Coding Techniques for Distributed Storage Systems

Frédérique Oggier
Division of Mathematical Sciences
Nanyang Technological University

March 3, 2012

Contents

Chapter 1

Mathematical and Coding Preliminaries

We will spend some time in this lecture to present some basic techniques from classical coding theory. 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: $0+0=0$, $0+1=1$, $1+1=0$, $0\cdot 1=0$ (XOR operations), $0\cdot 0=0$ and $1\cdot 1=1$. By $\mathbf{u}\in\mathbb{F}_2^k$ we mean a k-dimensional row vector $\mathbf{u}=[u_1,\ldots,u_k]$ with coefficients in $\mathbb{F}_2$.

1.1 Generator and Parity Check Matrix

A *code* is 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. It is standard to write an (n,k) code, to emphasize the parameters of the code, where n is called the length of the code, and k its dimension.

Definition 1.1. The ratio k/n is called the rate of an (n,k) code. It represents the amount of information per amount of redundancy.

Example 1.1. The repetition code takes one information symbols 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 1.2. 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

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

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 1.3. Let us take again the repetition code of Example 1.1. It is obvious that if we see a codeword of the form $\mathbf{x} = [u, \ldots, u]$, then it is a codeword from the repetion code. This is not a mathematically satisfying answer though! because 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 code (n, k) a $(n-k) \times n$ 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

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

1.2 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 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. Our next question is thus: given a linear code with parameters (n, k), what is the best possible number of erasure a code can support?

To bring an answer, we need the notion of *Hamming distance*, named in honour of Richard Hamming.

Definition 1.2. 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 1.4. If $\mathbf{x} = [1, 0, 0, 1]$ and $\mathbf{y} = [0, 0, 0, 1]$, then $d(\mathbf{x}, \mathbf{y}) = 1$.

Definition 1.3. Given a vector $\mathbf{x}$, the Hamming weight of $\mathbf{x}$ is the number of non-zero coefficients of $\mathbf{x}$, denoted by $wt(\mathbf{x})$.

Figure 1.1: Richard Hamming (1915-1998)

Note that

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

Example 1.5. 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 1.4. 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 it is needed to emphasize the minimum distance of a code, 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 1.6.

1. For the repetition code of Example 1.1, there are two codewords $[0, 0, 0]$ and $[1, 1, 1]$ and thus $d_H(\mathcal{C}) = 3$.

2. For the code of Example 1.2, 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 $d_H(\mathcal{C}) = 2$.

We can now link the capability of a code to tolerate erasures to its minimum distance. Suppose a codeword $\mathbf{x} = (x_1, \ldots, x_n)$ of length n has some coefficients

erased. How many can be recovered? Well, 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 1.7. 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 $(0,0,*)$, 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 explains below.

Theorem 1.1. *If H is the parity check matrix of a code $\mathcal{C}$ of length n, then the $d_H(\mathcal{C}) = d$ 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. □

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

Theorem 1.2. (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. □

Corollary 1.3. *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 MDS code has $d_H(\mathcal{C}) = n - k + 1$. □

A code achieving the Singleton bound is called *maximum distance separable (MDS)*. Because MDS codes are providing with the best erasure protection given k and n, they are usually the preferred erasure codes. So let us see what are examples of such codes.

Examples 1.8.

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 following $(7, 3)$ code is not MDS:

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

   The minimum distance of this code is 4, while $n - k + 1 = 7 - 3 + 1 = 5$.

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

Proposition 1.4. *A binary $(n, k, n - k + 1)$ linear code cannot exist for $k \neq 1$ and $k \neq n - 1$.*

Proof. By Theorem 1.1, we know that to such a code corresponds 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. □

So what is the problem? the problem is that we do not have enough "linear combinations" because 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.

1.3 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 $p(X)$. What $\mathbb{F}_2[X]/(p(X))$ contains are 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 1.9.

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 degree at degree 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 is 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. The converse is left in exercises.

Figure 1.2: Richard Hamming (1915-1998)

This can be made more formal, but 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 that keeping the polynomial notation.

Example 1.10. 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, most likely the most famous ones, the class of Reed-Solomon codes.

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 over some finite field $\mathbb{F}_q$. Consider the polynomial

$$f(X) = u_1 + u_2 X + \ldots + u_k X^{k-1}.$$

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 1.11. 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

$$f(X) = u_1 + u_2 X \in \mathbb{F}_2[X]$$

and a codeword looks like

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

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 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 of 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.

1.4 Exercises

Exercise 1. We saw that $\mathbb{F}_2 = \{0, 1\}$ with addition and multiplication modulo 2 has a field structure, meaning roughly that the following arithmetic holds: addition, subtraction, multiplication which is commutative, and division by a non-zero element. If we pick $\{0, 1, \ldots, n\}$ with addition and multiplication modulo n instead of 2, do we still have a finite field structure?

Exercise 2. Compute the parity check matrix of the code of Example 1.2.

Exercise 3. Compute the generator matrix and the parity check matrix of the ternary version of the code of Example 1.2, that is where operations are performed modulo 3 instead of modulo 2.

Exercise 4. Give a second proof of the Singleton bound by looking at the weight of a codework with only one nonzero information symbol.

Exercise 5. Prove that if $p(X) \in \mathbb{F}_2[X]$ is irreducible, then $\mathbb{F}_2[X]/(p(X))$ is a field.

Project 1. "Reed-Solomon codes in my data center". This question is more a "classical" coding theory type of question, and most answers can be found in the literature.

1. Explain in detail how Reed-Solomon codes can be made systematic.
2. Provide one worked out example of such a Reed-Solomon code.
3. Since repair is done by retrieving the data (decoding), you might want to discuss a bit the decoding of Reed-Solomon code.

This is useful to understand what form of Reed-Solomon codes is actually relevant for "real" storage applications.

Chapter 2

Coding for NDSS

Let us start by mentioning some motivating numbers: a study sponsored by the information storage company *EMC* estimated that the world's data is more than doubling every two years, reaching 1.8 zettabytes (1 zettabyte = 10^{21} bytes) of data to be stored in 2011.[1]

This includes all kinds of digital data continuously being generated by individuals, business and government organizations, who all need scalable solutions to store data reliably (and securely, though we will not address this issue). Storage technology has been evolving fast in the last quarter of a century to meet the numerous challenges posed in storing an increasing amount of data.

Among the many challenges to be addressed, the one on which we will focus is that of *redundancy*, or, informally speaking (this will be made more precise below), how to make sure that the stored data will be available whenever needed, no matter how long it is stored, irrespectively of the potential failures that the storage devices might have encountered.

A common practice to realize redundancy is to keep three copies of the data to be stored, called *3-way replication*: when one copy is lost, the second copy is used to regenerate the first one, and hopefully, not both the remaining copies are lost before the repair is completed. Let us pay attention to this process, since there is an important concept which actually got introduced here: from a user point of view, what matters is that whenever the data is needed, it is available. We will refer to this process as *data retrieval* or *data reconstruction*. However from a system point of view, it is also important that whichever redundancy is decided for a given data, this level of redundancy must remain over time (even though one copy is enough to be retrieved). This is sometimes called *repair* or *redundancy replenishment*.

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

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

be estimated not only in terms of the hardware, but also of real estate 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.

What will interest us for the rest of these lectures is the trade-off between fault-tolerance and efficiency in storage space, from the point of view of erasure codes. However, to well understand what is at stake here, we need to be a bit more precise on what kind of storage systems we consider, and what are the different requirements specific to these systems that we should keep in mind when thinking of code design. Note that the shift from replication towards erasure codes is by now not only a research topic but in fact a reality, since the new version of Google's file system, Microsoft's Windows Azure Storage as well as other storage solution companies such as CleverSafe and Wuala have started to integrate erasure codes.

2.1 Specificities of Coding for NDSS

In 1988, RAID (Redundant Arrays of Inexpensive Disks) was proposed [7] which combines multiple storage disks (typically from two to seven) to realize a single logical storage unit. Data is stored redundantly, using replication, or more recently erasure codes, the same erasure codes as discussed in the previous chapter. Such redundancy makes a RAID logical unit significantly more reliable than the individual constituent disks. While RAID has evolved and stayed an integral part of storage solutions till date, new classes of storage technology have emerged, where multiple logical storage units (we will call them "storage nodes" for short) are assembled together to scale out the storage capacity of a system. The massive volume of data involved means that it would be extremely expensive, if not impossible, to build single pieces of hardware with enough storage. By the term "networked", we refer to these storage systems which pool resources from multiple interconnected storage nodes (which in turn may use RAID). The data is distributed across these interconnected storage units and hence the name **networked distributed storage systems (NDSS)**. The challenges of designing codes for RAID systems could be the topic of a lecture of its own, and will not be treated here.

The type of storage systems we have in mind are typically (1) data centers and (2) peer-to-peer storage/backup systems, which tend to differ on their size and topology.

- **Size:** while data centers comprise of thousands of nodes, individual clusters such as that of Google File System (GFS) are formed out of hundreds up to thousands of nodes. P2P systems like Wuala in contrast form swarms

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

of tens to hundreds of nodes for individual files or directories, but may distribute such swarms arbitrarily out of hundreds of thousands of peers.

- **Topology:** while P2P systems are geographically distributed and connected through an arbitrary topology, data center interconnects have well defined topologies and are either collocated or distributed across a few geographic regions. Furthermore, individual P2P nodes may frequently go offline and come back online (temporary churn), creating an unreliable and heterogeneous connectivity. On the contrary, data centers use dedicated resources with relatively infrequent temporary outages.

Both data centers and peer-to-peer storage systems however share as common characteristics the need for efficient maintenance mechanisms: due to their size, failure of a significant subset of the constituent nodes, as well as other network components *is a norm rather than the exception*. Thus redundancy is needed, as well as replenishment of this same redundancy in case of failures. There are many ingredients playing a role in this process:

1. **Eager vs lazy repair:** First there is a need to distinguish when repair should be triggered: in P2P systems, a lazy approach (where several failures are tolerated before triggering repair) can avoid unnecessary repairs since nodes are temporarily offline. Data centers might opt for immediate repairs.

2. **One fault vs several faults, parallel vs serialized:** Even proactive repairs can lead to cascading failures[3]. Thus ability to repair multiple faults simultaneously, and more generally repair time, are essential.

3. **Repair bandwidth cost:** Bandwidth is a critical shared resource, both at the network's edges and within the interconnect. Consequently the repair process should try to minimize the amount of bandwidth needed.

4. **Repair fan-in:** Finally, repair means getting data from other nodes, which might or not be able to help. This means that the number of nodes involved should also be considered. We will use the term fan-in to refer to the number of nodes contacted by a node to perform repair.

5. **Exact vs functional repair:** The notion of functional repair was introduced in [12], where the idea is that the lost data does not have to be recovered exactly bit by bit, but in fact repairing some other data that provides the same amount of redundancy could be an option as well.

There are numerous other aspects which all together determine an actual system's performance, such as: data placement, disk accesses, meta-information management to coordinate the network, interferences among multiple objects contending for resources, to name a few ones. The coding techniques that we will present below will focus on some aspects but will not cover everything!

[3]For example `http://storagemojo.com/2011/04/29/amazons-ebs-outage/`

2.2 Network Model and Classical Erasure Codes

Let us now define the network model that we will consider for the rest of these lectures. We consider a network with N nodes, a source S that uploads data that needs to be stored in some nodes of the network, and a data collector DC that will access some nodes so as to obtain the data back. Both the source and the data collector are assumed to have infinite capacity links, to respectively upload and download the data.

The data is assumed to be one file, also called object, 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.

When one node (newcomer) participates in the repair process, it is assumed to connect to d nodes, sometimes called live nodes, and can download data from each of these d nodes. Every link has a download capacity of β symbols.

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). 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, so that in particular the storage capacity α of each node must satisfy

$$\alpha \geq B/k. \tag{2.1}$$

After the source S is done uploading the data, n ouf the N nodes are having one encoded fragment x_i of the original file.

The question is now: which (n, k) code should be used to provide redundancy? The rate of the code k/n plays an important role: it defines the *storage overhead*, usually defined as n/k. Next the values of n and k themselves do matter. Typical values of n and k depend on the different environments: for data centers, the number of temporary failures is relatively low, thus small (n, k) values such as $(9, 6)$ or $(13, 10)$ (with a respective overhead of $9/6 = 1.5$ and $13/10 = 1.3$) are generally fine. In P2P systems larger parameters like $(517, 100)$ are desirable (see for example Wuala) to guarantee availability since nodes frequently go temporarily offline.

Note that the replication code (see Example 1.1) is nothing else than replication. The $(3, 1)$ repetition code

$$u[1, 1, 1] = [u, u, u]$$

is exactly 3-way replication. Reed-Solomon codes are good candidates to try, in fact, these codes have been extensively studied in the context of the CD, and both the new Google file system as well as Microsoft Azure claim to have integrated Reed-Solomon codes. Reed-Solomon codes are MDS codes, and can be seen in a black box model as (n, k) codes with the property that out of any k encoded fragments, the original data can be retrieved. This black box model as been well studied in the P2P community as well (see the work done in OceanStore).

Let us here briefly explain how a "classical erasure code" (that is a Reed-Solomon code, or the black box model we just described) would be used in a networked distributed storage system (NDSS). When one object is stored using an erasure code and each encoded symbol is stored at a different node, then the object stays available as long as the number of node failures does not exceed the code recovery capability. In particular, if no failure occurs, a data collector DC can always retrieve the data from any choice of k nodes storing data. From this point of view, it thus makes sense to assume the use of an MDS code, since we saw in the previous chapter (recall the Singleton bound) that MDS codes offer the best recovery from erasures (a node going offline is seen as an erasure from the code point of view).

Let us be a bit more precise in illustrating the gain of using, say a Reed-Solomon code, instead of replication (or repetition code). Let f be the failure probability of individual storage nodes. The actual number of independent node failures is binomially distributed, hence the probability of losing an object if an (n, k) MDS erasure code is

$$\sum_{j=1}^{k} \binom{n}{n-k+j} f^{n-k+j}(1-f)^{k-j},$$

in contrast it is f^r with r-way replication. For example, 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 10^{-3} and $\sim 3 \cdot 10^{-6}$ respectively.

Where is the catch then? It is that 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 (that is B symbols) in one place (either by storing 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 of size B/k. This in turn means that the fan-in (or number of nodes that are contacted for repair) is $d = k$ for the amount of data transferred to be

$$k(B/k) = B$$

which is the minimum amount of bandwidth needed to recover the whole file. This strategy is a waste of communication bandwidth if only one encoded fragment is needed, however it does make sense in the case of lazy repair.

We will conclude this chapter by summarizing what are the main code properties we are interested in:

1. **Good fault-tolerance**: this is the purpose of introducing redundancy to start with.

2. **Reasonable storage overhead**: this is one reason to move from replication to more complicated erasure codes.

3. **Low communication cost**: high communication cost is the main drawback of using Reed-Solomon codes, if bandwidth is not an issue, Reed-Solomon codes achieve the two properties mentioned above.

4. **Fast repair time**: this is needed to prevent cascading effects. Repair should be done fast, possibly in parallel to speed up the process, and in fact simultaneous failures should be tolerated.

5. **Moderate fan-in**: nodes might not always be available to help the repair, if too many nodes are needed, having some of them responding too slowly may delay the repair, which in turn leaves the system vulnerable to further faults.

Chapter 3

Network Coding Based Techniques

We will now present an approach that aims at minimizing the repair bandwidth cost, inspired by network coding techniques.

3.1 Network Coding and Information Flow

Let us mention very briefly what is the idea behind network coding. When data has to be transmitted over a network, from a source to different sinks, intermediate nodes serve as relays, forwarding the information from one node to another. The idea of network coding is to let the intermediate nodes manipulate the packets they receive, and typically the operations are linear, that is: every intermediate node computes and forwards a linear combination of the packets it receives. The advantage of doing coding instead of forwarding is to gain in throughput: it was shown that in some network scenarios, the actual amount of information that the network can transmit is higher when using coding. The problem is that the sinks must decode the data, that is recover the original messages. Network coding studies the maximum amount of data than can be transmitted depending on the type of networks considered, design of network codes, etc

What matters for our context is the idea of information flow: the network is seen as a graph, data flows from the source to the sinks through this graph, and the amount of information that is actually transmitted can be determined by looking at a min-cut in the graph. Similarly, in the storage context, we have a source S that uploads some data to be stored in the network, and a data collector DC that downloads some of it. Information is flowing from S to DC through intermediate nodes:

$$\mathrm{S} \overset{\infty}{\to} \mathrm{DC}.$$

In this chapter, it is assumed that the data collector can contact any choice of k nodes out of which the data should be reconstructed, and that n nodes store

each one encoded fragment, each with storage capacity α. To capture the notion of storage capacity in the information flow, each node is seen as logical pair of two nodes $(x_{\text{in}}, x_{\text{out}})$, connected with an edge of capacity α:

$$\text{S} \overset{\infty}{\to} x_{\text{in}} \overset{\alpha}{\to} x_{\text{out}} \overset{\infty}{\to} \text{DC}.$$

In the event of one failure, the data lost must be replenished, and a newcomer downloads β symbols from d nodes each, in order to replace the missing data. The total amount of bandwidth used for this node is thus $\gamma = d\beta$. The information flow includes repair by considering an information flow from a live node to the newcomer $(x'_{\text{in}}, x'_{\text{out}})$:

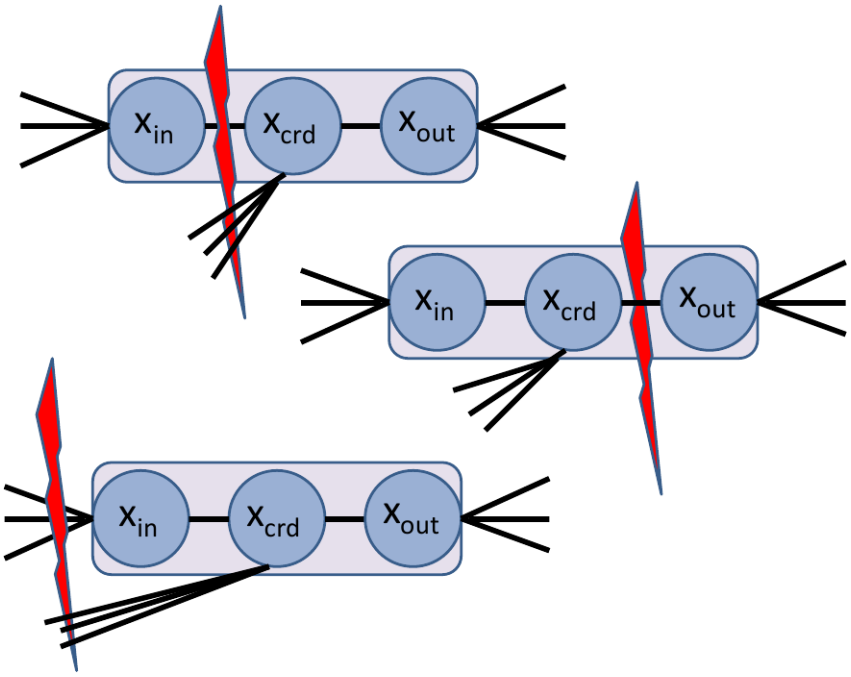
$$\text{S} \overset{\infty}{\to} x_{\text{in}} \overset{\alpha}{\to} x_{\text{out}} \overset{\beta}{\to} x'_{\text{in}} \overset{\alpha}{\to} x'_{\text{out}} \overset{\infty}{\to} \text{DC}.$$

Since it is too restrictive to assume that only one failure occurs (it is needed to have the capability to repair several faults simultanously), in case several faults occur, say t of them, a group of new comers is assumed to participate to the repair. Both [6, 11] independently proposed the idea that the new nodes joining the network could collaborate, and exchange information among each others. To model this scenario, each node is now seen as a triple instead of a pair (following [6]), that is each node corresponds to $(x_{\text{in}}, x_{\text{out}}, x_{\text{coord}})$. The link between x_{in} and x_{coord} is supposed to be infinite, since it represents the amount of data that the node temporarily stores (in all cases, it does not make any sense to have this link with capacity less than the storage capacity α, which is all that is needed for the min-cut argument below). The link between x_{coord} and x_{out} is as before the storage capacity α. In the event of t failures, when a new comer k connects to another new comer l, data is formally transferred between x^k_{in} and x^l_{coord}, with a bandwith of β', and each new comer has $t-1$ incoming edges (it connects to $t-1$ other new comers).

3.2 Min-cut Bounds

Now that the information flow model is established, let us see how a min-cut bound can be derived. Consider a network model as above, that is every node is a triplet, with storage capacity α, repair bandwidth β, fan-in d, and bandwidth for collaboration of β'. Suppose t failures occur.

Partition the network nodes 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. Any new node $(x_{\text{in}}, x_{\text{coord}}, x_{\text{out}})$ joining the network connects to d existing nodes, and assume that the data collector DC connects to k of these x_{out} (any other possible strategy is possible, this is one scenario which is chosen, since we only have information on the degree of the new nodes joining). Since we are interested in a min-cut between the source and the data collector, we assume that x_{out} belongs to $\bar{U}$ (and these are the only nodes in $\bar{U}$), thus U is the set of network nodes not in $\bar{U}$ to which belongs the source S.



For the first repair, a group of u_0 nodes is arriving in the network, and we consider this cut where in $\bar{U}$, we have all the u_0 x_{out} nodes. The x_{in} nodes can be either in U, say m of them, or in $\bar{U}$, for the remaining $u_0 - m$.

- For the m x_{in}, either x_{coord} is in U or it is in $\bar{U}$. If it is in U, then the cut includes the storage link, and α contributes to the cut. Now if x_{coord} is instead in $\bar{U}$, the cut is more than α, since the link between x_{in} and x_{coord} is already at least α. Since we do not know how many x_{coord} are on each side, a lower bound is to say that they are all in U, for a total cut of $m\alpha$.
- We now look at the $u_0 - m$ x_{in} in $\bar{U}$. Since this is the first group joining, those x_{in} in $\bar{U}$ need to connect to existing nodes in U, for a cut of $d\beta$. Similarly, every x_{coord} needs to connect to t x_{in} newcomers, but since $u_0 - m$ are already in $\bar{U}$, the cut will contain only $t - u_0 + m$ of the t edges.

So the total contribution of this first group of new comers 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

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

The process is the same with the second group joining. All the x_{out} are in $\bar{U}$, the m x_{in} in U, irrespective of where the corresponding x_{coord} is, contribute of at least $m\alpha$, and the $u_1 - m$ x_{in} in $\bar{U}$ contribute of $(u_1 - m)(d - u_0)\beta$, since they could connect to the u_0 nodes from the first group already in $\bar{U}$. Their corresponding x_{coord} 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 new coming nodes gives the min-cut bound.

Proposition 3.1. *A min-cut bound between the source and a data collector is*

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

where g is the number of different groups, u_i is the size of each group, and $k = \sum_{i=0}^{g-1} u_i$ 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

$$\sum_{i=0}^{g-1} u_i \min(\alpha, [d - \sum_{j=0}^{i-1} u_j]\beta + (t - u_i)\beta') \geq B. \tag{3.1}$$

Note that the authors of [4] who proposed *mutually cooperative recovery* also computed a min-cut bound in the context of collaboration among new comers, but they consider only a particular case of the above, when (1) $\beta = \beta'$, (2) the new nodes automatically contact all the live nodes. It is interesting for a better understanding to consider the case of $t = 1$ failure, see the exercises.

Two extreme cases of (3.1) can be identified. The highest contribution in β comes when there is no contribution from β', that is, $u_i = t$ for all i, in which case $g = k/t$, yielding

$$\sum_{i=0}^{k/t-1} t \min((d - it)\beta, \alpha) \geq B. \tag{3.2}$$

Vice-versa, the highest contribution in β' is required when that of β is reduced, that is $u_i = 1$ for all i, and $g = k$:

$$\sum_{i=0}^{k-1} \min((d - i)\beta + (t - 1)\beta', \alpha) \geq B. \tag{3.3}$$

Definition 3.1. We call any coding strategy that allows to obtain

$$\begin{aligned} \sum_{i=0}^{k/t-1} t\min((d-it)\beta,\alpha) &= B \\ \sum_{i=0}^{k-1} \min((d-i)\beta+(t-1)\beta',\alpha) &= B, \end{aligned}$$

a collaborative regenerating code ([6] use *coordinated regenerating codes* while [11] chose *cooperative regenerating codes*).

The goal is now, given n, k, d, t, to find the optimal trade-off between the storage cost α and the repair cost

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

subject to the min-cut constraints. This is not an easy optimization problem to solve in closed form expression. We thus discuss particular (though important) cases, that of the boundary points [6].

The minimum storage point: the minimum storage per node is $\alpha = B/k$, since the data can be retrieved from any choice of k nodes. From (3.2), we have a sum of k/t terms, each of size at most tB/k. Since the total must be at least B, each term in the sum must be tB/k. Thus for every i

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

and

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

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

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

The same computation with (3.3) gives

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

Thus at the minimum storage point, we have

$$\boxed{\alpha = \frac{B}{k},\ \beta = \beta' = \frac{B}{k}\frac{1}{d-k-t}.} \tag{3.4}$$

The minimum bandwidth point: to minimze γ alone, let α tend to infinity. Then (3.2) gives

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

that is

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

From (3.3) we similarly compute that

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

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

$$\boxed{\alpha = \frac{B}{k}\frac{2d+t-1}{2d-k+t},\ \beta = \frac{B}{k}\frac{2}{2d-k+t},\ \frac{B}{k}\frac{1}{2d-k+t}.} \tag{3.5}$$

3.3 Regenerating Codes

In the previous section, collaborative regenerating codes were defined as codes satisfying the optimal trade-off between storage overhead α and repair bandwidth $\gamma = d\beta + (t-1)\beta'$. In the case where $t = 1$, we drop the term "collaborative" and call them regenerating codes, as they were firstly introduced [2].

The above analysis is an information theoretic type of analysis, which provides bounds on the best that can be achieved. However, it does not provide codes.

We will present three constructions of regenerating codes, two of them are actually for the case of one failure $t = 1$, one at the minimum bandwidth point, one at the minimum storage point. The third construction will be a collaborative regenerating code at the minimum storage point.

Construction I: one failure at the minimum storage point.

Consider the minimum storage regeneration (MSR) point for $t = 1$ failure, which from (3.4) is given by

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

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

$$B = k(d-k+1)$$

and thus

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

By fixing $d = k + 1$, we further have

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

We present a construction from [10]. Partition the object $\mathbf{o} = (o_1, \ldots, o_B)$ into two vectors $\mathbf{o}_1 = (o_1, \ldots, o_k)$ and $\mathbf{o}_2 = (o_{k+1}, \ldots, o_B)$ (recall that $B = 2k$ 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 MDS code:

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

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

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

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

To retrieve the object from k nodes, a data collector downloads the 2 symbols from k 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 k indices. Because G' is the generator matrix of an (n, k) 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.

Suppose one node, say the jth node fails. A newcomer downloads $\beta = 1$ symbol v_i from $d = k + 1$ nodes (w.l.o.g we label these d nodes from 1 to $d = k + 1$), and this downloaded symbol is of the form

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

Now the newcomer computes two new symbols out of these d w_i, by taking linear combinations 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 k linear equations (the p_i have length k) in $d = k+1$ unknowns (either δ_i or ρ_i), thus can be solved, with in particular the δ_i 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 k linear equations in $k+1$ 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 above).

Example 3.1. Take $n = 5$ nodes, $k = 3$ and $d = k+1 = 4$, so that $B = 2k = 6$. We denote the object $\mathbf{o} = (o_1, \ldots, o_6)$ 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. 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 $w^3 = w+1$, as explained at the end of the first chapter. 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

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

Let us make some remarks about this code.

1. In the repair process, we recreate two symbols, where one of the two (namely $\mathbf{o}_1 {v'_j}^T + \mathbf{o}_2 p_j^T$) is not exactly the one that was lost. When the recreated fragment is "bit by bit" identical, the repair is called *exact*, otherwise it is called *functional*.

2. In the original description of usage of codes for storage, we mentioned that for an (n, k) code, the source S actually computes a codeword $\mathbf{x} = (x_1, \ldots, x_n)$ from an object of size B which was cut into k pieces of size B/k each. In the construction above, we could actually permute the coefficients of the object $\mathbf{o} = (o_1, \ldots, o_B)$ as $(o_1, o_{k+1}, o_2, o_{k+2}, \ldots)$, and group them by 2, so that (o_1, o_{k+1}) is the first symbol, and (o_i, o_{k+i}) is the ith one, $i = 1, \ldots k$. By doing so, we can see the above code as an (n, k) code over symbols of length 2. By definition, it is required that the data collector DC can retrieve the data from any choice of k nodes. This must be true even if no failure occurs. Thus this forces a regenerating code to actually contain an MDS erasure code. In order words, from this point of view, regenerating codes can be seen as a combination of an MDS erasure code and a network code. In this construction as well, the equivalent (n, k) code defined over symbols of length 2 must be MDS. However, note that the original $(2n, 2k)$ does not have to be MDS, though it is often (in more general cases) convenient to choose it that way.

3. Exactly because of the underlying MDS code, the fan-in d must satisfy that $d \geq k$. Indeed, if d were smaller than k, then it means that some of the x_i should be 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.

Construction II: one failure at the minimum bandwidth point. Consider the minimum repair bandwidth (MRB) point for $t = 1$ failure, which from (3.5) is given by

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

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

$$B = kd - \frac{k^2 - k}{2}$$

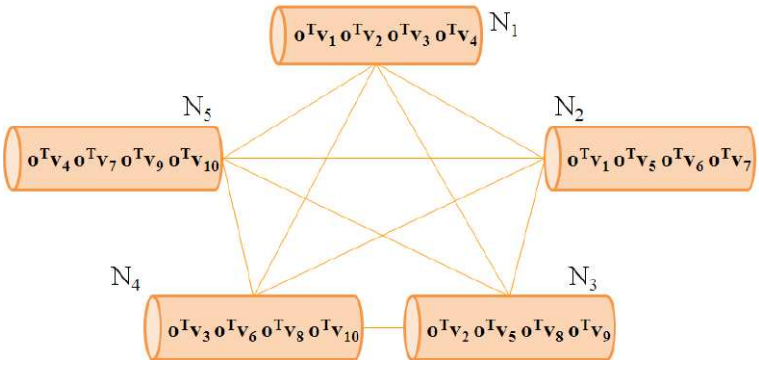


Figure 3.1: A regenerating code at MBR.

and thus

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

Now the fastest repair is obtained when $d = n - 1$, in which case, the choice of $\beta = 1$, $d = n - 1$ and k, the number of nodes from which a data collector can recover the object, determine the size B of the object:

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

In case the object does not fit this size, zero padding as well as cutting the original object can be done.

Coding with the above parameters can be done as follows [10]. Since we have assumed that the object is of the right size B, we can write it as

$$\mathbf{o} = (o_1, \ldots, o_B).$$

We are now in the presence of n nodes, and since the storage capacity α of every node is $\alpha = d = n - 1$, to each node is given $n - 1$ encoded pieces, computed as follows. 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.

Let us see what conditions the vectors v_j, $j = 1, \ldots, n(n-1)/2$, must satisfy.

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

2. The regeneration happens by construction, since if the ith node gets offline, a newcomer can get the content of the ith node by downloading one

fragment from each of the $n-1$ remaining nodes, since by construction, whenever a vertex connects two nodes, they share a common encoded fragment of the form $\mathbf{o}v_j^T$.

Example 3.2. Take $n = 5, k = 3$, 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

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

and we can choose for the generator matrix $G = [v_1, \ldots, v_{10}]$ the single parity check code $(10, 9)$, that is

$$G = \begin{bmatrix} \mathbf{I}_{10} & \mathbf{1} \end{bmatrix}.$$

By contacting $k = 3$ nodes, the data collector gets $k(n-1) = 3 \cdot 4 = 12$ encoded pieces which yield a system of 12 equations, out of which 3 are redundant, thus in fact 9 equations, 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 enough to indeed pick the $(B, n(n-1)/2)$ code to be MDS, which will give what we want.

Construction III: 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 $q \geq n$ a power of a prime. Suppose that the object $\mathbf{o}$ to be stored in n nodes can be written as $\mathbf{o} = (o_{11}, \ldots, o_{1k}, \ldots, o_{t1}, \ldots, o_{tk})$ with o_{ij} in either $\mathbb{F}_q$ or any bigger field containing $\mathbb{F}_q$. Note that this means that the object is cut into a number of pieces which depends on the number t of (predetermined, expected) failures, with $k < n - t$. The construction presented here comes from [11], which considers only the regime $k = d$.

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

$$\mathbf{O} = \begin{bmatrix} o_{11} & \ldots & o_{1k} \\ \\ o_{t1} & \ldots, & o_{tk} \end{bmatrix}.$$

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

$$\mathbf{O}\mathbf{g}_1.$$

The t rows represent t different pieces, forming the encoded data, that the corresponding node stores.

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.

Let us now assume that t nodes go offline, and t new nodes join. Let us call the t new nodes as nodes 1 to t. The ith newcomer will ask $(o_{i1}, \ldots, o_{ik})\mathbf{g}_j$ for any choice of k nodes among the live nodes.

Each newcomer can invert the matrix formed by the columns of G, and each decode $(o_{i1}, \ldots, o_{ik})$ respectively. Thus it can compute the piece corresponding to its ith row, and also can compute $(o_{i1}, \ldots, o_{ik})\mathbf{g}_j$ and send it to the jth node, which all will do similar computations and likewise deliver the missing pieces to the other newcomers, hence completing the collaborative regeneration process.

Example 3.3. Consider the $(n, k) = (7, 3)$ 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 $\mathbf{o}^T = (o_{11}, o_{12}, o_{13}, o_{21}, o_{22}, o_{23})$ with o_{ij} in either $\mathbb{F}_8$ or any finite field extension of $\mathbb{F}_8$, say $\mathbb{F}_q$. 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

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

Thus each encoded data block comprises of two pieces of size one unit each in this example, and the original object is of size six units, and each encoded block is of size two units.

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.

Let us now assume that 2 nodes go offline, and 2 new nodes join. Let us call the two new nodes as node 1 and node 2. The first newcomer will ask $(o_{11}, o_{12}, o_{13})\mathbf{g}_i$ for any choice of 3 nodes among the 5 live nodes, while the second newcomer will similarly ask $(o_{21}, o_{22}, o_{23})\mathbf{g}_i$ from any of the 5 live nodes.

Both newcomers can invert the matrix formed by the columns of G, and decode each respectively (o_{11}, o_{12}, o_{13}) and (o_{21}, o_{22}, o_{23}). Now the first node can compute the piece corresponding to its own first row, and also can compute $(o_{11}, o_{12}, o_{13})\mathbf{g}_2$ and send it to the second node, which likewise can compute $(o_{21}, o_{22}, o_{23})\mathbf{g}_1$ and send it to node 1, which completes the regeneration process.

3.4 Exercises

Project 2. "Regenerating codes in my data center".

1. Consider the min-cut bound 3.1 in the case of $t = 1$ failure. Rewrite the statement of the min-cut in this case, and rederive the proof of the min-cut bound in this simplified setting. The case of $t = 1$ failure was in fact treated in [2], before collaborative regenerating codes were introduced.

2. Pick an example of regenerating codes (not presented in these lecture notes) and detail it. Though these codes were optimized for one failure, discuss how your chosen code tolerate multiple failures.

This question is a bit more open than that on Reed-Solomon codes, since papers on regenerating codes might not explicitly discuss this issue.

Chapter 4

Hierarchical and Pyramid Codes

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

Other families of codes have been proposed, which instead combine two erasure codes, as for example hierarchical codes [3] and pyramid codes [5].

4.1 Hierarchical Codes

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 of this code, namely

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

We now combine these two codes 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, we see that the 3rd coefficient corresponds to the parity bit obtained from the the first code, the 6th coefficient to that of the second code, and the last coefficient to a parity bit computed from the all 4 information symbols.

This example can now be generalized in different ways:

1. Instead of combining the two first codes with the same parity code, another code could chosen.

2. Instead of combining two codes, one could combine an arbitrary number of codes.

3. Finally, the first codes could be something else than the parity code, and different codes could be used.

Thus more generally, let $\mathcal{C}_1, \ldots, \mathcal{C}_L$ be L different (n_i, k_i) codes, $i = 1, \ldots, L$. For each of them, we get

$$[u_{i,1}, \ldots, u_{i,k_i}] \mapsto [x_{i,1}, \ldots, x_{i,n_i}],$$

out of which we can create a new code on top of them, which will take the $L \cdot k_i$ coefficients and add, on top of the redundancy that already exists, another layer of redundancy. This leads to the idea of "local" versus "global" redundancy, where local refers to redundancy computed from a small subset of the information symbols.

The motivation for this work was 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).

The idea is that when few errors occur, only local redundancy is used to repair.

There is however little theory available about these codes, see [3] for simulation results.

4.2 Pyramid Codes

Let us start by explaining the motivating example given in [5]. 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 $|\mathbb{F}_q| \geq 11$):

$$[u_1, \ldots, u_8] \mapsto [x_1, \ldots, x_{11}],$$

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

$$[x_1, \ldots, x_{11}] = [u_1, \ldots, u_8, x_9, x_{10}, x_{11}].$$

Choose two symbols c_1, c_2 among x_9, x_{10}, x_{11}. Now split the information symbols into two groups $u_1, \ldots, u_4$ and $u_5, \ldots, u_8$, 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

$$[u_1, \ldots, u_4, \mathbf{0}],$$

and $c_{1,2}$ corresponding to c_2 with

$$[\mathbf{0}, u_5, \ldots, u_8].$$

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

$$[u_1, \ldots, u_8] \mapsto [u_1, \ldots, u_8, c_{1,1}, c_{1,2}, c_1, c_2].$$

We notice that hierarchical codes are quite 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. Again, the idea is that if few errors occur, local redundancy can be used to repair, thus reducing the fan-in (or number of nodes contacted to repair).

More generally, suppose that we start with an (n, k) MDS code, and split the k information symbols $u_1, \ldots, u_k$ into L groups $S_1, \ldots, S_L$, of size $|S_i| = k_i$, $i = 1, \ldots, L$, with $k = k_1 + \ldots + k_L$. 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 $c_{j,l}$, $j = 1, \ldots, m_0$, 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.

Pyramid codes are more structured, and thus the following can be shown [5] to determine their fault-tolerance.

Proposition 4.1. *A pyramid code constructed from a (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 a (n, k) MDS code encodes k information symbols into a codeword of length $m_1 + m_0 L$.

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. if $r \geq m_0$, 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. Then assume that all the local redundancy symbols have been erased. From the perspective of the original MDS code, it has 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. if $r < m_0$, then the number of erasures in the local redundancy symbols is less than the number of local redundancy symbols for one group S_l. Now 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, 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. $\square$

Chapter 5

Self-Repairing Codes

The codes proposed in the context of network coding (see Chapter 3) were aiming at reducing the repair bandwidth, and can be seen as the combination of an MDS code and a network code. Hierarchical codes (see Chapter 4) instead tried to repair the repair degree or fan-in (number of nodes needed to be contacted to repair) by using "erasure codes on top of erasure codes", similarly to the approach of pyramid codes. In this chapter, we present another family of codes, which also tries to reduce the fan-in, except that those codes are designed from scratch.

5.1 Homomorphic Self-Repairing Codes

We present here the main idea of [8]. Let $\mathbf{o}$ be an object of size M to be stored over a network of n nodes, that is $\mathbf{o} \in \mathbb{F}_{q^M}$, where q is a power of 2, and let k be a positive integer such that k divides M. We can write

$$\mathbf{o} = (\mathbf{o}_1, \ldots, \mathbf{o}_k),\ \mathbf{o}_i \in \mathbb{F}_{q^{M/k}}$$

which requires the use of a (n, k) code over $\mathbb{F}_{q^{M/k}}$, after which each $\mathbf{x}_i$ is given to a node to be stored.

It is known from Reed-Solomon codes (see Chapter 1) that linear coding can be done via polynomial evaluation. Let us recall how: take an object $\mathbf{o} = (\mathbf{o}_1, \mathbf{o}_2, \ldots, \mathbf{o}_k)$ of size M, with each $\mathbf{o}_i$ in $\mathbb{F}_{q^{M/k}}$, and create the polynomial

$$p(X) = \mathbf{o}_1 + \mathbf{o}_2 X + \ldots \mathbf{o}_k X^{k-1} \in \mathbb{F}_{q^{M/k}}[X].$$

Now evaluate $p(X)$ in n elements $\alpha_1, \ldots, \alpha_n \in \mathbb{F}_{q^{M/k}}$ (we do not evaluate the polynomial in zero, that is all α_i are nonzero), to get the codeword

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

Definition 5.1. We call *homomorphic self-repairing code*, denoted by $HSRC(n,k)$, the code obtained by evaluating the polynomial

$$p(X) = \sum_{i=0}^{k-1} p_i X^{2^i} \in \mathbb{F}_{q^{M/k}}[X] \tag{5.1}$$

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

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

where $p_i = \mathbf{o}_{i+1}$, $i = 0, \ldots, k-1$ 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

$$(a+b)^{2^i} = \sum_{j=0}^{2^i} C_j^{2^i} a^j b^{2^i - j} = a^{2^i} + b^{2^i},\ i \geq 1. \tag{5.2}$$

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 5.1. *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 (5.2), 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 (5.1) that enables self-repair.

Example 5.1. Recall that $\mathbb{F}_8$ denotes the finite field with 8 elements. Consider the polynomial

$$p(X) = p_0 X + p_1 X^2 + p_2 X^4 \in \mathbb{F}_8[X].$$

We have

$$p(w+1) = p_0(w+1) + p_1(w^2+1) + p_2(w^4+1) = p(w) + p(1).$$

Since $\mathbb{F}_{q^{M/k}}$ contains a $\mathbb{F}_q$-basis $B = \{b_1, \ldots, b_{M/k}\}$ (recall that a finite field built from $\mathbb{F}_2$ can be seen as a set of polynomials with binary coefficients - see the first chapter for more details - and more generally, a polynomial of degree M/k with coefficients in $\mathbb{F}_q$ can be seen as a vector of M/k coefficients in $\mathbb{F}_q$), the α_i, $i = 1, \ldots, n$, can be expressed as $\mathbb{F}_q$-linear combinations of the basis elements, and we have that

$$\alpha_i = \sum_{j=1}^{M/k} \alpha_{ij} b_j,\ \alpha_{ij} \in \mathbb{F}_q \Rightarrow p(\alpha_i) = \sum_{j=1}^{M/k} \alpha_{ij} p(b_j).$$

In words, that means that if p is evaluated in the elements of the basis B (or any other basis), then any encoded fragment $p(\alpha_i)$ can be obtained as an $\mathbb{F}_q$-linear combination of other encoded fragments.

That decoding is possible is guaranteed by either Lagrange interpolation, or by considering a system of linear equations, assuming that

$$k \leq M/k, \tag{5.3}$$

as detailed below.

Lagrange interpolation. Given k fragments $p(\alpha_{i_1}), \ldots, p(\alpha_{i_k})$ such that $\alpha_{i_1}, \ldots, \alpha_{i_k}$ are $\mathbb{F}_q$-linearly independent, the node that wants to reconstruct the file computes $q^k - 1$ linear combinations of the k fragments, which gives, thanks to the homomorphic property, $q^k - 1$ points in which p is evaluated. Lagrange interpolation guarantees that it is enough to have $q^{k-1} + 1$ points (which we have, since $q^k - 1 \geq q^{k-1} + 1$ for $k \geq 2$) to reconstruct uniquely the polynomial p and thus the object. This requires

$$q^{k-1} + 1 \leq q^{M/k} - 1,$$

namely there must be enough points in which to evaluate the polynomial, which holds subject to (5.3):

$$q^k \leq q^{M/k} \Rightarrow q^{k-1} + 1 \leq q^k - 1 \leq q^{M/k} - 1.$$

Solving a system of linear equations. Alternatively, one can consider decoding as solving a system of linear equations. Given k $\mathbb{F}_q$-linearly independent fragments, say $p(\alpha_{i_1}), \ldots, p(\alpha_{i_k})$, we can write

$$\begin{pmatrix} \alpha_{i_1} & \alpha_{i_1}^q & \alpha_{i_1}^{q^2} & \ldots & \alpha_{i_1}^{q^{k-1}} \\ \alpha_{i_2} & \alpha_{i_2}^q & \alpha_{i_2}^{q^2} & \ldots & \alpha_{i_2}^{q^{k-1}} \\ \vdots & & & \vdots & \\ \alpha_{i_k} & \alpha_{i_k}^q & \alpha_{i_k}^{q^2} & \ldots & \alpha_{i_k}^{q^{k-1}} \end{pmatrix} \begin{pmatrix} p_0 \\ p_1 \\ \vdots \\ p_{k-1} \end{pmatrix} = \begin{pmatrix} p(\alpha_{i_1}) \\ p(\alpha_{i_2}) \\ \vdots \\ p(\alpha_{i_k}) \end{pmatrix},$$

and the problem of recovering the object reduces to solving the above system of linear equations. Note that since $\mathbb{F}_{q^{M/k}}$ is a vector space of dimension M/k over $\mathbb{F}_q$, condition (5.3) is needed to guarantee that there exist k $\mathbb{F}_q$-linearly independent fragments.

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 5.1: Ways of reconstructing missing fragment(s) in Example 5.2

Example 5.2. Take a data file $\mathbf{o} = (o_1, \ldots, o_{12})$ of $M = 12$ bits ($q = 2$), and choose $k = 3$ fragments. We have that $M/k = 4$, which satisfies (5.3), that is $k = 3 \leq M/k = 4$.

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 self-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 $p(w^5)$ goes offline. A new comer can get $p(w^5)$ by asking for $p(w^2)$ and $p(w)$, since

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

Table 5.1 shows other examples of missing fragments and which pairs can reconstruct them, depending on if 1, 2, or 3 fragments are missing at the same time.

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, if 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 (namely, what is the probability that the object can be recovered given a probability distribution of node failures), which can be found in [8].

5.2 Projective Self-Repairing Codes

Another construction of self-repairing codes was proposed in [9], from which we present some examples below.

A construction for 5 nodes. 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, $i = 1, \ldots, 5$ 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). For data reconstruction: by contacting $k = 2$ nodes, the data collector gets 4 linear equations to decode an object of size $B = 4$.

A construction for 21 nodes.

We propose another code with different parameters, namely $n = 21$, $B = 6$, $k = 3$ and $\alpha = 2$.

N_1 to N_7	N_8 to N_{14}	N_{15} to N_{21}
(100000), (110111)	(011000), (001110)	(001010), (110100)
(010000), (101011)	(001100), (000111)	(000101), (011010)
(001000), (100101)	(000110), (110011)	(110010), (001101)
(000100), (100010)	(000011), (101001)	(011001), (110110)
(000010), (010001)	(110001), (100100)	(111100), (011011)
(000001), (111000)	(101000), (010010)	(011110), (111101)
(110000), (011100)	(010100), (001001)	(001111), (101110)

As an example of repair, suppose that the node N_1 fails. It can be repaired by contacting the following three pairs all involving N_4: (N_4, N_{12}), (N_4, N_{10}), and (N_4, N_5). Data reconstruction is as for the first example: contact $k = 3$ nodes, and get 2 symbols from each, that is 6 linear equations, to decode an object of size 6.

Code properties. We summarize some of the main properties of the above examples:

- *Non-MDS:* as the previous instance of self-repairing codes, these codes are usually not MDS (see [9] for tore details on the fault-tolerance of these codes).
- *Systematic Like Code:* It is usually appreciated from an implementation perspective to use a systematic code, since it makes the object retrieval immediate. We notice that though our code is not systematic, we 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.
- *Bandwidth cost for regeneration:* Unlike HSRC, these codes are not atomic, and instead comprise of α pieces. Thus, similar to regenerating codes, one could also expect to regenerate an encoded block piece-by-piece, by contacting more (larger d) number of nodes. For example, when using the code for $n = 21$ nodes, if the data for node N_1 needs to be regenerated, one could do so by contacting two nodes and downloading four pieces (units) of data, as we have already seen. One could instead also contact $d = 3$ nodes, and regenerate the two lost pieces by downloading only three units of data. For instance, by downloading (010000) from N_2, (110000) from N_7 and (000111) from N_9.

Where are these codes coming from? They are called "projective" because they come from a classical projective geometry construction which provides a partition of the space, as illustrated below with an example.

Example 5.3. Consider the finite fields $\mathbb{F}_2$, $\mathbb{F}_4$ and $\mathbb{F}_{16}$, which are included into each others as follows:

$$\begin{array}{c} \mathbb{F}_{16} \\ | 2 \\ \mathbb{F}_4 \\ | 2 \\ \mathbb{F}_2 \end{array}$$

We will show how to obtain a partition of $\mathbb{F}_{16}$. Write

$$\mathbb{F}_4^* = \{1, \nu, \nu^2\}$$

where $\nu^2 = \nu + 1$ and the notation $*$ means that 0 is removed,, and

$$F_{16}^* = \{\omega^i\}_{i=1}^{15}$$

where $\omega^4 = \omega + 1$. Since $\nu = \omega^5$, we can rewrite

$$\mathbb{F}_4^* = \{1, \omega^5, \omega^{10}\},$$

and

$$\mathbb{F}_{16}^* = \{\omega^i\}_{i=1}^{15} = \cup\{\omega^i, \omega^{5+i}, \omega^{10+i}\}_{i=1}^{5} = \bigcup_{i=1}^{5} \omega^i \mathbb{F}_4^*.$$

We thus have a partition (a disjoint union) of $\mathbb{F}_{16}^*$ into cosets of the form $\omega^i \mathbb{F}_4^*$, $i = 1, \ldots, 5$. More precisely, $\mathbb{F}_{16}$ can be decomposed into direct sums of $\mathbb{F}_2$:

$$\mathbb{F}_{16} = \mathbb{F}_4 \oplus \nu\mathbb{F}_4 = \mathbb{F}_2 \oplus \nu\mathbb{F}_2 \oplus \omega\mathbb{F}_2 \oplus \omega\nu\mathbb{F}_2,$$

so that each element of $\mathbb{F}_{16}$ can be written as a 4-tuple. For example, the coset $\omega\mathbb{F}_4^*$ contains the elements $\omega, \omega\nu, \omega\nu^2$. As $\nu^2 = \nu + 1$, $\omega\nu^2$ is the sum of the two other points. Thus writing $\omega = (0, 0, 1, 0)$ and $\omega\nu = (0, 0, 0, 1)$, we finally get that the plane defined by the coset $\omega\mathbb{F}_4^*$ is $\{(0010), (0001), (0011)\}$.

The above yields a partition of the space as follows:

$$\begin{aligned}
\mathbb{F}_4^* &= \{(1000), (0110), (1110)\} \\
\nu\mathbb{F}_4^* &= \{(0100), (0011), (0111)\} \\
\nu^2\mathbb{F}_4^* &= \{(0010), (1101), (1111)\} \\
\nu^3\mathbb{F}_4^* &= \{(0001), (1010), (1011)\} \\
\nu^4\mathbb{F}_4^* &= \{(1100), (0101), (1001)\}
\end{aligned}$$

Chapter 6

Exercises

Project 3. "Hierarchical codes in my data center".

1. Construct one concrete example of hierarchical code (by picking an erasure code of your choice).
2. By mimicking the construction of pyramid codes, try to prove the fault-tolerance of your proposed code.
3. Discuss the repair property of this code as well.

This question is really open!

Project 4. "Self-repairing codes in my data center".

1. Self-repairing codes allow to correct multiple failures, very often in parallel. Discuss a schedule algorithm of such repairs.

This question is even more open!!

Bibliography

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

[2] 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.

[3] 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.*

[4] 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.

[5] 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.*

[6] 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).

[7] 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.

[8] F. Oggier, A. Datta, "Self-repairing Homomorphic Codes for Distributed Storage Systems", INFOCOM 2011. Extended version at `http://arxiv.org/abs/1107.3129`

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

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

[11] K. W. Shum, "Cooperative Regenerating Codes for Distributed Storage Systems", to appear at *ICC 2011*, available at arXiv:1101.5257v1.

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