Delft University of Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft Institute of Applied Mathematics

Comparing Performance of Locally Repairable Codes

(Dutch title: Vergelijken van de Prestaties van Lokaal Repareerbare Codes)

Thesis submitted to the
Delft Institute of Applied Mathematics
in partial fulfillment of the requirements

for the degree

BACHELOR OF SCIENCE
in
APPLIED MATHEMATICS

by

Thomas van der Pas
4454472

Delft, The Netherlands
Tuesday 5th March, 2019

BSc thesis APPLIED MATHEMATICS

"Comparing Performance of Locally Repairable Codes"

(Dutch title: "Vergelijken van de Prestaties van Lokaal Repareerbare Codes")

THOMAS VAN DER PAS

Technische Universiteit Delft

Supervisor

Dr.ir. J.H. Weber

Thesis committee

Dr. J. L. A. Dubbeldam Dr. J. A. M. De Groot

March, 2019 Delft

Preface

This thesis is written as the final part of the Bachelor program in Applied Mathematics at Delft University of Technology. In the thesis, the comparison will be made between the performance of several erasure coding schemes. The family of schemes that this research entails is mainly focused on the so-called *Locally Repairable Code*. Changing parameters within this family of codes, creates performance changes on different aspects of the recoverability of erasures. It therefore becomes interesting to compare which parameter configurations are useful for which applications. The origin of my interest in this field of mathematics lays upon the first time I came into with the Applied Mathematics study. During one of the introductory days of our faculty, Dr. J. A. M. De Groot gave a lecture on the practicality and development of error correcting codes. The way he talked about how error correcting codes enabled the step from long-playing (LP) records to compact discs (cd) possible, has always stuck with me [1].

No wonder that I later took the course "Applied Algebra; Codes and Cryptography Systems", lectured by this same Dr. De Groot. This course formed a profound mathematical basis for writing the rest of this thesis. Last year, I took the course "Big Data Processing", in which the field of erasure coding was approached from a Computer Science point of view. This combination of knowledge enabled me to answer my thesis statement through analytical research.

The basis for this thesis stems from my passion for developing methods of handling and storing large amounts of data. As the world moves further into the digital age, generating thousands of gigabytes of data, a greater need to safely store and access all data arises [2]. The large amounts of data traffic that follows can cause data storage systems to crash and possibly lose data. Therefore, backups of all data is needed, leading to large storage overhead and traffic to repair these crashes [3]. In this thesis, we will look for solutions to safely store data without increasing the amount of storage overhead needed. We will also look at the influence of the data repair traffic within these storage systems. The Locally Repairable Codes, built upon the well-known Reed-Solomon code family and described in [4], does an attempt in reaching an optimum in the trade-off between data reliability and storage overhead. Therefore, my thesis is built upon the erasure code these researchers have managed to create.

Finally, I would like to use this opportunity to express my gratitude to a few. First and most to my supervisor, Dr. ir. J. Weber, for his patient advice and guidance throughout the research process. Lastly, my committee members, who were willing to review my thesis and presentation. Of these committee members, I would like to show an extra word of appreciation to Dr. De Groot. As mentioned earlier, he brought me in contact with different facets concerning erasure coding. Moreover, he got me in touch with my supervisor, Dr. Weber.

Thomas van der Pas
Delft, March 2019

Abstract

With the internet growing exponentially, the amount of information stored digitally becomes enormous. Storage systems are expected to withstand failures to prevent the loss of data. To provide the reliability, data is stored multiple times in different storage units. If in this case, one of the storage unit goes offline, a backup of the data will be available in another. Needing to deal with duplication causes massive amount of excessive storage overhead. To prevent this overhead, erasure coding can be used. This form of coding can reduce the storage space needed, while still being able to provide a certain level of fault tolerance. Erasure coding makes use of so-called parities; symbols that do not directly contain data, but can be used to retrieve actual information nonetheless. When informational symbols get lost, due to a failure, the parity symbols are combined to reconstruct the lost symbol. Since multiple parities might be used in this process, repair traffic is generated. This traffic slows down the rest of the data flow, retaining quick access to the data. The amount of traffic needed to recover an erasure, is called the locality of a code. Codes with small amounts of storage overhead tend to have a greater locality, causing a lot of repair traffic in case of an erasure.

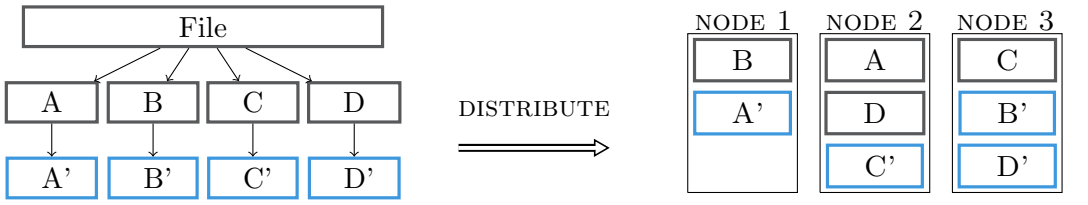
In this thesis, multiple codes are being compared based on their locality and storage overhead. Besides these characteristics, a large code distance is favorable in erasure coding schemes. The distance within the code words of a scheme, determines how many erasures the scheme can handle without losing data. The Maximum Distance Separable (MDS) codes, such as the widely used Reed-Solomon codes, are optimal in the trade-off between code distance and storage overhead. In other words, these codes only need a little amount of overhead in order to withstand a large number of erasures. The pitfall of these codes, however, is their large locality which makes them almost unusable for some applications. Nevertheless, they form a great base for another type of coding scheme; the Locally Repairable Codes (LRC). These codes are extended with extra parity symbols on top of the MDS code which ensures a code locality lower than that of the MDS codes. Since less traffic is needed to repair erasures, they form an interesting group considering the compromise between code locality and storage overhead.

The foundation of this thesis is formed by the content of the research paper "XORing Elephants: Novel Erasure Codes for Big Data" [4]. In this paper, researchers explain the creation of the $(10, 6, 5)$ Locally Repairable Code and compare its characteristics to those of the $(10, 4)$ Reed-Solomon code. On top of the researchers' findings, different parameter configurations will be applied to discover more of the LRC family in this thesis. Different configurations create codes that perform differently in recovering erasures. After comparing their characteristics, applications will be sought for each of the newly created LRCs. As one code might be optimal on the aspect of storage overhead, another might be optimal on the aspect of locality and yet another on the aspect of code distance. In Table 3 of the conclusion it shown which codes excels on what aspects.

Contents

1 Introduction

Due to the vast growth of data, storage systems storing multiple petabytes of data are becoming common today. These systems have to tolerate different failures, *erasures*, arising from unreliable components, software glitches, machine reboots, and maintenance operations. To guarantee high reliability and availability despite these failures, data is replicated (Figure 1a) and distributed across spatially distributed storage units, called *nodes* (Figure 1b). Note that the data is never placed in the same node as its replication. The reason for this will become clear in the process of recovering data after an erasure. The distributed storage system now consists of actual, *information data*, and excessive, replicated data. This last groups forms the *storage overhead* of the system. The ratio between information data and the total amount of data is called the *information rate*, which is closely related to the storage overhead. Excessive storage should be evaded so that storage overhead is minimized, while keeping the size of information data the same. By definition, this minimization causes an increase in information rate. On the other hand, we will see that a larger storage overhead can improve the reliability of a code. Therefore, it is always a *trade-off* between storage overhead and code reliability. Finding an optimum for different purposes within this trade-off drives the findings of new code designs.



(a) Partition and replication of file data.

(b) Distribution of data over nodes.

Figure 1: Process of distributing partitioned and duplicated file over three nodes.

Within a storage system, a user can request or *read* data stored in a node. If a failure occurs in one of the nodes, it will go offline. Reads to the - now unavailable - data in the failed node will result in *degraded reads*. In the period between failure and recovery, these reads are degraded because the system must first reconstruct the data stored in the unavailable node [5]. Only afterwards the original read can be executed. In the case of Figure 1, the reconstruction process is rather fast. If, for example, NODE 3 in figure 1b fails, while a user tries to read data part C of the file, the recovery process would take the C' piece of data stored in NODE 2 and replicate this into a new node, NODE 3'. After NODE 3 has been replicated into this new node, the read will be redirected to the first piece of data of NODE 3'. Now it becomes clear that if backup data would have been stored in the same node as the original data, this recovery would have become impossible. Both the original as well as the replicated piece of data would then have been erased. The reconstruction process, causing *repair traffic*, is a slower operation than reading the data immediately. The amount of data necessary for reconstruction, which is given by the *repair set* of that data, is measured in *locality*. A code with larger repair sets implies that the code has a greater locality, which thus leads to recoveries taking longer.

Clusters of nodes at Facebook generally store petabytes of information. Study shows that daily, a median of 50 nodes in these Facebook cluster are unavailable. A typical data node will be storing approximately 15 TB and the repair traffic with the current configuration is estimated around 10-20% of the total average of 2 PB/day cluster network traffic. Moreover, it is shown that a median of 180 TB of cross-rack traffic is generated as a result of node unavailability [4,6]. This illustrates the importance of research and development in the field of erasure recovery. Improved systems could namely save lots of 'unnecessary' data traffic clogging up the server.

1.1 Erasure Coding

The simplest way of enabling erasure recovery is by data replication as shown in Figure 1. This still happens on a large scale, but then mostly with all data replicated twice, called a 3-replication scheme. Using this scheme, a system is able to recover its data even after two erasures. An advantage of the replication of data is that recovery only asks for a small amount of repair traffic, namely only the replication itself. This scheme therefore has a small code locality. This pays off for nodes receiving many input/output (I/O) requests, resulting in quite some erasures. For the so-called *hot data* in these busy nodes, needing to deal with larger localities would create a bottleneck in its remaining traffic. Quite a large disadvantage of this scheme on the other hand, is the 200% overhead that the 3-replication incurs. Especially for data sets with relatively low I/O activity, the data replicas are rarely accessed, but still consume the same amount of storage space.

The group of data sets that deals with far less I/O activity is called the *cold data*. To show why exactly this cold storage is such an important group to look into, Facebook serves as a great example once more. This social media company stores more than 240 billion photos, with users uploading an additional 350 million new photos every single day. To house those photos, Facebook's data center team deploys 7 petabytes of storage gear every month. But not all of that photo data is created equal. An analysis of Facebook's traffic found that 82% of traffic was focused on just 8% of photos [7]. While this 8% might benefit from replicating its contents, other solutions should be sought for the remaining 92%. For this data, an improvement has overtaken to use different sorts of *erasure coding* instead. This form of coding uses far less storage space while still being able to provide the same level of fault tolerance. Erasure coding makes use of so-called *parities*, symbols that do not directly contain data, but can be used to retrieve actual information data when erased. The way these symbols do this, differs from case to case.

Table 1: Binary addition.

+	**0**	**1**
0	0	1
1	1	0

One of the simpler erasure codes contains a parity symbol based on the exclusive-or (XOR) operation. To show the functionality of this code, we will display the content of each part of the file (Figure 1a) as if it was just a single bit. Note that the only two elements in this *binary* field are 0 and 1, and that the additive behaviour of this field is as shown in Table 1. This behaviour corresponds to that of the XOR operation, which is represented by a $\oplus$ sign. After the file has been translated into a set of single bits, we will determine the single parity of this coding scheme. As an example file, take the bits $A = 1$, $B = 1$, $C = 0$ and $D = 1$. This creates the *message* 1101, to which we would like to add a parity. This parity will be the outcome of the XOR operation on all bits of the message. Therefore, it will be $1 \oplus 1 \oplus 0 \oplus 1 = 1$. Adding this parity bit to the message generates the *code word* "11011". This process is called *encoding*. This specific coding scheme has the property that its code words always contain an even number of ones.

To see that indeed this coding scheme can recover missing data, view an example in which the node containing the B part of the data set fails. The code word 1?011 remains. It can easily be recomputed that the question mark hides a 1, because only then the number of ones in the code word is even, which was a given using the XOR coding scheme. Since the recovery process of this scheme infers an operation on all parts of the file - one needs to check the content of all other data bits -, this form of erasure coding has a much greater locality than the replication scheme. The storage overhead on the other hand is reduced; from 200% necessary for the 3-replication scheme, to a much improved 25%. The trade-off between locality and storage overhead is always considered when optimizing the erasure coding design.

1.2 Thesis Statement

Besides the XOR coding scheme, other ways of setting up parities to overcome erasures are possible. One of the most famous erasure codes is the one found by, and named after the two scientists I. Reed and G. Solomon (Section 3). This code is a member of the Maximum Distance Separable (MDS) code family. In short, being an MDS code ensures that the code performs optimally in the trade-off between the number of erasures it can overcome, and the information rate of the code words. More information on this family of codes is provided in Section 2.3. Despite, or actually because of, being the optimum between recoverability and information rate, these codes have to give in greatly on their code locality. To improve on this last characteristic, without losing much on the others, an alternative erasure code is proposed in the article "XORing Elephants: Novel Erasure Codes for Big Data." [4]. The so-called Locally Repairable Code scheme, is built on top of the Reed-Solomon codes, to which it owes its favorable information rate and recoverability. This thesis is built upon the findings explained in the article with regards to these LRCs. Within the thesis, we will look for similar coding schemes and their characteristics using different sets of parameters. We will thereby look for useful applications for each scheme.

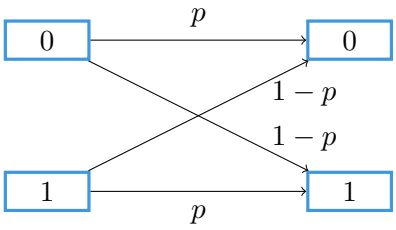
1.3 Organization

The remainder of this paper is organized as follows. Section 2 contains an overview of the necessary concepts of coding theory. In Section 3, Reed-Solomon codes are introduced as an application of coding theory. In Section 4, we then present our theoretical results; the construction of several Locally Repairable Codes and their characteristics. We defer the more technical proofs to the Appendix. After the creation of these LRCs we will compare their performances and look for a useful application for each one of them in Section 5. We finally conclude and survey possible future work in Sections 6 respectively 7.

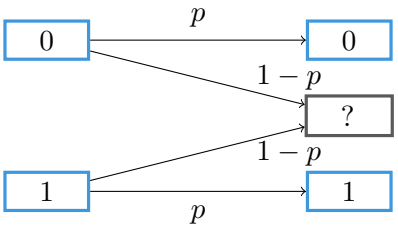
2 Prerequisites

2.1 Coding Theory

Throughout the coding theory, error correcting and erasure repairing coding have been used side by side. In a noisy environment, with chance p for a bit to be transferred correctly, there is a $1-p$ chance for the data transmission to fail. In a binary environment, such a transmission failure can either implicate that a 0 is read as a 1, or the other way around (figure 2a). This causes an error in the code word, which can be detected and corrected by error correcting codes. Another option is that the failure implicates an unreadable bit (figure 2b), which causes an erasure. This can, as the name suggests, be repaired by an erasure repairing code. From this point on, the focus will only be on this second group of codes; the erasure codes.



(2a) Chart of chances on error.



(2b) Chart of chances on erasures.

The main principle of erasure coding is adding symbols to data in order to enable erasure recovery in the data. The symbols are elements of a predetermined *alphabet*, Σ. This alphabet could for example exist of the symbols 0 and 1 or the 16 symbols of the hexadecimal number system. The size $|\Sigma|$ is often written as q, with q a prime power. The alphabet is associated with its corresponding *field*, $\mathbb{F}_q$. Roughly speaking, a field is a set of elements in which we can do additions, subtractions, multiplication, and division without leaving the set. In the example of the XOR operation in Section , the binary field is in play. For the binary alphabet, represented by the binary values $\{0,1\}$, the size of the alphabet equals $q=2$. An example of such a computation within the $\mathbb{F}_2$ was given by the binary addition Table 1. Since the binary code word given in the example (11011) was of length $n=5$, it is a member of the vector space $\mathbb{F}_2^5$. Every element within this field can be seen as a 5-dimensional row vector containing only zeros and ones, just like the example. We can write this erasure coding scheme out in a more general mathematical notation [8].

Example 2.1 (Parity code)**.** *During the encoding process of the general XOR coding scheme, one parity symbol is added in such a way that the sum of all symbols is even. This implies that encoding is performed via the map;*

$$f : \mathbb{F}_2^{n-1} \longrightarrow \mathbb{F}_2^n, f\left((x_1, x_2, \ldots, x_{n-1})\right) = \left(x_1, x_2, \ldots, x_{n-1}, \sum_{i=1}^{n-1} x_i\right)$$

The summation used has equivalent properties to the binary addition given in Table 1. For $n=5$, the corresponding code $\mathcal{C} \subset \mathbb{F}_2^5$ is given by: $\{$00000; 00011; 00101; 00110; 01001; 01010; 01100; 01111; 10001; 10010; 10100; 10111; 11000; 11011; 11101; 11110$\}$.

In the example above, the code words of code $\mathcal{C}$ are expressed in the form of a list. Especially with codes of greater length, this representation can become quite cumbersome. To prevent this inconvenience, Section 2.5 will introduce a more compact notation of a code.

2.2 Linear Block Codes

The most widely used erasure codes belong to the type of *block codes*. For a block code, the information sequence is divided into message blocks of k information symbols each. A message block is represented by the k-tuple $\mathbf{u} = (u_0, u_1, \ldots, u_{k-1})$, with $u_i \in \Sigma, \forall i \in \{0, \ldots, k-1\}$ and is called a *message*. There is a total q^k different possible messages. Each message $\mathbf{u}$ is then independently transformed into an n-tuple $\mathbf{v}= (v_0, v_1, \ldots, v_{n-1})$ of discrete symbols, called a *code word*. Therefore, corresponding to the q^k different possible messages, there are q^k different possible code words. This set of q^k code words of length n is called an $[k, n-k]_q$ *block code*. The ratio $R = k/n$ is called the *code rate*, and can be interpreted as the number of information symbols per transmitted code word. Important to note is that the n-symbol output code word depends only on the corresponding k-symbol input message, which makes block codes memoryless. A desirable structure for a block code to possess is the *linearity*. With this structure in a block code, the encoding complexity will be greatly reduced.

Definition 2.2 (Linear Block Codes)**.** *A block code of length n and q^k code words is called a linear $[k, n-k]_q$ code if and only if its q^k code words form a k-dimensional subspace of the vector space of all the n-tuples over the field $\mathbb{F}_q^n$.*

For a block code to be useful (i.e., to have a different code word assigned to each message), $k \leq n$ should hold. When $k < n$, $n-k$ redundant symbols are added to each message to form a code word. These so-called *parity* symbols provide the code with the capability of repairing erasures. For a fixed code rate R, more redundant bits can be added by increasing the block length n of the code while keeping the ratio k/n constant. As shown in the XOR coding scheme example, one parity symbol can already make messages more erasure proof. Since $\mathcal{C}$, defined in Example 2.1 solely contains words with an even number of 1's, it can be used for single erasure repair. In case one symbol is erased, its value can be determined by all remaining symbols. The reason that this specific coding scheme can recover one and only one erasure, is because of the *distance* between all the code words as written out in Example 2.1.

2.3 Code Distances

The distance between codes is measured using the Hamming Distance.

Definition 2.3. *(Hamming Distance) Let $x, y \in \mathbb{F}_q^n$. The Hamming distance $d(x, y)$ between x and y is equal to the number of code symbols in which x and y differ.*

The number of erasures that a code can repair, depends on the minimum distance between the different code words.

Definition 2.4 (Minimum Code Distance)**.** *Let $\mathcal{C}$ be a code. The minimum Hamming distance of $\mathcal{C}$ is defined as*

$$d = \min_{\{x,y \in C : x \neq y\}} d(x, y)$$

where $d(x, y)$ is the Hamming distance between x and y.

This parameter d will be introduced in the way linear block codes are noted. From now on we will use the message length k and code length n in combination with the minimum code distance d to express an $[k, n-k, d]_q$ code. Suppose we have such a code with minimum code word distance d, or a code with distance d for short. What is then the relation between this d and the number of erasures this code can recover? It turns out that a code word containing $d-1$ erasures, can still only be mapped to the original code word. Therefore, retrieval of the

origin is possible. However, any more than $d-1$ erasure will lead to confusing the code word with another (see Lemma 2.5). This way, we are able to distinguish the distance of a code, using the number of erasures it can recover. Looking at the XOR coding scheme for example, we saw that the code word 11011 could be retrieved from the word 1?011. One more erasure could occur, leading to a word 1??11 for example. This word could be mapped to either 11011 or 10111, leaving uncertainty about the original code word. Judging by the earlier determined statement, this would make up for a code distance $d = 2$ for the XOR coding scheme. Looking at all code words of $\mathcal{C}$ shown in Example 2.1, we can indeed identify words that only differ in two symbols (e.g. 11011 and 01111), but no two words with lesser difference than that. This confirms that the distance of the XOR coding scheme is indeed $d = 2$.

Lemma 2.5 (Minimum Code Distance)**.** *The minimum distance d of a code of length n, is equal to the minimum number of erasures of coded blocks after which the file cannot be retrieved.*

The addition of redundant symbols can propel the growth of the metric d, as shown in the example of the XOR coding scheme. Of course, the redundant symbols need to be chosen wisely, because some additions leave the minimum code distance unchanged (e.q. adding one and the same symbol to every code word). For the erasure tolerance, it is rewarding to choose the parity symbols in such a way that all serve the cause of enlarging d. This way a code achieves the optimal trade-off between its relative distance compared to its code rate. The *Maximum Distance Separable* (MDS) codes have inherited the quality of having the largest possible distance given the n and k of the code. The distance d_{MDS}, as shown in Equation 1, between the code words of an $[k, n-k]_q$-MDS code, gives it the property that any k out of the n coded blocks can be used to reconstruct the entire file. One of the most famous Maximum Distance Separable codes is the Reed-Solomon Code, which will be explained more thoroughly in Section 3.

$$d_{\text{MDS}} = n - k + 1 \tag{1}$$

2.4 Code Locality

With the Maximum Distance Separable codes being optimal in the trade-off between minimal code distance and code rate, one could expect that all replication coding has been replaced by these codes. The replication codes do, however, score much better than the MDS codes on the aspect of code locality. This metric refers to the amount of repair traffic caused by erasures.

Definition 2.6 (Locality)**.** *Consider a linear $[k, n-k, d]_q$ code $\mathcal{C}$. We say that that i-th symbol of $\mathcal{C}$ has locality r_i, if the value at this symbol can be recovered from accessing some other r_i symbols of $\mathcal{C}$ [9]. We say that $\mathcal{C}$ has locality r, if all symbols have locality r or less.*

In the case of a single erasure in the replication coding scheme, only its replica needs to be accessed to recover the erased symbol. Its locality is therefore equal to 1. The property of the $[k, n-k]_q$-MDS code that k out of the n symbols are used to reconstruct an erased symbol, causes its high valued locality of at least k.

Lemma 2.7. *Maximum Distance Separable codes with parameters k and n cannot have locality smaller than k. [4]*

Lemma 2.7 implies that MDS codes have quite an unfavorable locality, since at least k blocks are required to reconstruct a code word after a single erasure. These k symbols used in recovering the erasure are called the repair set of the erased symbol.

Definition 2.8 (Repair Set)**.** *Let $\mathcal{C}$ be a linear $[k, n-k, d]_q$-code and $i \in [n]$. A subset $\mathcal{R}_i \subseteq [n] \setminus \{i\}$ is a repair set of i if for every code word of $\mathcal{C}$ its i-th symbol can be repaired using the symbols indexed by $\mathcal{R}_i$.*

The larger the repair set, the more repair traffic an erasure recovery causes. In other words, codes with larger repair sets, have higher localities. Following from Definitions 2.6 and 2.8, we can derive the following corollary.

Corollary 2.8.1. $r_{\mathcal{C}} = \max_{i \in [n]} |\mathcal{R}_i|$.

The locality is the reason that replication coding, with smaller amounts of repair traffic, is still mainly in use for data storing systems. Especially to the hot, high erasure risk, data systems this excessive traffic is inconvenient. Companies choose the massive amounts of overhead caused by replicating all data over this cumbersome traffic. For the cold data systems on the other hand, the MDS code is a promising solution against the large amounts of storage overhead. Since cold data is not invoked much, chances on erasures are smaller than for hot data. Repair traffic will therefore be much less, and could give wiggle space for fighting the overhead by implementing this form of erasure coding. As shown in the Introduction (Section 1.1), cold data takes up a large part of the overall data storage. How to choose redundant symbols to achieve quick recovery with little overhead is the major challenge with designing erasure codes. This challenge drives the search for modified and improved erasure codes.

2.5 Codes as Matrices

As shown in Example 2.1, a code can be expressed as a list of code words. This becomes quite cumbersome, when one starts to work with binary messages of length $k = 8$ - called bytes - for example. This code will namely contain $2^8 = 256$ different code words already. A more convenient way of forming all elements of a linear block code is by the use of its so-called *generator matrix* (see Definition 2.9). This matrix is composed of rows which are the basis vectors of the code. Every code word can be generated by a linear combination of these basis vectors (see Example 2.10). Recall that a basis of a vector space $\mathcal{C}$ over $\mathbb{F}_q^n$ is a subset $\{\boldsymbol{g_1}, \ldots, \boldsymbol{g_k}\}$ such that every code word $c \in \mathcal{C}$ can be uniquely written in the form

$$c = \lambda_1 \boldsymbol{g_1} + \lambda_2 \boldsymbol{g_2} + \ldots + \lambda_k \boldsymbol{g_k}, \tag{2}$$

where $\lambda_1, \ldots, \lambda_k$ are elements of $\mathbb{F}_q$. Because of this linearity, the $(k \times n)$ generator matrix is able to generate the entire list of code words. This is shown for the XOR coding scheme in the example below.

Definition 2.9 (Generator Matrix)**.** *Suppose $\mathcal{C}$ is a linear $[k, n-k, d]$-code. A generator matrix for $\mathcal{C}$ is a $(k \times n)$ matrix with entries in $\mathbb{F}_q$, whose rows form a basis for $\mathcal{C}$.*

Example 2.10. *All code words of the XOR coding scheme (Example 2.1), can be formed by linear combinations of the vectors $\{10001, 11000, 11101, 11110\}$. This property makes these vectors the basis of this particular coding scheme.*

$$
\begin{aligned}
00000 &= 0*10001 + 0*11000 + 0*11101 + 0*11110 \\
00011 &= 0*10001 + 0*11000 + 1*11101 + 1*11110 \\
00101 &= 0*10001 + 1*11000 + 1*11101 + 0*11110 \\
&\vdots
\end{aligned}
\qquad
\begin{aligned}
&\vdots \\
11011 &= 0*10001 + 1*11000 + 1*11101 + 1*11110 \\
11101 &= 0*10001 + 0*11000 + 1*11101 + 0*11110 \\
11110 &= 0*10001 + 0*11000 + 0*11101 + 1*11110
\end{aligned}
$$

In general, a linear code $\mathcal{C}$ will have lots of different bases to choose from. This means that different generator matrices $\mathbf{G}$ exist for code $\mathcal{C}$. Any row operation on $\mathbf{G}$ will not change the ability to generate all code words of $\mathcal{C}$ as shown by Lemma 2.11 [10].

Lemma 2.11 (Matrix Operations)**.** *Suppose $\mathcal{C}$ is a linear $[k, n-k, d]$-code with generator matrix $\mathbf{G}$. If the matrix $\mathbf{G}'$ can be obtained from $\mathbf{G}$ by applying the following operations:*

MO1. Permuting the rows;

MO2. Multiplying a row by a non-zero element of $\mathbb{F}_q$;

MO3. Adding a multiple of a row to another row;

MO4. Permuting the columns;

MO5. Multiplying a column by a non-zero element of $\mathbb{F}_q$;

repeatedly, then $\mathbf{G}'$ is also a generator matrix for a code $\mathcal{D}$ equivalent to $\mathcal{C}$.

Using this Lemma we can permute the basis found in Example 2.10, into the following matrix.

$$\begin{pmatrix} \mathbf{g}_1 \\ \mathbf{g}_2 \\ \mathbf{g}_3 \\ \mathbf{g}_4 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix} \Rightarrow \begin{pmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix} = \left(\mathbf{I}_k \mid \mathbf{A}\right), \tag{3}$$

where $\mathbf{I}_k$ is the $k \times k$ identity matrix, and $\mathbf{A}$ is some $k \times (n-k)$ matrix. This new matrix $\mathbf{G}'$, which is also a generator matrix of $\mathcal{C}$, is now written in *standard form*. The generator matrix in standard form is unique for its linear code. This matrix has an advantage on encoding message using this matrix. Since the vector is first multiplied by the identity matrix, the message is kept intact. It only receives an extension after multiplication with the $\mathbf{A}_{k\times(n-k)}$-matrix. This property ensures a quicker decoding process.

Complementary to the generator matrix, we can use a *parity-check matrix* to check whether a vector is a coded word (see Definition 2.12). All generator matrices have complementary parity-check matrices. An additional benefit of the generator matrix in standard form, is that its parity-check matrix is easy to find. If a generator matrix of a code $\mathcal{C}$ is of the standard form $\mathbf{G} = \left(\mathbf{I}_k \mid \mathbf{A}\right)$ then a parity-check matrix of this code is $\mathbf{H} = \left(-\mathbf{A}^T \mid \mathbf{I}_{n-k}\right)$. Therefore, the parity-check matrix of Equation 3 would become $\mathbf{H} = \begin{pmatrix} -1 & -1 & -1 & -1 & \mid & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \end{pmatrix}$, because of the binary field. This matrix and Definition 2.12 give away the property that code words of the binary XOR coding scheme should always contain an even number of ones. If a word $\mathbf{x}$ contains an odd number of $\mathbf{1}$'s, then $\mathbf{Hx} \neq \mathbf{0}$, which is equivalent to $\mathbf{x}$ not being a code word of the scheme.

Definition 2.12 (Parity-check)**.** *Let $\mathcal{C}$ be a $[k, n-k, d]_q$-code. A $((n-k) \times n)$ matrix $\mathbf{H}$ over $\mathbb{F}_q$ such that $\mathbf{x} \in C \iff \mathbf{Hx} = \mathbf{0}$, is called a parity-check matrix of $\mathcal{C}$.*

Definitions 2.9 and 2.12 lead to the fundamental property given below.

Property 2.13 (Fundamental Property)**.** $\mathbf{GH}^T = \mathbf{0}_{k\times(n-k)}$.

3 Reed-Solomon Codes

Up until this point we have examined the XOR erasure coding given in Example 2.1 with its basis given in Example 2.10. In this Chapter we will introduce another particular linear block code family; the Reed-Solomon codes. Before we go into detail on this type of code, a short introduction of the two gentlemen inventing this code will be provided.

Irving S. Reed and Gustave Solomon were both mathematicians and engineers. Together they made a leap in the algebraic theory of error detection and correction. With their publication of the article "Polynomial codes over certain finite fields" [11], they introduced the now widely used *Reed-Solomon code*.

3.1 Definitions

In the work of Reed and Solomon, each code word is represented by a sequence of function values of a polynomial of degree less than k. In order to obtain a code word of the Reed-Solomon (RS) code, the message is interpreted as the description of a polynomial p of degree less than k over the finite field $\mathbb{F}_q$. In turn, the polynomial p is evaluated at $n \leq q$ distinct points $\{a_1, \ldots, a_n\}$ of the field $\mathbb{F}_q$, and the sequence of values is the corresponding code word.

Definition 3.1 (Reed-Solomon)**.** *The Reed-Solomon code $(k, n-k)RS \subseteq \mathbb{F}_q^n$ of dimension k over a finite field $\mathbb{F}_q$ with evaluation points $A = \{a_1, a_2, ..., a_n\} \subseteq \mathbb{F}_q$ is defined as:*

$$(n, n-k)RS = \{(f(a_1), ..., f(a_n)) : f \text{ is a polynomial over } \mathbb{F}_q \text{ of degree} < k\}.$$

The parameter for distance d is left out in the declaration of a $(k, n-k)$ RS code, because this parameter is given by Equation 1. The distance can therefore be easily deduced using only the n and k.

For practical uses of Reed-Solomon codes, it is common to use a finite field $\mathbb{F}_q$ with $q = 2^m$ elements. In this case, each symbol can be represented as an m-bit value. The sender sends the data points as *encoded blocks*, which will contain $n = 2^m - 1$ symbols. Thus, a Reed-Solomon code operating on 4-bit symbols has $n = 2^4 - 1 = 15$ symbols per block. The number k, with $k < n$, of symbols in the block is a design parameter and represents the number of information symbols per block. The larger the difference between k and n, the more non-information symbols the code entails. Containing non-informative symbols might sound like a disadvantage, but it enables the code to correct erasures. So, the larger this difference, the more erasures can be recovered using the RS-code. Even stronger, the difference between n and k is equal to the number of erasures that can be overcome: $n-k = e = d-1$, with e the number of erasures that can be overcome, and d the distance. This shows once more that the distance of the RS-code $d_{\text{RS}} = n - k + 1$. This is a distance equation we have already come across in Equation 1 of Section 2.3. In Section 2.3, the characteristics of the Maximum Distance Separable Codes are addressed with their advantages and disadvantages. In books and online, many examples can be found of creating $(2^m - 1 - e, e)$ Reed-Solomon codes, which can be fully designed around its erasure recoverability e. Doing so, requires quite some steps, such as finding the generator polynomial $f(x)$ and evaluating this around the points $\{a_1, a_2, \ldots, a_n\}$. In this thesis we will neglect these steps and continue after the point where the generator matrix of the RS code has been found. As stated in Definition 2.9, this $k \times n$ sized matrix consists of rows forming a basis of the RS code. In Lemma 3.2 the equation for the parity-check matrix of an RS code is described.

Lemma 3.2. *The $(k \times n)$ parity-check matrix of a $(k, n-k)$ Reed-Solomon code over a field $\mathbb{F}_q$, of order $q = 2^m$ is given by $[H]_{i,j} = a_{j-1}^{i-1}$, where $a_1, a_2, \ldots, a_n$ are n distinct elements of the field $\mathbb{F}_q$, for $i \in [k]$ and $j \in [n]$.*

3.2 Shortening and Extending

So far in the thesis, it might have appeared as if Reed-Solomon codes can only be of length $n = 2^m - 1$, $m \in \mathbb{N}$. Nevertheless, RS codes can be of any length n and can contain any number of information symbols $0 < k < n$. The different lengths can be acquired by a process called *shortening*. To explain this, we will use an example in which we want to obtain an RS code of length $n' \neq 2^m - 1$, with k' information symbols. To do so, we find an $n = 2^m - 1$ that is greater than n'. For this n we know that there exists an RS code for every $k < n$ as explained in Section 3.1. Use k that has the same difference from k' as n from n', to form the $(k, n-k)$ RS *mothercode*. This way, $n - n' = k - k' \implies n - k = n' - k'$. We therefore know that the distance d' of our desired RS code equals $d' = n' - k' + 1 = n - k + 1 = d$, i.e. the distance of our mothercode.

The generator matrix of this mothercode can be permuted into the standard form $\mathbf{G} = (\mathbf{I}_k \mid \mathbf{A})$. To reach the desired $(k', n' - k')$ RS code, we will remove rows and columns from this matrix $\mathbf{G}$ without losing distance between the code words. Appendix A shows how we can achieve the generator matrix $\mathbf{G}' = (\mathbf{I}_{k'} \mid \mathbf{A}')$ of our desired $(k', n' - k')$ code. This Appendix depicts an example of creating a (10,4) RS code using the method of shortening.

Besides the method of shortening a code, we can also increase the length of an existing code with the use of *extending*. Extending is a method in which we add symbols to a code. The reason for this can vary. It could be for enlarging the distance, reducing the locality or increase the number of information symbols. Note that, as stated in Property 3.3, the distance of a code is always at most equal to their extended version. In other words, adding symbols to a code will never reduce the minimum code distance of that code. The best way to visualize this phenomenon is by looking at the generator matrix of a code. Extending a code by one symbol implies adding one extra column to this matrix. The code words created by this matrix will on their turn also increase in length. The distance between two code words after extension can stay the same (if both extensions are equal) or increase by one (if the extensions differ). Since this holds for all code words, the distance of the extended code is at least equal to the original code distance.

Property 3.3. *By adding symbols to a code $\mathcal{C}$, the minimum code distance d of the newly created code is at least equal to that of $\mathcal{C}$.*

3.3 The (10, 6, 5) Locally Repairable Code

The methods explained in Section 3.2 make way for a vast variation in Reed-Solomon codes. An example of this variation, obtained through shortening, has already been addressed in Appendix A, in which a $(10, 4)$ RS code was derived using a $(11, 4)$ RS mothercode. The acquired code, schematically depicted in Figure 3, was the foundation of a research on the topic of erasure coding [4]. The researchers used a $(10, 4)$ Reed-Solomon code as the foundation to tackle the issue of the massive amounts of storage overhead, systems are dealing with nowadays. The rest of the thesis will be built upon the researchers' findings.

X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}	P_1	P_2	P_3	P_4

Figure 3: $(10, 4)$ Reed-Solomon code containing 10 information (X_i) and 4 parity symbols (P_j).

In Section 2.3, the fact was noted that Reed-Solomon codes belong to the Maximum Distance Separable codes. We also described the advantage of the MDS codes needing only a little

amount of storage overhead for erasure recovery. Therefore, it serves as an excellent base to tackle the storage overhead issue. This family of codes do, however, have to deal with one major disadvantage; a large code locality. This implies that in the event of an erasure, many of the leftover symbols need to be consulted before being able to reconstruct the erased symbol. Needing to consult multiple symbols in order to recover erased data, creates large amounts of repair traffic, which is a setback when it comes to using erasure coding. An application using erasure coding needs not only to make sure that the data stored on their servers is reliable at all times, but also that it is accessible at short notice. Having to deal with large bottlenecks created by repair traffic on a server supersedes this second requirement.

To overcome this disadvantage, the researchers adapted the $(10, 4)$ Reed-Solomon code using the second of the two methods described in Section 3.2: extending the code. They increased the length of the code in a way such that the locality of the code reduced. The original RS code then becomes a *precode* to this new, extended code. The rest of this section is dedicated to diving deeper into the characteristics of this code family, the so-called *Locally Repairable Codes*. These LRCs are denoted as $(k, n-k, r)$ codes, in which the r represents the locality besides the already familiar n and k. LRCs are constructed on top of MDS codes, from which a Reed-Solomon code is the most common choice. The symbols of the MDS precode are grouped into subsets length smaller than k and then combined with a newly introduced parity symbol. These extra symbols ensure that the locality within each block meets the length of the block, meaning that the locality drops below k. We know from Lemma 2.7 that the MDS code cannot have locality lower than k. Separating the code into these subset therefore ensures that we enable ourselves to improve the locality of the LRC compared to its MDS precode.

3.3.1 Local Parity

To draw a better image of the process of creating these Locally Repairable Codes, we will follow the researchers' work step by step. We will imitate the researchers finding of their $(10, 6, 5)$ LRC. As stated before, they started off with a Maximum Distance Separable code as schematically depicted in Figure 3. Next, they divided the code into subsets $S_1 = \{X_1, \ldots, X_5\}$, $S_2 = \{X_6, \ldots, X_{10}\}$ and $S_3 = \{P_1, \ldots, P_4\}$. By adding the local parity $Q_1 = c_1X_1 + c_2X_2 + c_3X_3 + c_4X_4 + c_5X_5$, a single erasure in S_1 could be repaired by accessing only 5 other symbols. For example, if X_3 is lost, it can be reconstructed by

$$X_3 = c_3^{-1} * (Q_1 - c_1X_1 - c_2X_2 - c_4X_4 - c_5X_5). \tag{4}$$

The multiplicative inverse of the field element c_3 exists as long as $c_3 \neq 0$, which is the requirement that is enforced for all the local parity coefficients. In Appendix C of their work, the researchers proof that coefficients can always be found such that all the equations are linearly independent. This is required for a well-functioning erasure recovery process. Adding the parities Q_2 and Q_3 as shown in Figure 4 enables the second and third subset to function the same way as S_1. The overall locality of the code is now reduced from 10 to 5, thereby creating a $(10, 7, 5)$ LRC.

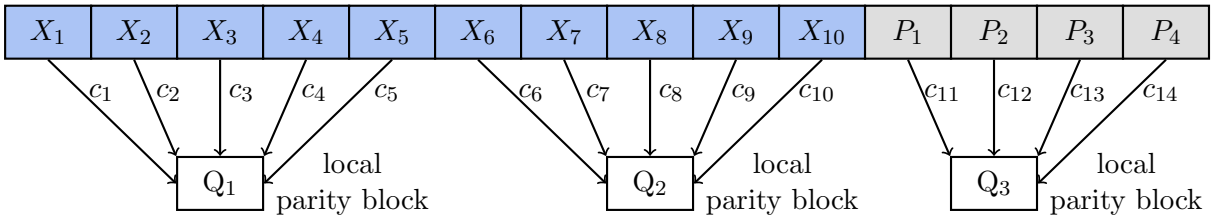


Figure 4: Locally Repairable Code of size 17 and locality 5.

3.3.2 Implied Parity

Although the Locally Repairable Code found by the researchers, was greatly improved considering the code locality compared to that of its Maximum Distance Separable precode, the storage overhead did not meet their satisfaction yet. They were able to perform one additional optimization. In their research paper, they show that the coefficients $c_1, c_2, \ldots c_{14}$ can be chosen such that the local parities satisfy an additional alignment equation $Q_1 + Q_2 + Q_3 = 0$. Therefore, it is unnecessary to store the parity Q_3 locally and instead consider it an implied parity as shown in Figure 5. When an erasure now occurs in the third, modified subset, S_3', the implied parity can be reconstructed and used to repair that failure. For example, if P_2 is lost, it can be recovered by reading 5 symbols P_1, P_3, P_4, Q_1 and Q_2 and solving the equation

$$P_2 = c_{12}^{-1} * (Q_1 - Q_2 - c_{11}P_1 - c_{13}P_3 - c_{14}P_4). \tag{5}$$

Since 5 symbols need to be consulted in order to repair any of the elements in S_3', the size of the repair sets of of these symbols is 5. That means that the locality of the symbols in this last subset has become 5. The first and second subset have not changed during the implication of parity block Q_3. The overall locality of the code therefore has not changed after the modification, and is still equal to $r = 5$.

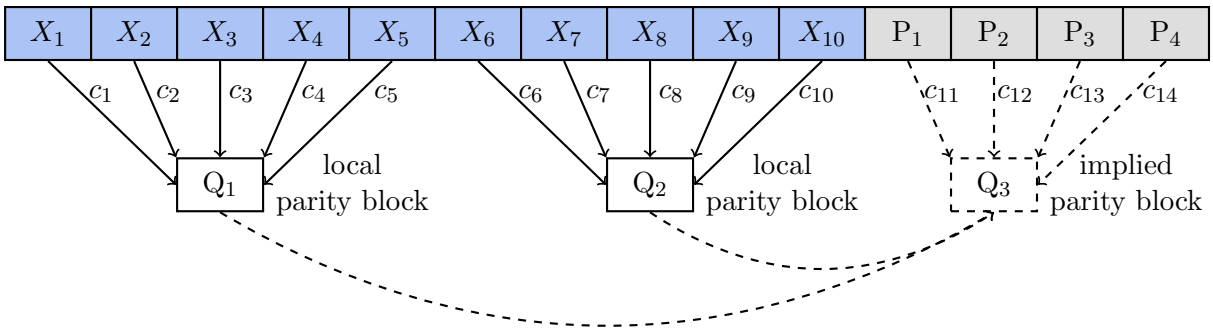


Figure 5: Locally Repairable Code of size 16 and locality 5.

The creation of this Locally Repairable Code is completely given in the research paper [4]. In the paper it is shown that the created $(10, 6, 5)$ LRC has a distance d of 5. It therefore keeps the same distance as its $(10, 4)$ Reed-Solomon precode. Moreover, it improves the code in terms of locality. An Maximum Distance Separable code always has a locality of $r = k$. In the case of the $(10, 4)$ Reed-Solomon code, which is a member of that family, the r would therefore be equal to 10. The $(10, 6, 5)$ LRC following from this RS will on the other hand have a locality of only 5. Quite the improvement.

To use the method of implying parities more broadly for creating Locally Repairable Codes with different parameter sets, a general proof is given in Appendix B.1. In the next section, we will go further into the properties of this family of codes and discover ones with different parameter sets.

4 Locally Repairable Codes

As an addition to the Locally Repairable Code found in Section 3.3, we will dedicate this section to finding similar coding schemes with different sets of parameters. A boundary condition we will enforce on these schemes, will be that $k = 10$. The reason for this condition is so that each coding scheme can be used on similar applications. Only then it becomes interesting to compare their performances. The performance of each of the schemes depends on its locality, storage overhead and code distance. In the upcoming sections, we will calculate the performance of several different parameter sets. In Section 5, comparisons between the codes are made.

Before being able to create coding schemes with different sets of parameters, we first have to generalize some of the properties used in the creation of the previous $(10, 6, 5)$ Locally Repairable Code. To do so, we define Locally Repairable Code $\mathcal{L}$ and proof Properties 4.1 & 4.5 with non-determined parameters n, r and d. As shown in the last section, the precode of $\mathcal{L}$ is the $(k, n-k)$ Reed-Solomon code with a fixed message length $k = 10$ and varying code length $n \in \mathbb{N}_{>10}$. For our LRC $\mathcal{L}$, we propose Property 4.1, which is proven in Appendix B.1. This property enables the parity implication process, such as shown in Sections 3.3.1 and 3.3.2.

Property 4.1 (Parity Alignment)**.** *Let $\mathcal{L}$ be a Locally Repairable Code. Let blocks Q_i be the y parities created by the method of extending the Reed-Solomon precode of $\mathcal{L}$. Within $\mathcal{L}$, each block Q_i is implied by a linear combination of the other parity blocks Q_j $(j \neq i)$ of $\mathcal{L}$.*

The next property we will investigate on a general level involves the locality when implying one of the parity blocks instead of storing them all locally. Using the previous property, we know that it is possible to leave out one of the parity blocks. The removed parity is implied by the other local parities without loss of information nor the ability to correct erasures. In Sections 3.3.1 and 3.3.2, we have seen an example of such implication in action. The transition from the $(10, 7, 5)$ LRC to the $(10, 6, 5)$ LRC shows that with removing one of the parities, the locality of a code will not necessarily increase. To pick a parameter set that behaves according this property, the LRC should satisfy Requirement 4.4. This requirement introduces three new variables: y, $s_{\max}$ and $s_{\min}$.

The variable y corresponds to the number of parity blocks Q_i that are introduced by the extension of the precode to create the Locally Repairable Code. This variable excludes the number of parity blocks that are already contained within the used Reed-Solomon code (the P_is in Figure 6). Since each introduced parity block creates a subset, the created LRC will contain y subsets S_i. These subsets are of size $|S_i|$, which we will introduce as the variable s_i.

Example 4.2 (y and s_i)**.** *In the figure below, the $(10, 4)$ Reed-Solomon code is schematically depicted as the blocks X_1 through P_4. The code is then expanded by $y = 3$ additional parity blocks to create the $(10, 7, 5)$ Locally Repairable Code. This process separates the scheme into three subsets S_1, S_2 and S_3. Of these subsets. both S_1 and S_2 have size s_1 respectively s_2 both equal to 6. The subset S_3, on the other hand, is of size $s_3 = 5$.*

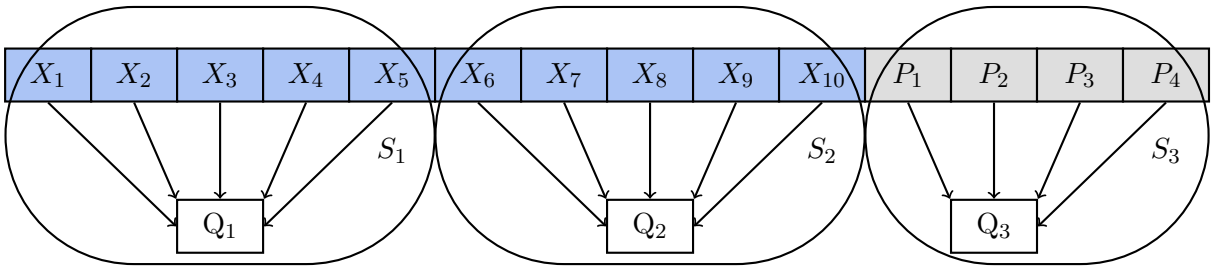


Figure 6: The expansion of 3 parity blocks to form the $(10, 7, 5)$ Locally Repairable Code.

The variables $s_{\min}$ and $s_{\max}$ used in Requirement 4.4, are the sizes of the smallest respectively the largest subset. The formal definition of these variables would thus be

$$s_{\min} = \min_{1 \leq i \leq y} s_i \quad \text{and} \quad s_{\max} = \max_{1 \leq i \leq y} s_i. \tag{6}$$

Example 4.3. *Directly following from Equation 6 and the previous example,* $s_{min} = s_3 = 5$ *and* $s_{max} = s_1 = s_2 = 6$ *of the* $(10, 7, 5)$ *Locally Repairable Code (Figure 6).*

If the following requirement is met by a Locally Repairable Code $\mathcal{L}$ with all parities stored locally, we can say that Property 4.5 holds. Below, we will give a formal proof for this statement. Moreover, an example is given in Section 4.3 in which this requirement is not met. The section also shows the corresponding consequences if one tries to imply a parity in that case anyway.

Requirement 4.4. $s_{min} + y - 2 \leq s_{max}$

Property 4.5. *Let* $\mathcal{L}$ *be a Locally Repairable Code with smallest subset* S_{min}*. If* $\mathcal{L}$ *meets Requirement 4.4, the locality of* $\mathcal{L}$ *remains the same after removing the parity block that belongs to the smallest subset* S_{min}*.*

Example 4.6. *Take a look at the* $(10, 7, 5)$ *Locally Repairable Code and Example 4.3. Since*

$$s_{min} + y - 2 = 5 + 3 - 2 = 6 \leq 6 = s_{max},$$

Requirement 4.4 holds. The researchers were therefore able to imply parity block Q_3*, without increasing the locality of the code (see Section 3.3.2). The reason for selecting parity block* Q_3 *was because belonged to subset* S_3*, which showed to be smallest in size (see Example 4.3).*

Proof (Property 4.5). **Locality before implying any parity blocks.**
To see whether the locality changes after implying one of the parity blocks, we first need to determine the locality $r_{\mathcal{L}}$ of the Locally Repairable Code $\mathcal{L}$ with all parities stored locally. To find the locality of a code, Corollary 2.8.1 showed that it is sufficient to find the largest repair set within the coding scheme. The size of this set namely determines the locality of the code.

Let $\mathcal{R}_j$ be the repair set for the block $j \in \mathcal{L}$ of the Locally Repairable Code with all parity blocks stored locally. Note that each subset S_i functions as the repair set of the members within the subset. In other words, if block $j \in S_i$, then $\mathcal{R}_j = S_i \setminus j$. The size of each repair set is therefore one smaller than the size of the corresponding subset (see Example 4.7). Since this holds for all blocks of the code $\mathcal{L}$, we can generalize to

$$\forall j \in S_i,\ |\mathcal{R}_j| = |S_i| - 1 = s_i - 1, \tag{7}$$

where $i \in \{1, \ldots, y\}$. This shows that the size of the repair sets $\mathcal{R}_j$ of each element depend on the size of the subset S_i it is contained in $(j \in S_i)$.

Example 4.7. *Take another look at subset* S_1 *of the* $(10, 7, 5)$ *Locally Repairable Code in Figure 6. In Equation 4, it is shown that the set* $\mathcal{R}_{X_3} = \{X_1, X_2, X_4, X_5, Q_1\} = S_1 \setminus \{X_3\}$*, which consists of 5 elements, making its size equal to 5. Using variable* s_1*, calculated in Example 4.2, and Equation 7, we can indeed determine that since* $X_3 \in S_1$*,* $|\mathcal{R}_{X_3}| = s_1 - 1 = 5$*.*

As shown in Corollary 2.8.1, we know that the locality of a code is equal to the size of the largest repair set, i.e. $r_{\mathcal{L}} = \max_{j \in \mathcal{L}} |\mathcal{R}_j|$. Using Equation 7, the largest repair set will be of size $s_{\max} - 1$ (defined in Equation 6). Thus, the locality of the Locally Repairable Code before implying any of the parity blocks is $r_{\mathcal{L}} = s_{\max} - 1$.

Proof (Property 4.5 continued). **Locality after implying the parity block located in the smallest subset.**
Now, we would like to determine the locality of the code after removing the parity block within the smallest subset $S_{\min}$. The removed parity, lets call it $Q_{\min}$ is then implied by the other local parities to keep the code's distance and erasure recoverability. However, this implication could change the locality of the newly created Locally Repairable Code, $\mathcal{L}'$. We will determine its locality $r_{\mathcal{L}'}$ by looking at the size of the repair sets $\mathcal{R}'_j$ once more for all $j \in \mathcal{L}'$.

In Section 3.3.2, we determined that the repair sets of some of the symbols changed after removing parity block $Q_{\min}$ and implying it instead. To determine the locality of $\mathcal{L}'$ we will calculate this change in repair set size. First, note that the all subsets $S_i \setminus S_{\min}$, where $i \in [y]$, were not modified by the implication (see Figure 7 where S_1 and S_2 haven't changed compared to Figure 6). Therefore, their repair sets haven't changed, i.e. $\mathcal{R}_j = \mathcal{R}'_j$ for all $j \in \mathcal{L}' \setminus S_{\min}$. The largest repair set for all elements $j \in \mathcal{L}' \setminus S_{\min}$ will therefore still be of size $s_{\max} - 1$.

Example 4.8. *In the figure below, we see the creation of the* $(10, 6, 5)$ *Locally Repairable Code after removing* Q_3*. As determined in Equation 5 of Section 3.3.2, the left over parity blocks* Q_1 *and* Q_2 *can compensate for the now implied parity block. Comparing to Figure 6 of the* $(10, 7, 5)$ *LRC, we notice that nothing has changed in the sets* S_1 *and* S_2*. However, the subset* S'_3 *has increased in the number of members compared to the previous set* S_3 *of the* $(10, 7, 5)$ *LRC. This modification was to compensate for the removed parity* Q_3*.*

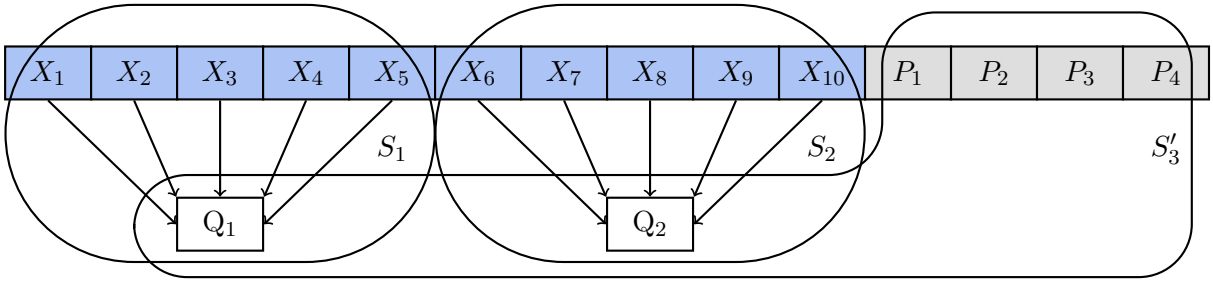


Figure 7: The implication of the parity Q_3 to create the $(10, 6, 5)$ Locally Repairable Code.

To find out whether the locality of the new code $\mathcal{L}'$ has changed, we now only have to look at the elements of the modified subset $S_{\min}$. This set changed in size due the removal of parity block $Q_{\min}$. Instead, the set obtained the remaining parities as depicted in Figure 7. Lets call this modified subset S', since it is not necessarily the smallest anymore. In fact, its size equals

$$\begin{aligned} |S'| &= |S_{\min} \cup \{Q_1\} \cup \ldots \cup \{Q_y\} \setminus \{Q_{\min}\}| \\ &= |S_{\min} \setminus \{Q_{\min}\}| + |\{Q_1\} \cup \ldots \cup \{Q_y\} \setminus \{Q_{\min}\}| \\ &= |S_{\min}| - 1 + y - 1 = s_{\min} + y - 2. \end{aligned}$$

In Equation 7, we determined that the repair sets $\mathcal{R}'_j$ of the elements $j \in \mathcal{L}'$ are one smaller in size than the size of its subset. For the elements $j \in S'$ we can now determine that.

$$|\mathcal{R}'_j| = |S'| - 1 = s_{\min} + y - 3.$$

Looking at the repair sets, we determine that the locality $r_{\mathcal{L}'}$ of $\mathcal{L}'$ is equal to

$$r_{\mathcal{L}'} = \max_{j \in \mathcal{L}} |\mathcal{R}'_j| = \max\left(s_{\max} - 1, |\mathcal{R}'_{j \in S'}|\right) = \max\left(s_{\max} - 1, s_{\min} + y - 3\right).$$

This leads to the property that if $\mathcal{L}$ meets the requirement $s_{\min} + y - 3 \leq s_{\max} - 1$, then $r_{\mathcal{L}} = s_{\max} - 1 = r_{\mathcal{L}'}$.

□

Using Property 4.1 and 4.5, we are able to discover Locally Repairable Codes with different parameter sets, and compare their performances.

4.1 The (10, 5, 7) Locally Repairable Code

Similar to the researchers' design decision given in the previous section, the first parameter set of our Locally Repairable Code will be based upon a $(10, 4)$ Reed-Solomon code. This time though, we we will introduce only two parities on this RS precode, i.e. $y = 2$. These two parities both create a subset of size 8, containing the first seven respectively last seven blocks of the RS precode (see Figure 8). Since this code has an $s_{\max} = s_{\min} = 8$ and a $y = 2$, the equation below shows that Requirement 4.4 holds.

$$s_{\min} + y - 2 = 8 + 2 - 2 = 8 \leq 8 = s_{\max}$$

Using Properties 4.1 and 4.5, we now know that we can imply one of the parity symbols to improve the storage overhead without increasing the code locality. Implying block Q_2 will create the code formation as shown below.

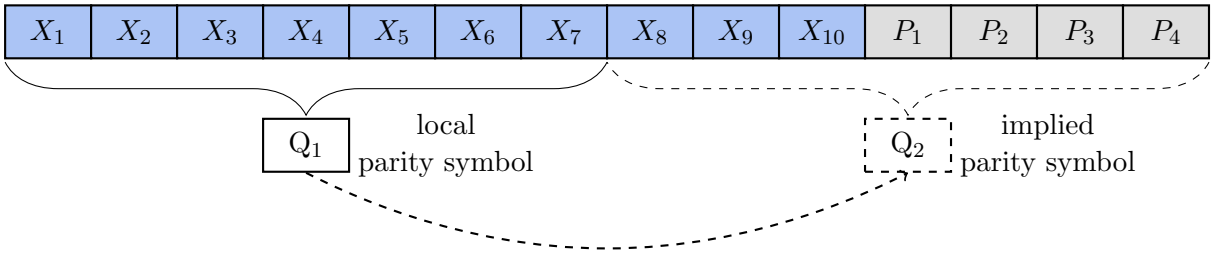


Figure 8: Locally Repairable Code of size 15, distance 5 and locality 7.

Next, we would like to establish the code distance d_{LRC} of our newly created Locally Repairable Code. With the precode being a $(10, 4)$ RS code with $d_{\text{RS}} = n - k + 1 = 5$, we establish using Property 3.3, that $d_{\text{LRC}} \geq 5$. We will now prove that $d_{\text{LRC}} = 5$ is the maximum distance possible for a length $n = 15$ code that has block locality $r = 7$. Lets say the distance of this size 15 code is 6. We will derive a contradiction. The code distance would in this case be equal to $d = 6 = 15 - 10 + 1 = n - k + 1$. This characteristic belongs to the Maximum Distance Separable (MDS) codes only. That would mean that our LRC is a member of the MDS code family. Lemma 2.7 shows however that the locality of an MDS code cannot be smaller than k. The locality of this LRC equals 7, which is much smaller than the $k = 10$. This makes it impossible for this LRC to be a member of the MDS code family. This contradiction shows that d_{LRC} must be smaller than 6. Thus, we determine that $d_{\text{LRC}} = 5$ is indeed the maximum distance possible for the created $(10, 5, 7)$ Locally Repairable Code.

4.2 The (10, 3, 4) Locally Repairable Code

The next parameter set will be based on a different Reed-Solomon code. To further improve the storage overhead in comparison to the previous Locally Repairable Codes, we will start with the $(10, 1)$ RS code as precode. Based on this precode, we will introduce 3 parities as shown in Figure 9. With the y now equal to 3, the Requirement 4.4 becomes

$$s_{\min} + y - 2 = 4 + 3 - 2 = 5 \leq 5 = s_{\max}.$$

Therefore it holds. Using Properties 4.1 and 4.5 we know that we can imply parity block $Q_{\min}$. Since in this case, S_3 is the smallest subset, $Q_{\min} = Q_3$. Because the requirement holds, the locality of the code will stay the same after the implication of parity block Q_3.

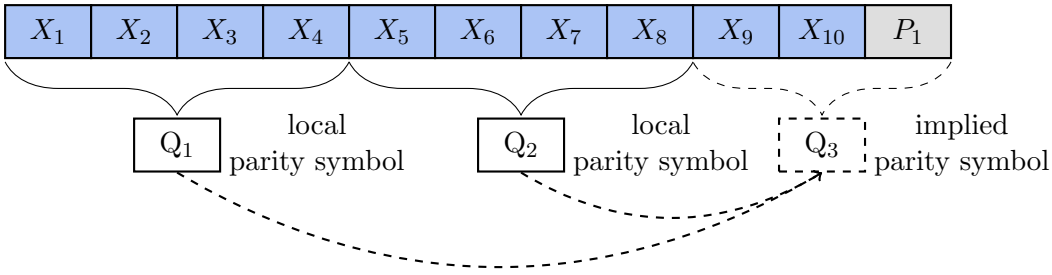


Figure 9: Locally Repairable Code of size 13, distance 2 and locality 4.

Next, we would like to establish the code distance d_{LRC} of our newly created Locally Repairable Code. With the precode being a $(10, 1)$ RS code with $d_{\text{RS}} = n-k+1 = 2$, we establish using Property 3.3, that $d_{\text{LRC}} \geq 2$. We will now prove that $d_{\text{LRC}} = 2$ is the maximum distance possible for a length 13 code that has block locality 4. To prove so, we will introduce the following theorem [9].

Theorem 4.9. *Let $\mathcal{C}$ be a linear $[k, n-k, d]_q$-code of information locality r. The minimum code distance of $\mathcal{C}$ is bounded by $d \leq n - \left\lceil \frac{k}{r} \right\rceil - k + 2$.*

Using the linearity of the LRC family, we derive that the distance is bounded by $d_{\text{LRC}} \leq 13 - \left\lceil \frac{10}{4} \right\rceil - 10 + 2 = 2$. Inferring from the inclusion $2 \leq d_{\text{LRC}} \leq 2$, we determine that $d_{\text{LRC}} = 2$ is indeed the maximum distance possible for the created $(10, 3, 4)$ Locally Repairable Code.

4.3 The (10, 6, 3) Locally Repairable Code

As a last attempt to find an alternative Locally Repairable Code, we use a $(10, 2)$ Reed-Solomon code as a basis. Here we add 4 local parities equally divided over the 12 coded blocks. In this case, all created subsets are of equal size (see Figure 11). This leads to an equal maximum and minimum subset size, i.e. $s_{\max} = s_{\min}$. Considering the following equation, we establish that Requirement 4.4 does not hold.

$$s_{\min} + y - 2 = 4 + 4 - 2 = 6 \nleq 4 = s_{\max}$$

We will show what happens if one implies one of the parities anyhow. Even though the requirement does not hold, we know from Property 4.1, that an implication is still possible. E.g., if one removes parity block Q_1 from subset S_1, then S_1 will gain all other local parities. Where before the subset only contained the members $S_1 = \{X_1, X_2, X_3, Q_1\}$, the implication will enlarge the set to the members $S_1' = \{X_1, X_2, X_3, Q_2, Q_3, Q_4\}$. The implication is depicted below and can be compared to Figure 11. The result will be repair sets of size 5 for some members of S_1'. Take look at block X_2 for example. After the implication it received the repair set $\mathcal{R}_{X_2} = \{X_1, X_3, Q_2, Q_3, Q_4\}$. From Corollary 2.8.1, we know that the size of the repair sets determine the locality of a code. We know therefore that this implication enlarged the locality significantly from $r = 3$ to $r = 5$.

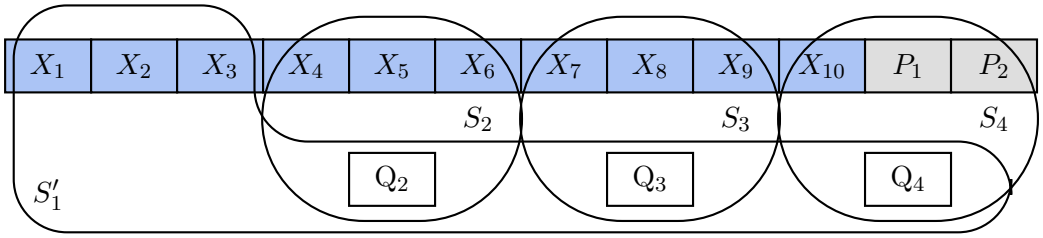


Figure 10: Locally Repairable Code of size 15 and locality 5.

Because of the negative effect of removing a parity, we will store all of the blocks locally in our new code. This creates the code formation with locality $r = 3$ as shown in the figure below.

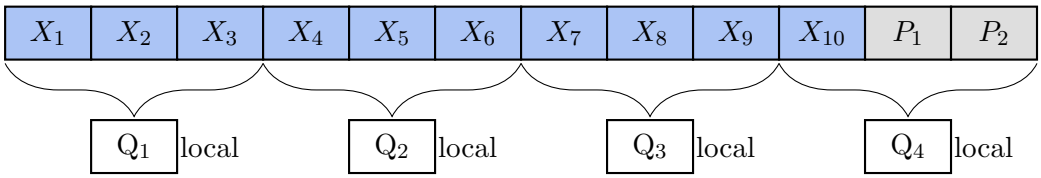


Figure 11: Locally Repairable Code of size 16, distance 3 and locality 3.

Next, we would like to establish the code distance d_{LRC} of our newly created Locally Repairable Code. With the precode being a $(10, 2)$ RS code with $d_{\text{RS}} = n-k+1 = 3$, we establish using Property 3.3, that $d_{\text{LRC}} \geq 3$. We will now prove that $d_{\text{LRC}} = 3$ is the maximum distance possible for a length 16 code that has block locality 4. To prove so, we will introduce the following theorem [12].

Theorem 4.10. *Let $\mathcal{C}$ be an $(k, n-k, r)$ Locally Repairable Code with t disjoint repair sets of size $|\mathcal{R}|$. Then the minimum distance of $\mathcal{C}$ is bounded above as follows:*

$$d \leq n - \sum_{i=0}^{t} \left\lfloor \frac{k-1}{|\mathcal{R}|^i} \right\rfloor.$$

To use this theorem on the parameter set of our created Locally Repairable Code, we need to determine the variables $|\mathcal{R}|$ and t. We will start with the first, and zoom in on $\mathcal{R}$ and its size $|\mathcal{R}|$. This variable overarches all subsets $\mathcal{R}_j$, which serve as the repair set for each element j in the LRC depicted in Figure 11. As stated earlier, the subsets S_1 through S_4 are all of size 4. Deriving from Equation 7, used in the proof of Property 4.5, we know $|\mathcal{R}_j| = |S_i| - 1$ for all blocks $j \in S_i$, where $i \in \{1, \ldots, y = 4\}$. Now, since $|S_i| = 4,\ \forall i \in \{1, \ldots, 4\}$, $|\mathcal{R}_j| = 4 - 1 = 3$ for all blocks j in the LRC given in Figure 11. Since parameter $\mathcal{R}$ is the overarching variable of these repair sets, its size $|\mathcal{R}| = 3$ as well.

Next, we take a look at the second variable, t. It indicates the number of disjoint repair sets. To find this amount, note that the repair sets $\mathcal{R}_j$ for elements within the same set S_i, will contain similar elements. To show this, take a look at $\mathcal{R}_{X_4} = \{X_5, X_6, Q_2\}$ for example. If we take another element within this same set S_2, e.g. X_6, we find $\mathcal{R}_{X_6} = \{X_4, X_5, Q_2\}$. To see if these sets are disjoint, we check whether their intersection is empty. The equation below shows that this is clearly not the case.

$$\mathcal{R}_{X_4} \cap \mathcal{R}_{X_6} = \{X_5, Q_2\} \neq \varnothing$$

As it turns out we can only find non-intersecting repair sets if we take elements from different subsets S_i. Since there are only $y = 4$ subsets, we will only be able to find $t = 4$ disjoint repair sets.

Using the Theorem and the parameters determined above, we find that the distance of our Locally Repairable Code will yield a bound. This bound is calculated below.

$$\begin{aligned} d_{\text{LRC}} &\leq n - \sum_{i=0}^{t} \left\lfloor \frac{k-1}{|\mathcal{R}|^i} \right\rfloor = 16 - \sum_{i=0}^{4} \left\lfloor \frac{10-1}{3^i} \right\rfloor \\ &= 16 - \left(\left\lfloor \frac{10-1}{3^0} \right\rfloor + \left\lfloor \frac{10-1}{3^1} \right\rfloor + \left\lfloor \frac{10-1}{3^2} \right\rfloor + \left\lfloor \frac{10-1}{3^3} \right\rfloor \right) \\ &= 16 - \left(\lfloor 9 \rfloor + \lfloor 3 \rfloor + \lfloor 1 \rfloor + \left\lfloor \frac{1}{3} \right\rfloor + \left\lfloor \frac{1}{9} \right\rfloor \right) = 3 \end{aligned} \tag{8}$$

Deriving from the lower bound given by the precode, and the upperbound given in Equation 8, we determine that distance $d_{\mathrm{LRC}} = 3$ is indeed maximum for the created $(10, 6, 3)$ Locally Repairable Code.

5 Performance

To bring all information from the previous section together, we will create a clear overview of the performance of each of the Locally Repairable Codes. We will go over each of the codes, and try to benefit from the advantages of each. We will do so, by trying to find useful applications per LRC. The 3-replication and (10,4) Reed-Solomon coding schemes are also included. This way, comparisons can be made more easily.

In Table 2, we go over all the erasure coding schemes we have crossed. The variables that have been focused on are the storage overhead, locality and distance of each of the code. These characteristics differentiate the variation of codes. On first sight, one can note that the 3-replication has an extremely large overhead. For every information symbol, two non-information symbols are in place to back up the information in the case of an erasure. The reason that this coding scheme is still used nowadays, is due to its favorably small code locality. Only one symbols needs to be visited in order to recover an erasure. If we want to replace the 3-replication coding scheme, we have to come up with schemes that deliver codes with a small code locality as well. The main focus of the thesis was to find coding schemes that could tackle the massive amounts of storage overhead. Lowering the storage overhead of a scheme however, leads to an increased locality. The Reed-Solomon code is a clear example of this process. Note in Table 2 that the improved storage overhead is followed by a significant increase in code locality. So much even, that it exceeds the locality of the 3-replication scheme by ten times. In short, we have given ourselves the task to find a code that scores well in cutting storage overhead, while remaining low on locality.

Table 2: Comparing the characteristics of erasure coding schemes.

Erasure Coding	*Precode*	*Storage Overhead*	*Locality*	*Distance*
3-Replication	-	2×	1	3
(10, 4) RS	-	0.4×	10	5
(10, 6, 5) LRC	(10,4) RS	0.6×	5	5
(10, 5, 7) LRC	(10,4) RS	0.5×	7	5
(10, 3, 4) LRC	(10,1) RS	0.3×	4	2
(10, 6, 3) LRC	(10,2) RS	0.6×	3	3

The goal of finding an optimum in the trade-off between overhead and locality, was one that drove the research addressed in Section 3.3. It was shown how the Locally Repairable Codes, built upon the Maximum Distance Separable codes, are a great attempt in finding an optimum here. While maintaining a large code distance, the $(10, 6, 5)$ LRC introduced a locality of 5 while greatly reducing the storage overhead to 0.6× compared to the 3-replication scheme. To extend this research, we dove deeper into the characteristics of these LRCs. We tried out different parameter sets, and managed to create three different ones (Section 4.1 through 4.3). Each of them having its own advantages and disadvantages, as will be clarified next.

5.1 Improving Storage Overhead

The main issue we tried to tackle within this thesis, is the vastly growing amounts of storage overhead in data storage systems nowadays. A way to do so, is by finding improved coding schemes used within these systems. The main improvement of these schemes should then be focused around their storage overhead without losing its recoverability. This trade-off was a constant notion when creating the codes explained below.

The $(10, 5, 7)$ Locally Repairable Code was our first attempt in tackling this issue. With this LRC, we were able to reach an overhead that is almost equal to the overhead of its Reed-Solomon precode. The disadvantage of the RS codes being its locality, this $(10, 5, 7)$ LRC improves on this characteristic. Since the locality of this code is not completely favorable yet, the scheme could be applied to cold data sets; data that is not visited often. Since not many erasures will occur within these sets, little repair traffic will take place. For these sets, the code locality will not jeopardize the functionality of the code. This LRC provides for the same data reliability as the $(10, 6, 5)$ LRC found by the researchers, while maintaining less amounts of storage. Nevertheless, our next task was to look for codes that, while remaining low on overhead, provide a more favorable locality.

5.2 Improving Locality

The following and last two parameter sets created Locally Repairable Codes with locality closer to the one of the 3-replication code. This, without gaining much storage overhead. Since we had no limitations in using only the $(10, 4)$ Reed-Solomon precode, we explored the possibilities further. To decrease the storage overhead, it would only be logical to start with precodes containing less parity symbols. The $(10, 1)$ Reed-Solomon code, with just one parity symbol, could introduce a good basis towards such a code. As visible in Table 2, the $(10, 3, 4)$ LRC that followed from this $(10, 1)$ RS, has an utmost optimal storage overhead indeed. Moreover, focusing on its locality being only 4, this code has made large improvements on characteristics so far. However, these improvements come with a price. This becomes clear when looking into the last characteristic: the code distance. Losing distance between code words disables recovery of multiple erasures. Without this ability, data could get lost more easily. Because data reliability is number one priority when it comes to data storage, coding schemes need to be able to provide this. Since the $(10, 3, 4)$ LRC only provides for a code distance of 2, it can recover just one erasure before losing data. Nevertheless, this might be enough for some cold data applications. These data sets would have so little input/output activity that erasures seldom take place. If an erasure does take place, it would be preferred if these are recovered quickly. The longer it takes for data to recover, the larger the chances on a second erasure before recovery. Because of the favorable locality of the $(10, 3, 4)$ LRC, its usage might be preferred over the usage of other codes with the same code distance, e.g. the $(10, 1)$ RS code.

In the case of applications that asks for an erasure coding with greater recoverability, a different parameter set should be used. To ensure a distance that could at least solve two erasures, we grounded our final Locally Repairable Code upon the $(10, 2)$ Reed-Solomon code. The $(10, 6, 3)$ LRC that followed from this $(10, 2)$ precode, has a promising locality of 3. A disadvantage of this LRC however, is the fact that all the parities needed to be added locally to the code. As shown in Section 4.3, the implication of a parity block of the $(10, 6, 3)$ LRC was possible, but would greatly increase the locality of the code. What we are therefore left with is a code with a storage overhead equal to the one of the researchers. The coding scheme could serve for more hot data related applications. For example, applications in which the 3-replication code are dominating right now. With a similar locality and distance and an improved storage overhead compared to these replication codes, the $(10, 6, 3)$ LRC could be an interesting substitute to look into.

6 Conclusion

The challenge we took up in this thesis, was to find coding schemes that would find optima in the trade-off between storage overhead and code locality. This while keeping a code distance great enough to provide the requested data reliability. In an attempt to find such an optimum, researchers experiment with existing coding schemes to try and create improved ones. The research group in the paper "XORing Elephants: Novel Erasure Codes for Big Data" used Maximum Distance Separable (MDS) code as base, to create their so-called Locally Repairable Code (LRC) [4]. In Section 2.3, the advantages and disadvantages related to the MDS code family are explained. The Reed-Solomon codes (Section 3), are members of this family, and were used by the researchers to create their $(10, 6, 5)$ LRC. The variables in this notation respectively stand for the number of information symbols (k), the total number of parity symbols ($n - k$) and the locality (r) of the code. In Section 2, we saw that n stands for the total size of the code, and that this had to be larger than the number of informational symbols to enable the erasure recoverability of a code. The non-informational parity symbol(s) can namely be combined with the informational symbols to recover erasures within a system. The distance (d) of a code expresses the exact number of erasures the code can recover ($d - 1$). All these characteristics tell something about the performance of a code. The goal of this thesis was to find a code that performed optimally in the trade-off between overhead and code locality. The researchers defined a fine playing field for finding new coding schemes that perform well within this trade-off.

In Table 3, an overview is given on the Locally Repairable Codes that were described throughout the thesis. A score is given to each of the characteristics, to make comparison easier. The first thing that one might notice are the great disadvantages of using the 3-replication scheme over the others. Even considering its optimal locality, the code yields a 200% storage overhead, which was exactly to be overcome. Nevertheless, this code is still used in data storage systems because of his favorable locality.

Table 3: Comparing the performance of erasure coding schemes.

Erasure Coding	*Storage Overhead*	*Locality*	*Distance*
3-Replication	$----$	$+++$	$-$
(10, 4) RS	$+$	$---$	$+$
(10, 6, 5) LRC	$-$	$\pm$	$+$
(10, 5, 7) LRC	$\pm$	$-$	$+$
(10, 3, 4) LRC	$++$	$+$	$--$
(10, 6, 3) LRC	$-$	$++$	$-$

Now it is the turn of comparing the performance of the more complicated coding schemes. The Reed-Solomon code shows to be quite disadvantageous considering its locality. The unfavorable locality makes it almost unusable for erasure coding applications. Recovery of an erasure simply takes up to much of the bandwidth of a storage system. This family of codes do however serve as a useful basis for the Locally Repairable Codes. These codes all score well at different characteristics. It completely depends on the needs of an application which code would be best to use. For the storage overhead they all have a great advantage over the replication scheme. With these proposed codes, we have bravely attempted to tackle the issue of the enormous amounts of storage overhead in storage systems nowadays.

7 Future Work

Although several characteristics were kept into account while creating the codes, there is one that was left out. For companies it is not only priority that data is secured and quickly accessible, but also that storage is reliable over long periods of time. During which, nodes might become individually unreliable, leading to data loss. The time that a coding scheme stayes reliable differs from case to case. For this uncertainty, a Mean Time To Data Loss (MTTDL) measure is defined (see Definition 7.1). It equals the time it takes for a given storage system to exhibit enough failures such that at least one block of data cannot be retrieved or reconstructed [13].

Definition 7.1 (MTTDL)**.** *A Mean Time To Data Loss is a measure of the reliability of a system defined as*

$$MTTDL = \int_0^\infty R(t)dt,$$

in which $R(t) = P(T_{FF} > t)$ *represents the probability that a system provides an uninterrupted service during a certain time interval* $[0, t]$*.*

For future work, it might be interesting to take the MTTDL into account when comparing the different coding schemes. This might be a way to distinguish the performance of the Locally Repairable Codes even better, or lead to other interesting results.

Lastly, the search to finding new code schemes is far from over. The list of codes found in Section 4 can easily be extended by starting with other precodes or using the methods of shortening and extending (see Section 3.2) differently. By means of this thesis, an image has hopefully been drawn on the endless resources available on tackling the initial challenge: finding an optimum in the trade-off between storage overhead and code locality in erasure coding.

References

[1] N. Nagaraj, "Lossless data compression with error detection using Cantor set," *arXiv e-prints*, p. arXiv:1308.2299, Aug 2013.

[2] D. H. C. Du, "Recent advancements and future challenges of storage systems," *Proceedings of the IEEE*, vol. 96, pp. 1875–1886, Nov 2008.

[3] M. Xia, M. Saxena, M. Blaum, and D. A. Pease, "A tale of two erasure codes in HDFS," in *13th USENIX Conference on File and Storage Technologies (FAST 15)*, (Santa Clara, CA), pp. 213–226, USENIX Association, Feb 2015.

[4] M. Sathiamoorthy, M. Asteris, D. Papailiopoulos, A. Dimakis, R. Vadali, S. Chen, and D. Borthakur, "XORing elephants: Novel erasure codes for big data," vol. 6, Aug 2013.

[5] O. Khan, R. Burns, J. Plank, W. Pierce, and C. Huang, "Rethinking erasure codes for cloud file systems: minimizing I/O for recovery and degraded reads," in *10th USENIX Conference on File and Storage Technologies (FAST 12)*, pp. 20–20, 2012.

[6] K. V. Rashmi, N. B. Shah, D. Gu, H. Kuang, D. Borthakur, and K. Ramchandran, "A solution to the network challenges of data recovery in erasure-coded distributed storage systems: A study on the Facebook warehouse cluster," in *Presented as part of the 5th USENIX Workshop on Hot Topics in Storage and File Systems*, (San Jose, CA), USENIX, Jun 2013.

[7] R. Miller, "Facebook builds exabyte data centers for cold storage," Jan 2013. Last consulted on 20-12-2018 via: `https://www.datacenterknowledge.com/archives/2013/01/18/facebook-builds-new-data-centers-for-cold-storage`.

[8] W. Spee, "Comparing various locality approaches for codes repairing two erasures," *Proceedings of the TU Delft, Applied Mathematics BSc Thesis*, Jun 2018.

[9] P. Gopalan, C. Huang, H. Simitci, and S. Yekhanin, "On the locality of codeword symbols," *CoRR*, vol. abs/1106.3625, Jun 2011.

[10] M. Fayers, *Lecture Notes on Coding Theory*. School of Mathematical Sciences Queen Mary, University of London, Mar 2008.

[11] I. Reed and G. Solomon, "Polynomial codes over certain finite fields," *Journal of the Society for Industrial and Applied Mathematics*, vol. 8, pp. 300–304, Jun 1960.

[12] I. Tamo, A. Barg, and A. A. Frolov, "Bounds on the parameters of locally recoverable codes," *CoRR*, vol. abs/1506.07196, Mar 2015.

[13] K. Kralevska, "Applied erasure coding in networks and distributed storage," *CoRR*, vol. abs/1803.01358, Mar 2018.

Appendices

Appendix A Shortening

To give an explanation on the way shortening works, consider the following example. In this example we will create a Reed-Solomon code $\mathcal{C}'$ with the parameters $k' = 10$ and $n' = 14$. These parameters ensure a code distance $d' = n' - k' + 1 = 14 - 10 + 1 = 5$. To create this $(10, 4)$ RS code, shortening is needed, because the code is not of the usual parameter form $(k, 2^m - 1 - k)$. Nevertheless, it is possible to create an RS code of this odd form. To do so, we need to make use of a mothercode $\mathcal{C}$ of length n greater than n' for which $\exists m \in \mathbb{N},\ 2^m - 1 = n$. In our example, the RS code of length $n = 15 = 2^4 - 1$ would be an excellent choice. This would however create a length difference s of $n - n' = 15 - 14 = 1 = s$ between our desired code and the mothercode. To create the same difference in message length, the k of our mothercode will be equal to $k = k' + s = 10 + 1 = 11$. We will thus use an $(11, 4)$ RS code with corresponding (15×11) generator matrix $\mathbf{G}$ to create our desired code $\mathcal{C}'$. The distance d that this parameter set entails, equals $d = n - k + 1 = 15 - 11 + 1 = 5 = d'$. We therefore want to make sure that our desired code has the same code distance as the $(11, 4)$ RS mothercode $\mathcal{C}$. The generator matrix $\mathbf{G}$ of the mothercode will look as follows in standard form.

$$\mathbf{G} = \big(\mathbf{I}_k \mid \mathbf{A}_{k\times(n-k)}\big) = (\mathbf{I}_{11} \mid \mathbf{A}_{11\times 4}) = \left(\begin{array}{cccc|cccc} 1 & 0 & \cdots & 0 & a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ 0 & 1 & \cdots & 0 & a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ \vdots & & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & a_{11,1} & a_{11,2} & a_{11,3} & a_{11,4} \end{array}\right)$$

This generator matrix is composed of the vectors as depicted in Equation 9. These vectors form the basis of the $(11, 4)$ RS mothercode, which are referred to by variables $\mathbf{g}_1$ through $\mathbf{g}_k$. We know from Equation 2, that we can form all code words $c \in \mathcal{C}$ by a linear combinations of those vectors.

$$\left\{ \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ a_{1,1} \\ \vdots \\ a_{1,4} \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \\ a_{2,1} \\ \vdots \\ a_{2,4} \end{bmatrix}, \ldots, \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ a_{11,1} \\ \vdots \\ a_{11,4} \end{bmatrix} \right\} = \{\mathbf{g}_1, \mathbf{g}_2, \ldots, \mathbf{g}_k\} \tag{9}$$

To create our desired code $\mathcal{C}'$, we need to decrease both the message length (k) as well as the code length (n) of the mothercode $\mathcal{C}$ by $s = 1$. We will start with the message length, by removing one of the vectors from the basis above, e.g. $\mathbf{g}_1$. We are now left with the vectors $\mathbf{g}_2$ through $\mathbf{g}_k$. These vectors form a basis for a code with $k' = k - 1 = 11 - 1 = 10$. This is exactly the k' that we wanted to achieve for code $\mathcal{C}'$. But how do we know that the original distance d of our mothercode hasn't decreased by the removal of one vector from the basis? By getting rid of one vector, we have cut out some of the original code words of the mothercode $\mathcal{C}$. Removing some of the code words can never decrease the distance between the remaining ones. On the contrary, the new distance d' might even have increased after the removal. Therefore, we know that the distance of this new code d' is at least the same size as d, i.e. $d' \geq d$.

The last step to get to the desired code $\mathcal{C}'$, is to reduce the code length n as well, but without losing any distance. To be able to do so, note that the remaining basis vectors all start with the symbol 0 (see Equation 10). This means that all words formed by a linear combination of these

vectors, will have a first symbol equal to 0. Removing this 0 at the start of each code word, does not affect the distance between the words, but does decrease the length of the words by one. This is exactly what we wanted to do, in order to create code $\mathcal{C}'$ of length $n' = n - 1 = 14$.

$$\{\mathbf{g}_2, \mathbf{g}_3, \ldots, \mathbf{g}_k\} = \left\{ \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \\ a_{2,1} \\ \vdots \\ a_{2,4} \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ 0 \\ a_{3,1} \\ \vdots \\ a_{3,4} \end{bmatrix}, \ldots, \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \\ a_{11,1} \\ \vdots \\ a_{11,4} \end{bmatrix} \right\} \tag{10}$$

Combining all the previous information, we know that the basis depicted in Equation 11 has the following characteristics:

- $d' \geq 5$ (distance);
- $k' = 10$ (number of information symbols);
- $n' = 14$ (length of codewords).

$$\left\{ \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ a_{2,1} \\ \vdots \\ a_{2,4} \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \\ a_{3,1} \\ \vdots \\ a_{3,4} \end{bmatrix}, \ldots, \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ a_{11,1} \\ \vdots \\ a_{11,4} \end{bmatrix} \right\} \tag{11}$$

Since the distance of a code is at most $n - k + 1$ we know that

$$5 \leq d' \leq n' - k' + 1 = 14 - 10 + 1 = 5 \Longrightarrow d' = 5.$$

Thus, the code we created with the generator matrix $\mathbf{G}'$ given below, is indeed a $(10, 4)$ Reed-Solomon code.

$$\mathbf{G}' = \left(\begin{array}{ccc|cccc} 1 & \cdots & 0 & a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & 1 & a_{11,1} & a_{11,2} & a_{11,3} & a_{11,4} \end{array} \right) = (\mathbf{I}_{10} \mid \mathbf{A}_{10\times 4}) = \left(\mathbf{I}_{k'} \mid \mathbf{A}_{k'\times(n'-k')}\right)$$

Appendix B Proofs

The first part of the appendices is dedicated to the proof of Property 4.1. Mathematicians have already used this property in the paper "XORing Elephants: Novel Erasure Codes for Big Data", but proved it only for their parameter set [4]. To be able to use this property on a larger scale, it is necessary to generate a more general proof.

Property B.1 (Parity Alignment)**.** *Let $\mathcal{L}$ be a Locally Repairable Code. Let blocks Q_i be the y parities created by the method of extending the Reed-Solomon precode of $\mathcal{L}$. Within $\mathcal{L}$, each block Q_i is implied by a linear combination of the other parity blocks Q_j $(j \neq i)$ of $\mathcal{L}$.*

Proof. To proof this property, we will first go over the process of creating the parity blocks $\{Q_1, Q_2, \ldots, Q_y\}$, with $Q_i \in \mathbb{F}_q$ of $\mathcal{L}$. First, the $(10, n-10)$ Reed-Solomon precode of $\mathcal{L}$ is separated into y subsets. We know that $k = 10$, because that was a condition we enforced on finding new Locally Repairable Codes. For this variable y, any number below n is possible, with n being the length of the precode. The number of parity blocks that will be added to the precode to create a Locally Repairable Code $\mathcal{L}$, is equal to this chosen y.

Example B.2 (y)**.** *In the figure below, a $(10, n-10)$ Reed-Solomon code is schematically depicted as the blocks X_1 through P_{n-k} to form a code of length n. The coding scheme is then used to create a Locally Repairable Code with $y = 3$, by separating it into three subsets.*

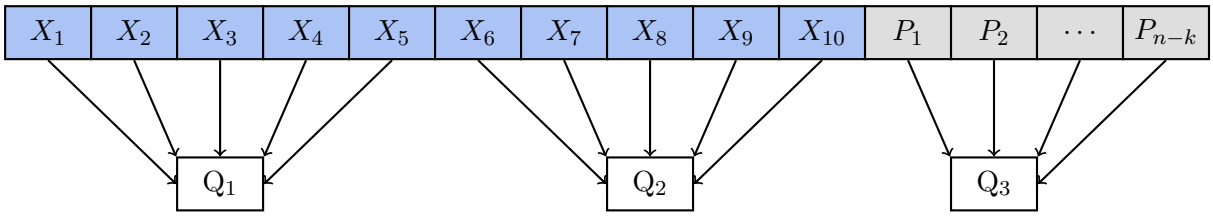


Figure 12: The $(10, n-10)$ Reed-Solomon precode separated by three parity blocks.

To distinguish the different parity blocks, we will introduce the variables x_i, $i \in [y]$ with $0 = x_0 < x_1 < \ldots < x_y = n$. These variables refer to the positions on which the Reed-Solomon code is split up for each parity block.

Example B.3 (x_i)**.** *In Figure 12, the $(10, n-10)$ Reed-Solomon code is split up into three parity blocks $\{Q_1, Q_2, Q_3\}$. The positions x_0 through x_3 on which the code is split, are equal to $x_0 = 0$, $x_1 = 5$, $x_2 = 10$ and $x_3 = n$. Q_1 now depends on the blocks of positions $x_0 + 1$ through x_1, Q_2 on those of positions $x_1 + 1$ through x_2 and Q_3 on those of positions $x_2 + 1$ through x_3.*

A parity being dependent on a set of blocks means that this parity is formed by a linear combination of these blocks. This can for example be seen in parity Q_2, which is created by the linear combination

$$Q_2 = c_6 X_6 + c_7 X_7 + c_8 X_8 + c_9 X_9 + c_{10} X_{10},$$

where c_j are elements of the field $\mathbb{F}_q$ of $\mathcal{L}$.

To clearly show that the alignment equation holds for the parity blocks, we will define the generator matrix of our $(10, n-10)$ Reed-Solomon precode $\mathbf{G}_{(10\times n)} = \begin{pmatrix} \mathbf{g}_1 & \mathbf{g}_2 & \ldots & \mathbf{g}_n \end{pmatrix}$, where $\mathbf{g}_j$ denotes the j-th column of $\mathbf{G}$ for $j \in [n]$. After selecting the variables x_i, $i \in [y]$, each parity block Q_i can be depicted as a summation of the columns of $\mathbf{G}$. When adding the parity blocks to the generator matrix as vectors $\mathbf{Q_i}$, they yield the following equation

$$\mathbf{Q}_{i+1} = \sum_{j=x_i+1}^{x_{i+1}} \mathbf{g}_j, \text{ with } i \in \{0, \ldots, y-1\}. \tag{12}$$

Example B.4. *Take another look at Figure 12 and Example B.3. With the chosen variables x_i for $i \in [y]$, we know that e.g. Q_2 is a linear combinations of the blocks $x_1 + 1 = 6$ through $x_2 = 10$. This linearity is expressed within the generator matrix $\mathbf{G}$ as shown in Equation 12:*

$$\mathbf{Q}_2 = \sum_{j=6}^{10} \mathbf{g}_j.$$

Adding these parity blocks to the original matrix $\mathbf{G}$ will yield the $(10 \times (n+y))$ generator matrix of $\mathcal{L}$: $\mathbf{G}_\mathcal{L} = \begin{pmatrix}\mathbf{G} \mid \mathbf{Q}_1 & \ldots & \mathbf{Q}_y\end{pmatrix}$. To proof the property of the alignment equation, we will convert the matrix $\mathbf{G}_\mathcal{L}$ to standard form, if it is not already. To do so we need to apply a full-rank transformation on the rows of $\mathbf{G}_\mathcal{L}$ in the following way: $\mathbf{G}'_\mathcal{L} = \mathbf{A}\mathbf{G}_\mathcal{L} = \mathbf{A}\begin{pmatrix}\mathbf{G}_{:,1:10} & \mathbf{G}_{:,11:n}\end{pmatrix} = \begin{pmatrix}\mathbf{I}_{10} & \mathbf{A}\mathbf{G}_{:,11:n}\end{pmatrix}$, where $\mathbf{A} = \mathbf{G}^{-1}_{:,1:10}$ and $\mathbf{G}_{:,x:y}$ is a submatrix of $\mathbf{G}_\mathcal{L}$ that consists of columns with indices from x to y. According to Lemma 2.11, this transformation retains the distance and locality properties of original generator matrix $\mathbf{G}_\mathcal{L}$ [4].

In Lemma 3.2 we have seen that the parity check matrix of a Reed-Solomon code is given by $[\mathbf{H}]_{i,j} = a_{j-1}^{i-1}$, where $a_0, a_1, \ldots, a_{n-1}$ are n distinct elements of the field $\mathbb{F}_q$, for $i \in [k]$ and $j \in [n]$. Because of this equation, the matrix contains the row $[\mathbf{H}]_{1,j} = \begin{bmatrix}a_0^0 & a_1^0 & \ldots & a_{n-1}^0\end{bmatrix} = \mathbf{1}^\intercal$. Due to Property 2.13, the vector $\mathbf{1}$ of length n therefore has to be orthogonal to the generator matrix $\mathbf{G}$ of our precode, i.e., $\mathbf{G}\mathbf{1}^T = \mathbf{0}_{k\times 1}$. This means that for the columns $\mathbf{g}_j$ of generator matrix $\mathbf{G}$

$$\begin{aligned}
\mathbf{G}\mathbf{1}^T = \mathbf{0}_{k\times 1} &\Leftrightarrow \sum_{j=1}^{n} \mathbf{g}_j = \mathbf{0}_{k\times 1} \\
&\Leftrightarrow \sum_{i=0}^{y-1} \sum_{j=x_i+1}^{x_{i+1}} \mathbf{g}_j = \mathbf{0}_{k\times 1} \\
&\Leftrightarrow \sum_{i=0}^{y-1} \mathbf{Q}_{i+1} = \mathbf{0}_{k\times 1}.
\end{aligned}$$

With the last step and "-" becoming "+" due to the binary extended field, it becomes clear that

$$\mathbf{Q}_i = \sum_{j=1}^{i-1} \mathbf{Q}_j + \sum_{j=i+1}^{y} \mathbf{Q}_j, \forall i \in \{1, \ldots, y\}.$$

Since the alignment equation holds for the parity vector blocks of the generator matrix $\mathbf{G}_\mathcal{L}$, it is safe to assume that each block Q_i is implied by a linear combination of the other parity blocks Q_j $(j \neq i)$ of $\mathcal{L}$ [4].

□