Erasure Coding in Distributed Storage Systems

Kevin Wellenzohn

February 10, 2015

Abstract

Data centers are nowadays highly distributed, possibly over continents, to cope with the huge amounts of data collected these days. To deal with data loss in an environment where hardware failures are very common, researchers have developed new kinds of *erasure codes*. Erasure coding is a common technique used in all sorts of digital communication, including deep space communication and QR-codes, but has only seen now application in distributed storage systems. Facebook and Microsoft use a new class of erasure codes to build more storage efficient and reliable data centers. In this paper we introduce erasure coding and show how such codes are applied in a distributed storage system.

1 Introduction

Due to the ever increasing amount of data generated in the internet, companies like Google, Facebook, Microsoft and Amazon have to deal with huge amounts of data spread over many data centers on multiple continents. These data centers house thousands of servers (often simple commodity machines), which suffer frequently hardware failures and therefore the data stored on those machines is potentially at risk of being completely lost. The standard approach to keep data safe is the so called *three replication*, where three copies of the whole data is stored on different servers, possibly in different data centers. This technique has, however, a tremendous space overhead of 200%, which makes data centers much more expensive.

Lately, researchers at Facebook, Microsoft and others designed *erasure codes* for the use in distributed storage systems, considerably reducing the necessary space overhead to around 60%. Erasure coding is a technology that only few people ever notice and yet forms the backbone of modern digital communication. Whenever data needs to be digitally transmitted over some lossy channel, which is the rule rather than the exception, some form of erasure coding is applied. Erasure codes allow to detect and if possible to recover to some extent data that was corrupted during the transmission over a so called *erasure channel*. They are widely used in deep space communication by the NASA, satellite transmission in general, digital television, broadband modems, wireless communication and storage systems such as the Compact Disc (CD) or RAID Level 6.

The aim of this paper is to show how erasure coding is applied to protect data from loss in a distributed storage system. We will first introduce erasure codes on a conceptual level and briefly survey the quite large zoo of erasure codes that have been proposed until today. In a next step we fill focus on one particular and very famous erasure code, the Reed-Solomon (RS) code. This sets the groundwork for understanding how companies like Facebook and Microsoft have leveraged erasure codes to build more storage efficient and reliable data centers.

2 Erasure Coding

Whenever data is transmitted there is the risk of losing parts of the information due to the imperfect underlying transportation mode. In the internet we cope with this problem using specialized protocols such as TCP/IP that rely on a back-channel to request missing or corrupted data packets. However, in many occasions there is no such back-channel (e.g. in satellite communication) and we have to use some form of *forward error correction* (FEC). Using FEC data is encoded in such a way that the receiver can successfully decode the message with *high* probability even in the presence of *erasures* (packet losses) and data corruption (e.g. flipped bits).

Erasure Coding is one form of forward error correction where data is sent over a so called erasure channel (see Section 2.1 for more details). On the sender side, the message of length k is *encoded* into a code word of length n, where $n > k$, which essentially adds $n - k$ parity symbols. Once the message is encoded the code word is sent over the erasure channel and parts of the code word reaches the recipient. The receiver can *decode* the possibly incomplete code word and recover the original message if "enough" data is available. This is only a very superficial description that lacks many necessary details, which we will, however, gradually introduce in this and the following section.

Erasure codes are used in numerous applications and this is just an incomplete list of such. The NASA and ESA put forward a recommendation of using Reed-Solomon (RS) codes (described in Section 3) for the data transmission in all of their deep space missions [8]. Furthermore, erasure codes are also used in satellite communication; in fact, if you watch digital television, chances are good that the reliable data transmission is achieved through erasure coding. Erasure codes particularly excel in broadcast environments where no back-channel is available or very expensive. Even the Compact Disc (CD) uses RS codes to protect the data from scratches on the surface and the nowadays very popular QR-codes use similar codes to make sure that the content can be accessed, even if the QR-code is damaged. Closer to the application of erasure codes in distributed storage systems is the next use case. RAID Level 6 uses RS-codes to tolerate any two disk failures.

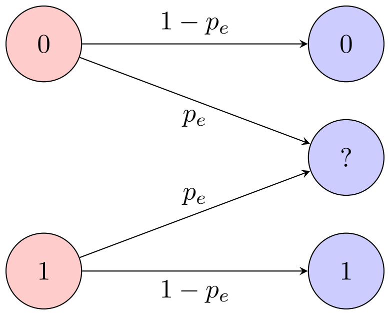


Figure 1: Binary Erasure Channel

2.1 Erasure Channel

In the previous high-level description we briefly talked about the erasure channel but left out the details. It is, however, important to clearly define the properties of the transportation channel to design efficient erasure codes. An erasure channel is a channel where information that is sent across is either received with probability $1 - p_e$ without any corruption at all, or it is lost ("erased") altogether with probability p_e, as visualized by figure 1. Note that if a bit is lost the receiver knows that.

This is a limiting assumption which simplifies the algorithm design, but it does not reflect reality, where during transmission bits can also be flipped and it is often not clear which bits were lost. However, we can attack the first problem by adding checksums to each packet which is sent across the channel and if the checksum computation does not match on the receiver end we can treat the package as if it was lost during transmission. The second problem can be addressed by adding a sequence number to each packet, which gives the receiver the possibility to detect complete packet losses. Some codes have the capability of detecting errors (i.e. flipped bits) by themselves, one such code is the RS-code discussed in Section 3.

2.2 Zoo of Erasure Codes

The history of erasure codes and error correcting codes in general is a relatively long one for computer science terms, as it dates back to the 1950s when Richard Hamming introduced the Hamming codes for the error-prone punched card reader. Since then a plethora of codes were developed of which we will present here only a few noteworthy.

The most popular erasure code to date is the RS-code that we mentioned now already a few times and will discuss in more detail in Section 3. The code was introduced by Irving Reed and Gustave Solomon in 1960 in their article "Polynomial Codes over Certain Finite Fields" [5] and has shaped the field of Coding Theory for the years to come.

Roughly forty years after RS codes were presented a new class of erasure codes, the so called *fountain codes*, were introduced by Michael Luby et al. in [2]. Fountain codes are a special kind of erasure codes as they are *rateless*. One can imagine a fountain spitting out infinitely many data packets and the receiver can hold a bucket to capture a few packets of the possibly never ending stream. If the receiver manages to collect a tiny fraction more packets than the original message is composed of, the receiver can decode the message. The possible applications are manifold, for instance imagine a satellite that broadcasts a large file to thousands of people. If the satellite just sent the raw file packet by packet and a particular receiver fails to obtain a single packet, the whole transmission is unsuccessful. However, if the satellite used fountain codes, it would keep sending packets encoded in a particular scheme and the receiver could decode the file even though he missed a few packets during the transmission. Tornado codes presented in [2] are the first instance of fountain codes. They use a layered approach to encode message blocks, where all but the last layer use so called *Low-Density Parity-Check* (LDPC) codes and the last uses Reed Solomon codes. The Tornado codes are by orders of magnitudes faster to encode and decode than RS codes, especially as the file size grows. The Luby Transform (LT) codes [4] and Raptor (**Rap**id **Tor**nado) [7] codes are the result of further developments of the Tornado codes. RaptorQ, the latest version of Raptor codes, is used as erasure code by the Digital Video Broadcasting (DVB) consortium for television broadcasting.

3 Reed Solomon Codes

So far we mentioned RS codes and how important they are in modern communication a couple of times and in this section we will describe in detail how they work and what properties they have. Note that RS codes were invented by two mathematicians and so it comes as no real surprise that RS codes have a rather mathematical structure and rely on some more advanced mathematical constructs of number theory. This section is, however, intended for non-mathematicians and gives a gentle introduction to RS codes. If you are interested in a more formal and complete description of RS codes we refer to [1].

RS codes are parameterized by the two parameters n and k ($k < n$), where n is the total block length of which k blocks are reserved for message content and the remaining $t = n-k$ blocks are parity information. RS codes can correct up to t arbitrary block erasures, or put differently, the receiver cannot reconstruct the message if at least $t+1$ blocks are erased. The code has therefore the property that the receiver can successfully decode the message if he receives *any* k out of the n transmitted blocks. In that sense RS codes are optimal, because the minimum number of packets that have to reach the receiver is equal to number of packets the original message is composed of. Codes that have this property are called *maxiumum distance separable* (MDS) codes. Fountain codes, for instance Raptor codes, that we have seen before in Section 2.2 do not have this property. Similarly we will see in Section 4 that modern erasure codes for distributed storage systems sacrifice this property to get more efficient codes.

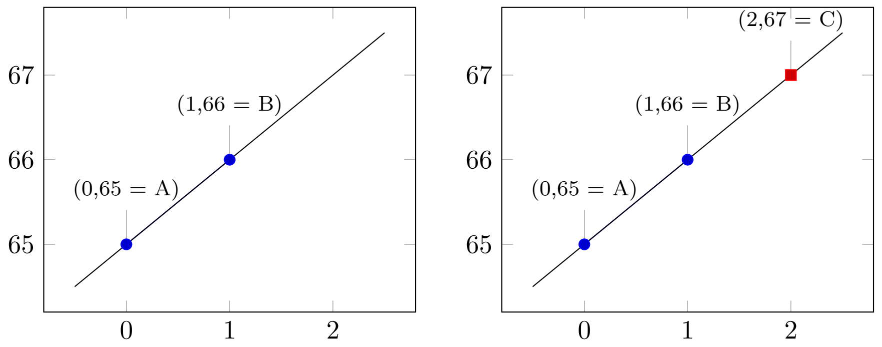
An RS code is typically denoted in the form (n, k). The most commonly used RS code, used by the NASA and many others, is $(255, 223)$, which means a message is broken into blocks of length 255 bytes, each containing 223 bytes information and $t = 255 - 223 = 32$ bytes parity. This code can correct up to 32 erased bytes out of any 255 transmitted bytes.

3.1 Intuition

Let us describe the idea behind RS codes by means of an example. Our goal is to transmit the two characters $\boxed{\text{AB}}$ and for that we decide to use a $(3, 2)$ RS code, i.e. we add 1 parity character to the encoded message. According to the ASCII table A is denoted by the number 65 and B is 66. We can imagine those two numbers as to be the two points $(0, 65)$ and $(1, 66)$ in the two dimensional space. Observe that we can draw a line, of the form $m(x) = a_0 + a_1 x$, through those two points. To exactly define the line we need to know the two coefficients a_0 and a_1, which we can find by solving the following linear system

$$\begin{aligned} a_0 + a_1 \cdot 0 &= 65 \\ a_0 + a_1 \cdot 1 &= 66 \end{aligned}$$

The coefficients turn out to be $a_0 = 65$ and $a_1 = 1$, giving us the line $m(x) = 65 + 1x$, which is visualized in figure 2a.



(a) Line passing through $(0, 65)$ and $(1, 66)$ (b) The point $(2, 67)$ forms the parity

Figure 2: Reed-Solomon example encoding of the string $\boxed{\text{AB}}$

Once we know the line $m(x)$ we can add any number of parity letters we want by evaluating $m(x)$ at more points. For instance, since we want one parity letter in our example, we evaluate $m(x)$ at $x = 2$, resulting in $m(2) = 65 + 1 \cdot 2 = 67$. Figure 2b shows that the additional point $(2, 67)$ lies, as intended, on the line $m(x)$. Looking 67 up in the ASCII table shows us that it corresponds to letter C, so the complete encoded message is $\boxed{\text{ABC}}$.

Now imagine that during the transmission of $\boxed{\text{ABC}}$ an erasure occurred and the receiver got $\boxed{\text{A?C}}$, how can we recover the missing second letter? First observe, that we actually *can* recover the letter, because we used a $(3, 2)$ RS code, which allows us to recover $t = 3 - 2 = 1$ erased letters. For the recovery the receiver has essentially to restore the function $m(x)$, and evaluate $m(x)$ at the missing position $x = 1$. This can be done the same way the sender built $m(x)$ in the first place, namely find a line that passes through the two points $(0, 65)$ and $(2, 67)$. Note that, if two erasures occurred during transmission, we could not restore $m(x)$, because one point is not sufficient to exactly identify the line $m(x)$ passing through it.

3.2 Formal Description

In this section we will describe more formally what we already introduced rather informally in the previous section. RS codes rely on the fact that a polynomial of degree j, $m(x) = a_0 + a_1x + a_2x^2 + \ldots + a_jx^j$, can be described by evaluations of that polynomial for any $j + 1$ distinct x values.

For an (n, k) RS code this means that the message polynomial $m(x)$ has degree $k - 1$ and the sender evaluates $m(x)$ at k points for the original message and at $t = n - k$ additional x values for parity information. The sender and receiver have to agree in advance at which x values the message is contained, such that the receiver knows in the decoding procedure where to look for the original message. By convention, the values of the polynomial $m(x)$ for $0 \leq x < k$ form the original message and similarly for $k \leq x < n$ the parity symbols are computed.

3.2.1 Galois Fields

So far we did not mention what happens if $m(x)$, for some x, is not a natural but a real number. For several reasons we do not want to deal with real numbers in the parity section, for example because they can be arbitrarily long and it becomes hard and imprecise to transmit them. Therefore we limit ourselves to use only numbers of a so called *Galois field*, which is a field with a finite number of elements that adheres to the rules of *modular* arithmetic. A Galois field is usually denoted by $GF(p^q)$, where p is some prime number, called the *characteristic* of the field, and q is some positive integer. The field $GF(p^q)$ contains p^q elements and supports two operations, $+$ (sometimes written also $\oplus$) and $*$, usually referred to as addition and multiplication, respectively. By means of addition and multiplication the inverse operations subtraction and division are intuitively defined. An important and particularly useful property of Galois fields is that all supported operations are *closed*, i.e. the result of an operation is again an element of the field.

A Galois field $GF(p)$, where p is a prime and the characteristic is $q = 1$, is called a prime field. Addition and multiplication in a prime field are defined in terms of modular arithmetic. For example in $GF(9)$, we have that $5 + 5 = 1$ and $5 \times 5 = 7$. However, things are more complicated for Galois $GF(p^q)$ with $q \neq 1$, and as it turns out, those are exactly the Galois fields we are interested in. Because for the most common $(255, 223)$ RS code we use $GF(2^8) = GF(256)$, which is particularly useful as we can represent one symbol as exactly one byte and similarly see each element of the field as one character in the extended ASCII table.

It turns out that since $GF(2^8)$ has characteristic $p = 2$, the addition of two numbers corresponds to the XOR of the two numbers in binary notation. Interestingly enough, the subtraction modulo 2 is again simply the XOR, which means addition and subtraction are the same operations in this field. For example, $150 \oplus 128 = 22$ and $150 \oplus 150 = 0$.

Multiplication and division in $GF(2^8)$ are somewhat more complicated and we will not delve into too much detail at this point. An element of $GF(p^q)$ can be seen as a polynomial with degree strictly less than q, where the coefficients are in the field $GF(p)$, e.g. we can see $13_{10} = 1101_2 \in GF(2^8)$ as the polynomial $x^3 + x^2 + 1$ in the field $GF(2)$, where the coefficients are either 0 or 1. Then the multiplication of two elements $a, b \in GF(p^q)$ is defined as the multiplication of the polynomials of a and b in $GF(p)$ modulo some *irreducible* polynomial of degree exactly q in $GF(p)$. There are lists of irreducible polynomial for every important values of p and q; and one candidate for $GF(2^8)$ is $p(x) = x^8+x^4+x^3+x^2+1$. Let us make an example where we want to multiply 40 by 8 in $GF(2^8)$. First, we convert the numbers to polynomials in $GF(2)$, which is $40_{10} = 101000_2 \Rightarrow x^5 + x^3$ and $8_{10} = 1000_2 \Rightarrow x^3$. Next we multiply the polynomials, $(x^5 + x^3) \times x^3 = x^8 + x^6$ (Remember to always perform addition in $GF(2)$, so e.g. $x^5+x^5 = 0$). Now we have to compute $(x^8+x^6) \mod (x^8+x^4+x^3+x^2+1)$, which turns out to be $x^6 + x^4 + x^3 + x^2 + 1$. Converting the result back to a number in $GF(2^8)$ gives us the result of the multiplication $1011101_2 = 93_{10}$. This is definitely not trivial and this short description was not aimed to give a thorough discussion of the topic, but we hope this is a good starting point for the reader to conduct further investigations on this topic.

3.2.2 Encoding and Decoding

In our example in section 3.1 we discussed so far one way of encoding a message using RS codes, however in the relatively long history of RS codes a number of encoding and decoding schemes haven been developed. In fact, Reed and Solomon did not provide any efficient decoding algorithm by themselves when they presented the RS code in [5]. Nine years later Elwyn Berlekamp and James Massey were the first to describe an efficient decoding algorithm, which is today known as the Berlekamp-Massey algorithm. Encoding and decoding RS codes can be achieved in $O(n^2)$.

Encoding and decoding algorithms also differ in the format of the output, as some algorithms generate *systematic* codes, while others do not. A code is *systematic* if the original message is part of the output, i.e. the message is preserved in the encoding procedure and only some parity information is added. A *systematic* code is definitely desirable, because it makes the decoding in the absence of erasures particularly easy, as the message can be directly processed without any further decoding steps. In our example we presented a systematic version of the code, as $\boxed{\text{AB}}$ was encoded into $\boxed{\text{ABC}}$, but the first version of

RS codes, published in [5], were not systematic. Reed and Solomon used the message symbols $m_0, m_1, \ldots, m_{k-1}$ as the coefficients of the message polynomial $m(x) = m_0 + m_1x + m_2x^2 + \ldots + m_{k-1}x^{k-1}$, which makes the encoding faster.

4 RS-Codes in Practice

So far we have already seen numerous applications of erasure codes in general and Reed Solomon codes in particular, including deep space communication and QR-codes. In this section we will focus on a relatively new use case, namely that of distributed storage systems. Big internet companies like Google, Facebook, Microsoft and Amazon do not release exact numbers of how much data they store, all we know is that it is a huge number. In a paper [3] published in 2012 Microsoft says that they would soon surpass an Exabyte of data and one can only guess how much more data Google stores. All of these companies operate a network of data centers spanning the whole planet. Hardware failures are very common in such large environments, Facebook for instance counted on average 20 node failures per day on a relatively small 3000 node Hadoop cluster [6].

Protecting against data loss in this large scale environment is therefore a challenging problem. The de-facto industry standard for building more durable storage systems is the so called three replication, where the data is replicated twice and stored on different locations. However, this has a dramatic storage overhead of 200%, which also makes data centers much more expensive. For this reason distributed storage operators have transitioned to erasure coding, which promises lower storage overhead and increased overall reliability at the same time.

There is, unsurprisingly, one caveat, which is the high network communication overhead induced by erasure coding. In short, the problem is that for every lost block, a certain number of other blocks must be fetched from other servers in the network to recover the lost block. This network traffic is called repair traffic and it turns out that in today's data centers the network is a main bottleneck. Therefore one does not want to "waste" precious network bandwidth for repair traffic, but instead use it for the actual computations carried out on the servers. In this section we show how Facebook and Microsoft independently adapted erasure coding techniques to overcome or at least dampen the effect of this problem.

4.1 Erasure Codes at Facebook

Facebook relies heavily on Apache Hadoop for running large analytical processing jobs on petabytes of information. The data stored in this 3000 node Hadoop cluster with 30 PB of logical storage is mostly read and rarely changed. The underlying file system, termed Hadoop File System (HDFS), is a distributed file system that uses by default three replication. For the above mentioned reasons Facebook developed and open-sourced HDFS RAID, a HDFS implementation relying on Reed Solomon erasure codes. HDFS RAID uses a $(14, 10)$ RS code, which reduces the storage overhead to merely 40%, while still guaranteeing the same level of reliability as three replication. In HDFS RAID a file is split up into fixed-sized *stripes*, each of which is again split into 10 equally blocks each of size 256 MB. Each stripe is RS encoded and 4 parity blocks are computed. Upon some data loss, HDFS RAID executes a MapReduce data recovery task, which fetches any 10 of the still existing blocks and recovers the lost block.

HDFS RAID has a serious problem, though, because the network bandwidth required by the repair job is significant. When a data node fails all blocks on it must be recovered, which means for any lost block, 10 other blocks must be transferred over the network. This means per one bit lost information, 10 bit data must be sent across the network. Compared to three replications this is a 10x increase in required necessary network traffic for repair. Since a typical data node at Facebook stores 15 TB data, its failure means a staggering total network traffic of 150 TB only for repair. Facebook calculated that if they deployed HDFS RAID to merely 50% of their data nodes, the internal data network would be fully saturated and the data center would essentially do nothing else than to repair itself. This is known as the *Repair Problem* and Facebook (and others) address this problem by trading increased storage overhead for decreased required network bandwidth.

4.1.1 Block Locality

Let us introduce the concept of block locality which will prove to be the key for more efficient erasure codes for large distributed storage systems.

Definition 1 *(Block Locality) A (n, k) erasure code has block locality r, when each coded block is a function of at most $r \leq k$ other coded blocks of the code.*

An erasure code with locality r has thus the property that if some block is lost, at most r other blocks are need for the repair. If r is chosen small enough, i.e. $r \ll k$, we can achieve a much faster repair for single block failures. Note that a *maximum distance separable* (MDS) code, introduced in section 3, has locality $r = k$, as a block can be seen as a function of k other blocks. Therefore it is impossible to achieve a code that has the MDS property and a good locality at the same time. However, both properties are desirable, since the MDS property means an optimal fault tolerance and a good locality enables fast repairs. So the question is how one can achieve “almost” MDS codes with good locality? We will describe such an erasure code that Facebook has come up with in the following section.

4.1.2 Locally Repairable Codes

Facebook proposed the so called *Locally Repairable Codes* (LRC), which build on top of the $(14, 10)$ RS codes, and pay the price of a slightly higher storage overhead for reduced network bandwidth in return. The key idea is simple, yet effective. Additionally to the $t = 4$ *global* parity blocks, LRC codes compute *local* parity blocks. The blocks are split into three groups, namely the first five of the ten data blocks, the remaining five data blocks and the 4 parity blocks. Each of this three groups gets one additional *local* parity block. Figure 3 visualizes this graphically. The local parities S_1 and S_2 can be computed as the XOR of the data blocks in their local group, for instance $S_1 = X_1 \oplus X_2 \oplus X_3 \oplus X_4 \oplus X_5$. For any single block failure the block can be recovered by considering only the other blocks in the local group. For example, if block X_3 were lost it could simply be recovered by computing $X_3 = S_1 \oplus X_1 \oplus X_2 \oplus X_4 \oplus X_5$. Only for more complex failure patterns, e.g. when two blocks of the same local group are lost, the blocks in other groups have to be taken into account. Compared to the standard $(14, 10)$ RS code, the LRC code increases the storage overhead from 40% to 70%, but reduces the block locality from 10 to 5. Furthermore, the parity block S_3 does not have to be physically stored on disk, but can be derived from the blocks S_1 and S_3, which, if applied, reduces the storage overhead to 60% only. Facebook implemented and open-sourced LRC codes in another HDFS extension, called HDFS-Xorbas.

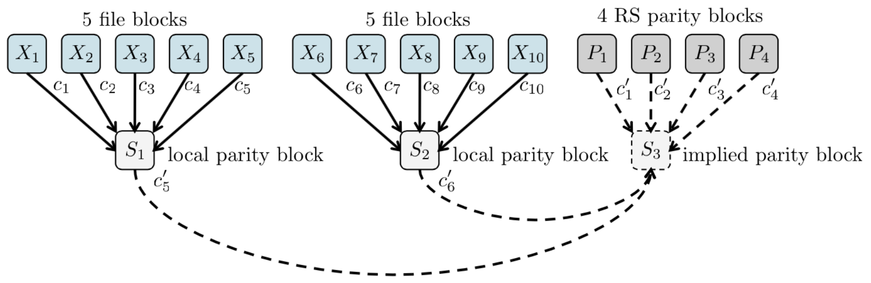


Figure 3: LRC code built on top of the RS $(14, 10)$ code

4.2 Erasure Codes at Microsoft

Microsoft operates since 2008 a large cloud infrastructure called Windows Azure that forms the back-end for many of Microsoft's products, including search, serving video, music and game content and many more. Moreover, customers can buy Windows Azure cloud storage space and access this from their Windows operated devices. This is a inherently different use case of storage space than what we previously discussed for the analytical and mostly read-only data in Facebook's Hadoop cluster, because data is frequently added, changed and deleted. Yet, also Microsoft uses erasure coding to ensure data durability [3] and for economical reasons, since smaller data centers are less expensive to build and need less power to be operated.

4.2.1 Windows Azure Storage (WAS)

Microsoft's distributed file system, called Windows Azure Storage (WAS), is an append-only file system, where data is written to the end of so called *active* *extents*. While extents are *active*, i.e. they have not reached their maximum size of, say, 1 GB, they are replicated three times. However, as soon as an extent reaches its maximum size, it is *sealed*, erasure encoded and the three replicas are deleted. Erasure encoding reduces the storage overhead considerably, but this comes at the cost of performance when dealing with node failures or *hot* storage nodes. A storage node is said to be hot if there is a sudden increase in its usage, which can result in a higher latency of the system. Three replication can deal with hot storage nodes, because if a node is hot, a load balancer can decide to use another replica of the required data fragment, whereas in erasure encoded systems this is not possible. However, one optimization that WAS applies to meet this problem is to treat the hot node as if it were offline and in parallel reconstruct the required data fragment, which is on the hot node, from the remaining data fragments stored on "cool" data nodes. WAS then serves whichever fragment is first available, either the fragment read from the slow hot node, or the dynamically reconstructed fragment.

4.2.2 Local Reconstruction Codes

Microsoft's WAS does not use RS codes, or any other MDS codes in general, for erasure coding because of the *Repair Problem*, that also Facebook described and which we already introduced in sec 4.1. Interestingly enough, both companies independently developed a similar approach to the repair problem, namely that of block locality. While Facebook calls its erasure code **Locally Repairable Codes**, Microsoft termed them **Local Reconstruction Codes**. Not only the name is similar, but also both codes are very similar. The key idea is the same, the blocks are split up into groups and additionally to the global parities, for each group a local parity block is computed. Figure 4 shows an example of Microsoft's LRC (mLRC) code with 6 data blocks and in total 4 parity blocks (2 global, 2 local), though they highlight that this is not actually the code used in production, but a smaller version with less data fragments.

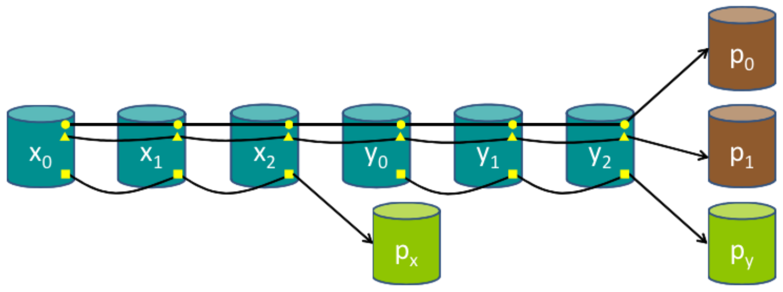


Figure 4: Microsoft's LRC code with $k = 6$ data fragments, $t = 2$ global parities and 2 local parities

The mLRC code in figure 4 with 6 data blocks and 4 parity blocks cannot cope with *arbitrary* 4 block failures as a $(10, 6)$ RS code could do, because mLRC does not have the MDS property. For example, if the data blocks x_0, x_1, x_2 and the local parity block p_x were erased, mLRC could not recover them. Such a failure pattern is called *information-theoretically* non-decodable, because it is impossible to recover 3 data blocks from merely 2 global parity blocks. However, mLRC can successfully repair any failure pattern that is *information-theoretically* decodable, which means that if m blocks are erased there are at least m other valid blocks (data blocks from the same local group, local or global parity blocks) available for the repair. A code that is able to repair all such failure patterns is said to have the *Maximally Recoverable* (MR) property and also Facebook's LRC (fLRC) has this property.

WAS does not use RS erasure codes, to compute the local or global parity blocks. Instead it has a set of coding equations that define how the parities are computed. Let

$$\begin{aligned} q_{x,0} &= \alpha_0 x_0 + \alpha_1 x_1 + \alpha_2 x_2 & \qquad q_{y,0} &= \beta_0 y_0 + \beta_1 y_1 + \beta_2 y_2 \\ q_{x,1} &= \alpha_0^2 x_0 + \alpha_1^2 x_1 + \alpha_2^2 x_2 & \qquad q_{y,1} &= \beta_0^2 y_0 + \beta_1^2 y_1 + \beta_2^2 y_2 \\ q_{x,2} &= x_0 + x_1 + x_2 & \qquad q_{y,2} &= y_0 + y_1 + y_2. \end{aligned}$$

Then, the local and global parities can be computed as follows

$$\begin{aligned} p_0 &= q_{x,0} + q_{y,0} & \qquad p_x &= q_{x,2} \\ p_1 &= q_{x,1} + q_{y,1} & \qquad p_y &= q_{y,2} \end{aligned}$$

where the coefficients α_i and β_i are elements of the Galois field $GF(2^4)$. In case of a failure event, the equations can be re-arranged such that the necessary block can be recovered from other blocks in the equation.

4.3 Evaluation Model

We motivated the use of erasure codes in distributed storage systems also by an increased reliability of the storage system, without ever mentioning how much more reliable the system actually is by using such codes. The standard way to measure the reliability of the system is the *mean-time to data loss* (MTTDL), measured in days. A commonly used approach to determine the MTTDL of a system is a Markov model as shown in figure 5. The states of the Markov model are the number of block erasures that we can accept before data is irrecoverably lost. For a $(14, 10)$ RS code or Facebook's LRC, which can both accept up to 4 erasures, the Markov model has 5 states. Instead, for three replication the number of states is only 3. The transition probabilities λ_i are the probabilities of block erasures due to disk failures, and can be estimated assuming a mean time to failure for a disk of 4 years. Similarly, the probabilities p_i are estimated by how fast the block can be repaired, which, for erasure codes, is influenced by the number of blocks necessary for the decoding process.

Table 1 summarizes the MTTDL for the various algorithms. Note that the data of the first three rows were taken from Facebook's paper [6] and the last row

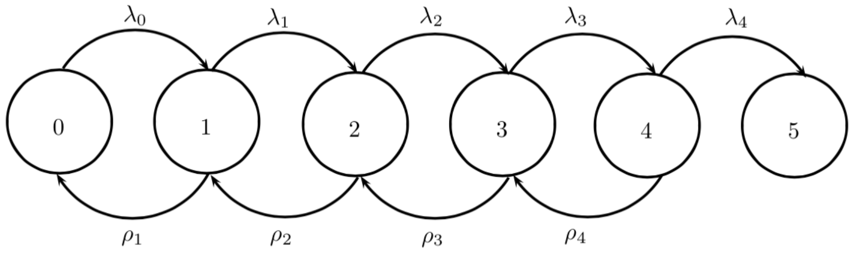


Figure 5: Markov model for a $(14, 10)$ RS code and for Facebook's LRC

Scheme	Storage overhead	Repair traffic	MTTDL (days)
3-replication	2x	1x	2.3079E + 10
RS (14,10)	0.4x	10x	3.3118E + 13
fLRC	0.6x	5x	1.2180E + 15
mLRC	0.66x	3x	9.49E + 14

Table 1: Comparison of the MTTDL measure for different schemes

is taken from Microsoft's paper [3], hence it is difficult to directly compare the two numbers, as they might have been computed using different assumptions on the parameters and transition probabilities in the Markov model. Nevertheless, these numbers offer a rough idea what capabilities those algorithms have. The first observation is that all erasure codes have a considerably higher MTTDL, and thus offer a higher reliability, than the accepted industry standard of three-replication. Moreover, the storage overhead of the erasure code methods is well below that of the standard approach. However, in terms of repair traffic three replication clearly outperforms all of its competitors, though we can see that the LRC codes have a much lower repair traffic than the standard Reed Solomon codes.

5 Conclusion

In this report we studied how erasure coding can be applied in distributed storage systems. First we gave an overview what erasure codes are and where they are applied these days. We discussed one particular kind of erasure code, the Reed Solomon codes, that are one of the oldest, yet still one of the most widely used erasure codes. Reed Solomon codes are a heavily mathematical construct and therefore we briefly introduced some related concepts, such as Galois fields. Then we shifted our focus on distributed storage systems and showed at the example of two large companies, Facebook and Microsoft, how erasure codes can be applied to decrease the storage overhead and increase the reliability of those systems. The importance of erasure coding in such environments can be clearly seen by the fact that companies, like Google, Facebook and Microsoft, who operate planet-scale distributed storage systems, all transition to some form of erasure coding.

References

[1] E. Berlekamp. *Algebraic Coding Theory.* McGraw-Hill series in systems science. Aegean Park Press, 1984.

[2] J. Byers, M. Luby, and M. Mitzenmacher. A digital fountain approach to asynchronous reliable multicast. *Selected Areas in Communications, IEEE Journal on*, 20(8):1528–1540, Oct 2002.

[3] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, and S. Yekhanin. Erasure coding in windows azure storage. In *Proceedings of the 2012 USENIX Conference on Annual Technical Conference*, USENIX ATC'12, pages 2–2, Berkeley, CA, USA, 2012. USENIX Association.

[4] M. Luby. Lt codes. In *Foundations of Computer Science, 2002. Proceedings*, pages 271–280, 2002.

[5] I. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, 1960.

[6] M. Sathiamoorthy, M. Asteris, D. Papailiopoulos, A. G. Dimakis, R. Vadali, S. Chen, and D. Borthakur. Xoring elephants: Novel erasure codes for big data. *Proc. VLDB Endow.*, 6(5):325–336, Mar. 2013.

[7] A. Shokrollahi. Raptor codes. *IEEE/ACM Trans. Netw.*, 14(SI):2551–2567, June 2006.

[8] S. B. Wicker. *Reed-Solomon Codes and Their Applications.* IEEE Press, Piscataway, NJ, USA, 1994.