punctured Reed-Mueller codes [32] have been proposed very recently that we describe in this chapter. Punctured Reed-Mueller codes are based on polynomial evaluation (similar to one of the instances [23, 25] of Self-Repairing codes) and can achieve a repair degree of either 2 or 3.

8.1 Self-Repairing Codes

We present here the main idea of [23]. Let $\mathbf{o}$ be an object of size B to be stored over a network of n nodes, that is $\mathbf{o} \in \mathbb{F}_{q^B}$, where q is a power of 2, and let k be a positive integer such that k divides B. We can write

$$\mathbf{o} = (o_1, \dots, o_k),\ \ o_i \in \mathbb{F}_{q^{B/k}}$$

which requires the use of an (n, k) code over $\mathbb{F}_{q^{B/k}}$ to generate a codeword $\mathbf{x} = (x_1, \dots, x_n)$, following which each x_i is given to a different node to be stored.

It is known from Reed-Solomon codes (see Chapter 3) that linear coding can be done via polynomial evaluation. Let us recall how: take an object $\mathbf{o} = (o_1, o_2, \dots, o_k)$ of size B, with each o_i in $\mathbb{F}_{q^{B/k}}$, and create the polynomial

$$p(X) = o_1 + o_2 X + \dots o_k X^{k-1} \in \mathbb{F}_{q^{B/k}}[X].$$

Now evaluate $p(X)$ at n elements $\alpha_1, \dots, \alpha_n \in \mathbb{F}_{q^{B/k}}$ the polynomial is evaluated only for nonzero α_is), to obtain the codeword

$$(p(\alpha_1), \dots, p(\alpha_n)),\ \ k < n \leq q^{B/k} - 1.$$

Definition 9. We call *homomorphic self-repairing code*, denoted by $HSRC(n, k)$, the code obtained by evaluating the polynomial

$$p(X) = \sum_{i=0}^{k-1} p_i X^{2^i} \in \mathbb{F}_{q^{B/k}}[X] \tag{8.1}$$

at n non-zero values $\alpha_1, \dots, \alpha_n$ of $\mathbb{F}_{q^{M/k}}$ to get an n-dimensional codeword

$$(p(\alpha_1), \dots, p(\alpha_n)),$$

where $p_i = o_{i+1}$, $i = 0, \dots, k-1$ and each $p(\alpha_i)$ is given to node i for storage.

Since we work over finite fields that contain $\mathbb{F}_2$, recall that all operations are done in characteristic 2, that is, additions are performed modulo 2. Let $a, b \in \mathbb{F}_{q^m}$, for some $m \geq 1$ and $q = 2^t$. Then we have that $(a+b)^2 = a^2 + 2ab + b^2 = a^2 + b^2$ since $2ab \equiv 0$ mod 2, and consequently

$$(a+b)^{2^i} = \sum_{j=0}^{2^i} C_j^{2^i} a^j b^{2^i - j} = a^{2^i} + b^{2^i}, \ i \geq 1. \tag{8.2}$$

The last equality holds by noticing that the binomial coefficient $C_j^{2^i} = \binom{2^i}{j}$ is always a multiple of 2 and thus congruent to 0 modulo 2, except when $j = 0$ and $j = 2^i$.

Lemma 1. Let $a, b \in \mathbb{F}_{q^m}$ and let $p(X)$ be the polynomial given by $p(X) = \sum_{i=0}^{k-1} p_i X^{2^i}$. We have

$$p(a+b) = p(a) + p(b).$$

Proof. If we evaluate $p(X)$ in $a+b$, we get

$$p(a+b) = \sum_{i=0}^{k-1} p_i (a+b)^{2^i} = \sum_{i=0}^{k-1} p_i (a^{2^i} + b^{2^i})$$

by (8.2), and

$$p(a+b) = \sum_{i=0}^{k-1} p_i a^{2^i} + \sum_{i=0}^{k-1} p_i b^{2^i} = p(a) + p(b).$$

□

It is the choice of this polynomial in (8.1) that enables local repairs. Decoding is achieved by either Lagrange interpolation or by considering a system of linear equations, subject to the constraint that $k \leq B/k$.

Example 20. Take a data file $\mathbf{o} = (o_1, \ldots, o_{12})$ of $B = 12$ bits ($q = 2$), and choose $k = 3$ fragments. We have that $B/k = 4$, which satisfies $k = 3 \leq B/k = 4$.

The file $\mathbf{o}$ is cut into 3 fragments $\mathbf{o}_1 = (o_1, \ldots, o_4)$, $\mathbf{o}_2 = (o_5, \ldots, o_8)$, $\mathbf{o}_3 = (o_9, \ldots, o_{12}) \in \mathbb{F}_{2^4}$. Let w be an element of $\mathbb{F}_{2^4}$, such that $w^4 = w + 1$. The polynomial used for the encoding is

$$p(X) = \sum_{i=1}^{4} o_i w^i X + \sum_{i=1}^{4} o_{i+4} w^i X^2 + \sum_{i=1}^{4} o_{i+8} w^i X^4.$$

The n-dimensional codeword is obtained by evaluating $p(X)$ in n elements of $\mathbb{F}_{2^4}$, $n \leq 15$.

For $n = 4$, if we evaluate $p(X)$ in w^i, $i = 0, 1, 2, 3$, then the 4 encoded fragments $p(1), p(w), p(w^2), p(w^3)$ are $\mathbb{F}_2$-linearly independent and there is no local repair possible.

Now for $n = 7$, and say, $1, w, w^2, w^4, w^5, w^8, w^{10}$, we get:

$$(p(1), p(w), p(w^2), p(w^4), p(w^5), p(w^8), p(w^{10})).$$

Suppose node 5 which stores $p(w^5)$ becomes unavailable. A newcomer can reconstruct $p(w^5)$ by asking for $p(w^2)$ and $p(w)$, since

$$p(w^5) = p(w^2 + w) = p(w^2) + p(w).$$

Table 8.1 shows other examples of missing fragments and which pairs can reconstruct them, depending on if 1, 2, or 3 fragments are missing simultaneously.

As for decoding, since $p(X)$ is of degree 4, a node that wants to recover the data needs $k = 3$ $\mathbb{F}_2$-linearly independent fragments, say $p(w), p(w^2), p(w^3)$, out of which it can generate $p(aw + bw^2 + cw^3)$, $a, b, c \in \{0, 1\}$. Out of the 7 non-zero coefficients, 5 of them are enough to recover p. Finally, the rate of this code is $3/7 \simeq 0.43$ (or storage overhead $7/3$).

It is important to note that these codes are not MDS, thus to evaluate their fault-tolerance, a static resilience analysis is needed (see Chapter 4), which can be found in [23].

Another construction of self-repairing codes, called *projective self-repairing codes* was proposed in [24], from which we present

missing fragment(s)	pairs to reconstruct missing fragment(s)
$p(1)$	$(p(w),p(w^4))$;$(p(w^2),p(w^8))$;$(p(w^5),p(w^{10}))$
$p(w)$	$(p(1),p(w^4))$;$(p(w^2),p(w^5))$;$(p(w^8),p(w^{10}))$
$p(w^2)$	$(p(1),p(w^8))$;$(p(w),p(w^5))$;$(p(w^4),p(w^{10}))$
$p(1)$ and $p(w)$	$(p(w^2),p(w^8))$ or $(p(w^5),p(w^{10}))$ for $p(1)$ $(p(w^8),p(w^{10}))$ or $(p(w^2),p(w^5))$ for $p(w)$
$p(1)$ and $p(w)$ and $p(w^2)$	$(p(w^5),p(w^{10}))$ for $p(1)$ $(p(w^8),p(w^{10}))$ for $p(w)$ $(p(w^4),p(w^{10}))$ for $p(w^2)$

Table 8.1: Ways of reconstructing missing fragment(s) in Example 20

an example below, to illustrate that very different techniques may be used to construct codes achieving low repair degree.

Example 21. Suppose you have $n = 5$ nodes to store the data, an object of size $B = 4$, $k = 2$ (meaning the data collector contacts 2 nodes to retrieve the data) and a storage capacity of $\alpha = 2$. Let us denote by N_i, $i = 1,\dots,5$ the 5 storing nodes, and by $\mathbf{o} = (o_1, o_2, o_3, o_4)$ the object to be stored. We propose the following code:

node	basis vectors	data stored
N_1	$v_1 = (1000),\ v_2 = (0110)$	$\{o_1, o_2 + o_3\}$
N_2	$v_3 = (0100),\ v_4 = (0011)$	$\{o_2, o_3 + o_4\}$
N_3	$v_5 = (0010),\ v_6 = (1101)$	$\{o_3, o_1 + o_2 + o_4\}$
N_4	$v_7 = (0001),\ v_8 = (1010)$	$\{o_4, o_1 + o_3\}$
N_5	$v_9 = (1100),\ v_{10} = (0101)$	$\{o_1 + o_2, o_2 + o_4\}$

By basis vectors v_1 and v_2, we mean that the object is encoded by using v_1, v_2 but also $v_1 + v_2$, namely

$$\mathbf{o}v_1^T,\ \mathbf{o}v_2^T.$$

Since $\mathbf{o}v_1^T + \mathbf{o}v_2^T = \mathbf{o}(v_1^T + v_2^T)$, the node only needs to store $\mathbf{o}v_1^T$, $\mathbf{o}v_2^T$ though it does "know" $\mathbf{o}(v_1^T + v_2^T)$. Let us give an example of repair: If say N_1 fails, the data pieces o_1 (corresponding to the basis vector (1000)) and o_2+o_3 (corresponding to the basis vector (0110)) are lost. A new node joining the network can contact nodes N_3 and N_4, from which it gets respectively $v_5 = (0010)$, $v_6 = (1101)$ and $v_7 = (0001)$, $v_8 = (1010)$. Now $v_8 + v_5$ gives (1000) while $v_8 + (v_6 + v_7)$ gives (0110). Unlike Homomorphic Self-Repairing codes, this code is not atomic, and instead comprises of α pieces per node. Thus, similarly to Regenerating codes, one could also expect to regenerate an encoded block piece-by-piece, by contacting more (larger d) number of nodes. For instance, N_1 can also be repaired by downloading o_2 from N_2, o_3 from N_3 and $o_1 + o_3$ from N_4. By contacting $d = 3$ nodes, the two lost pieces of N_1 are repaired by downloading only three units of data.

For data reconstruction: by contacting $k = 2$ nodes, the data collector gets 4 linear equations to decode an object of size $B = 4$.

Note that though this particular code happens to be MDS, other instances are usually not MDS (see [24] for more details on the fault-tolerance of these codes). We further notice that though the above code is not systematic, one can however contact $B = \alpha k$ specific nodes (instead of k), namely those storing as pieces each of the canonical basis vectors (or unit vectors) to reconstruct the object in a systematic manner. Systematic codes have practical benefits, since it simplifies the object retrieval process.

8.2 Punctured Reed-Mueller Codes

A family of codes with repair degree 2 or 3 based on evaluations of a multivariate polynomial is given in [32]. Let p be a prime. Consider an object $\mathbf{o} = (o_1, \ldots, o_B) \in \mathbb{F}_p^B$ of length B, and the multivariate polynomial

$$f(X_1, \ldots, X_B) = \sum_{i=1}^{B} o_i X_i.$$

A codeword is obtained by evaluating $f(X_1, \ldots, X_B)$ in the points

$$\mathbf{a}_i, \mathbf{a}_i + t((p-1)\mathbf{a}_i + \mathbf{a}_j) \in \mathbb{F}_p^B,$$

for $i = 1, \ldots, N$, $i < j \leq N$, $1 \leq t \leq 1 + L$, where N and L are design parameters. Note that given i and j, we can set $\mathbf{h} = (p-1)\mathbf{a}_i + \mathbf{a}_j$, and the polynomial is evaluated in

$$\mathbf{a}_i, \mathbf{a}_i + t\mathbf{h},$$

where t goes from 1 to $1 + L$. These points correspond to $L + 2$ points on a line.

Each coefficient of the codeword is stored at a distinct node. Suppose that the node storing $\mathbf{a}_i$ fails. The repair goes as follows: two nodes corresponding to two other points on the same line are contacted by a newcomer, which will obtain, say

$$f(\mathbf{a}_i + t_1\mathbf{h}),\ f(\mathbf{a}_i + t_2\mathbf{h})$$

for two distinct values t_1 and t_2 of t from 1 to $1 + L$. Consider the polynomial $g(X) = f(\mathbf{a}_i + X\mathbf{h})$. Equivalently, we might say that the newcomer obtained two evaluations of this polynomial

$$g(t_1),\ g(t_2).$$

Since g is a polynomial of degree 1 in X, these two evaluations are enough for the newcomer to obtain $g(X)$, and by computing $g(0)$, the newcomer obtains the missing data. This construction allows for a repair degree of $d = 2$ (as long as no more than $L - 1$ failures occurred).

This construction actually comes from a punctured $RM(1, B)$ Reed-Mueller code, and a similar one using a punctured $RM(2, B)$ Reed-Mueller code gives $d = 3$. We let the interested reader refer to [32] for further details.

8.3 Bounds and Trade-Offs

Suppose that an object of length k is encoded into a codeword, and the encoded data is distributed over n storage nodes, each storing α symbols of data. Assume further that if one such storage node fails, the lost redundancy can be replenished at a newcomer by contacting d live nodes, and downloading β amount of data from each. The two theorems below describe a trade-off between the storage overhead $n\alpha/k$ and the repair degree d.

Theorem 5. (1) If $\alpha = \beta$, then $(k/n\alpha) \leq d/(d+1)$, and equality holds for MDS codes with $n = d + 1$.
(2) If $\alpha = d\beta$, then $(k/n\alpha) \leq 1/2$ for $d > 1$.

Theorem 6. For the case $d = 2$, we have that

$$R \leq \alpha + \beta/(3\alpha).$$

More precisely, if $n = 3q - e$ with $e \in \{0, 1, 2\}$ then

$$k \leq q\alpha + (q - e)\beta.$$

Note that these bounds hold for functional repair (that is, when the data does not have to be repaired bit-by-bit), and it is unlikely that exact repair mechanisms can achieve them, while the three previously described code constructions achieve exact repair.

9 Concluding Remarks

The continuously increasing amount of data to be stored nowadays has triggered a shift in the way data is being stored in networked distributed storage systems (NDSS), which have started to adopt erasure codes to reduce their storage overhead while maintaining high fault tolerance. While the main industry players have already introduced coding techniques of their choice, different research communities are also presenting new coding solutions, best suited for the need of NDSS. The goal of this survey was two fold: (1) to give some basic knowledge on storage systems and coding techniques, and (2) to present some of the new families of codes that have been recently proposed, focusing on the repairability aspect of the code. These codes were classified into 3 groups, depending on what is the main design criterion for better repairability: minimize the repair bandwidth (using network coding techniques), reduce the repair fan-in (combining two erasure codes), and finally minimize the repair fan-in. While much is left to be done on codes with good repairability, we conclude this survey by mentioning a few of the numerous other open research directions.

9.1 Future Directions

Most of the codes discussed in this survey were analyzed theoretically for a single object in isolation. In order to get a better understanding of the actual performance of these codes, it will be valuable to analyze them in more realistic settings, such as under different failure patterns, and in a network where each storage node stores multiple objects (e.g. [27]). Additionally, some other aspects of erasure code based storage systems which seem worthy of further study include the problems of inserting erasure encoded data, or migrating replicated data into erasure coded ones, as well as the resilience of such systems against Byzantine failures (rather than just fail-stop failure model of erasures). We elaborate on these next.

Data Insertion. A fairly unexplored aspect of using erasure codes in NDSS is the problem of data insertion. When using replication, the data source sends the whole information to a storage node, which stores a copy of the data and can just pipeline the information to other nodes where the data is replicated. Things are not that trivial for erasure coded data: in a naive solution, the node that uploads the encoded pieces to other nodes needs to first have the complete data, then it carries out all the computational tasks, and finally bears the communication cost to distribute all the encoded pieces. Mechanisms to create the encoded pieces in a distributed fashion, using the network resources of multiple system nodes, either immediately when data is being introduced in the system or by migrating replicated data into erasure coded ones, are some relatively new directions being explored in that context (e.g. [28], [8]).

Security. The problem of securely storing data in an erasure coding based NDSS has already attracted a fair amount of interest. We will mainly mention a few aspects that are specific to the kind of codes presented here. The most basic approach is to look into error correcting capacity of some codes, which provides natural protection against undesired mutation. However, with the novel coding approaches for repair, new vulnerabilities also come into being. For instance, in network coding based techniques, if data is polluted upstream, content at many downstream storage nodes may be affected, e.g., [30, 26].

On a different note, when data storage is outsourced to a third party, the data owner may need to carry out proofs of data possession and retrievability (PoR). Erasure codes provide a possibility to do so in an efficient manner, and yet, over time, for mutable content, using deterministic coding structure may reveal information that can be exploited by the third party to falsely pass the verifications. New codes based on randomization [39] are thus being investigated for robust PoR.

References

[1] R. Alshwede, N. Cai, S.-Y.R. Li and R.W. Yeung, "Network information flow", *IEEE Transactions on Information Theory*, vol. 46, no. 4, 2000. year=2000,

[2] D. Beaver, S. Kumar, H. C. Li, J. Sobel, and P. Vajgel. "Finding a needle in Haystack: Facebooks photo storage," *OSDI 2010.*

[3] R. Bhagwan, K. Tati, Y-C. Cheng, S. Savage, G.M. Voelker, "Total Recall: System Support for Automated Availability Management", *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2004.

[4] R. E. Blahut, "Algebraic Codes for Data Transmission", *Cambridge.*

[5] `www.cleversafe.com`

[6] A. Datta and F. Oggier, "An Overview of Codes Tailor-made for Networked Distributed Storage Systems", preprint, available on Arxiv.

[7] A. Datta and F. Oggier, "Redundantly Grouped Cross-object Coding for Repairable Storage", APSys 2012.

[8] A. Datta, L. Pamies-Juarez, F. Oggier, "On Data Insertion and Migration in Erasure-Coding Based Large-Scaled Storage

Systems", to appear in *ICDCIT 2013*.

[9] A. G. Dimakis, P. B. Godfrey, Y. Wu, M. Wainwright and K. Ramchandran, "Network Coding for Distributed Storage Systems" *IEEE Transactions on Information Theory*, Vol. 56, Issue 9, Sept. 2010.

[10] A. G. Dimakis, K. Ramchandran, Y. Wu, C. Suh, "A Survey on Network Codes for Distributed Storage", *The Proceedings of the IEEE*, March 2011, Vol 99, No 3.

[11] A. Duminuco, E. Biersack, "Hierarchical Codes: How to Make Erasure Codes Attractive for Peer-to-Peer Storage Systems" , *Eighth International Conference on In Peer-to-Peer Computing, P2P 2008.*

[12] P. Elias, "Error-free coding", *Transactions on Information Theory*, vol. 4, no. 4, September 1954.

[13] P. Gopalan, C. Huang, H. Simitci, S. Yekhanin, "On the locality of codewords symbols", *Electronic Colloquium on Computational Complexity (ECCC)* vol. 18, 2011.

[14] `hadoop.apache.org/`

[15] http://wiki.apache.org/hadoop/HDFS-RAID

[16] H. D. L. Hollmann, "Storage codes - coding rate and repair locality".

[17] Y. Hu, Y. Xu, X. Wang, C. Zhan and P. Li, "Cooperative Recovery of Distributed Storage Systems from Multiple Losses with Network Coding", *IEEE Journal on Selected Areas in Communications*, vol. 28, no 2, February 2010.

[18] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Lin, S. Yekhanin, "Erasure Coding in Windows Azure Storage".

[19] C. Huang, M. Chen, and J. Li, "Pyramid Codes: Flexible Schemes to Trade Space for Access Efficiency in Reliable Data Storage Systems", *Sixth IEEE International Symposium on Network Computing and Applications, 2007. NCA 2007.*

[20] A.-M. Kermarrec, N. Le Scouarnec, G. Straub, "Repairing Multiple Failures with Coordinated and Adaptive Regenerating Codes", in the proceedings of the 2011 International Symposium on Network Coding (NetCod 2011).

[21] R. Koetter, M. Medard, "An Algebraic Approach to Network Coding", *IEEE/ACM Transactions on Networking*, vol. 11, 2001.

[22] W.K. Lin, D.M. Chiu, Y.B.Lee, “Erasure code replication revisited”, *P2P 2004.*

[23] F. Oggier, A. Datta, “Self-repairing Homomorphic Codes for Distributed Storage Systems”, INFOCOM 2011.

[24] F. Oggier, A. Datta, “Self-Repairing Codes for Distributed Storage – A Projective Geometric Construction”, ITW 2011.

[25] F. Oggier, A. Datta, “Homomorphic Self-Repairing Codes for Agile Maintenance of Distributed Storage Systems”, `http://arxiv.org/abs/1107.3129`

[26] F. Oggier, A. Datta, “Byzantine Fault Tolerance of Regenerating Codes”, *P2P 2011.*

[27] L. Pamies-Juarez, F. Oggier, A. Datta, “An Empirical Study of the Repair Performance of Novel Coding Schemes for Networked Distributed Storage Systems”, `http://arxiv.org/abs/1206.2187`.

[28] L. Pamies-Juarez, A. Datta, F. Oggier, “RapidRAID: Pipelined Erasure Codes for Fast Data Archival in Distributed Storage Systems”, `http://arxiv.org/abs/1207.6744`

[29] D. A. Patterson, G. Gibson, R. H. Katz "A case for redundant arrays of inexpensive disks (RAID)" *ACM SIGMOD International Conference on Management of Data*, 1988.

[30] S. Pawar, S. El Rouayheb and K. Ramchandran, "Securing Dynamic Distributed Storage Systems against Eavesdropping and Adversarial Attacks," *IEEE Transactions on Information Theory (Special Issue on Facets of Coding Theory: from Algorithms to Networks)*, Vol. 57, No. 9, Sep 2011.

[31] K. V. Rashmi, N. B. Shah, P. Vijay Kumar, K. Ramchandran, “Explicit Construction of Optimal Exact Regenerating Codes for Distributed Storage”, *Allerton 2009.*

[32] A. S. Rawat, S.Vishwanath, “On Locality in Distributed Storage Systems”, *ITW 2012.*

[33] I. Reed, G. Solomon, “Polynomial codes over certain finite fields”, *Journal of the Society of Industrial and Applied Math-*

ematics, 1960.

[34] K. W. Shum, "Cooperative Regenerating Codes for Distributed Storage Systems", *ICC 2011*, available at `http://arxiv.org/abs/1101.5257`.

[35] K. W. Shum, Y. Hu, "Cooperative Regenerating Codes", preprint, available at `http://arxiv.org/abs/1101.5257`

[36] Y. Wu, A. G. Dimakis, "Reducing Repair Traffic for Erasure Coding-Based Storage via Interference Alignment", *ISIT 2009.*

[37] `www.wuala.com/`

[38] R. W. Yeung, S.-Y. R. Li, N. Cai and Z. Zhang, "Network Coding Theory: Single Sources", *Foundations and Trends in Communication and Information Theory*, Volume 2, Issue 4.

[39] E. Stefanov, M. van Dijk, A. Oprea and A. Juels, "Iris: A Scalable Cloud File System with Efficient Integrity Checks", Cryptology ePrint Archive, Report 2011/585, 2011.

Index