A Fault-Tolerant RAID-like System with List Decodable Codes

Ming Yu Liu

August 19, 2004

A Fault-Tolerant RAID-like System with List Decodable Codes

Ming Yu Liu

Abstract

With list decoding of error-correcting codes, we can correct errors beyond the traditional "error-correction radius". The advantage is that the transmitted message can suffer from more errors caused by the noise in the communication channel. But after we perform the list-decoding algorithm on the received word, we get a list of codewords, and still don't know which is the correct one. Codes that have list-decoding algorithm are called list-decodable codes.

In the thesis, we will use the list-decodable codes to build a RAID-like system with high fault tolerance, for example, more than half the system is faulty. That is, we can safeguard a document in the system, even when more than half the system are failure. We will also bring up some experimental results about our system.

Keywords: Coding theory, List-decoding, Reed-Solomon codes, RAID, Fingerprint

Contents

List of Figures

List of Tables

Chapter 1

Introduction

When we transmit a message in a communication channel, some errors caused by the noise in the channel may occur such that we receive a different message on the other side. To ensure that the receiver can recover the original message from the received information, some redundant information is required. Coding theory deals with such problem with efficiency in the time of encoding/decoding and the space of redundant information.

In coding theory, the distance $d(\mathcal{C})$ of a code $\mathcal{C}$ is the minimum *Hamming distance* over all pairs of non-identical codewords in $\mathcal{C}$. It is the key factor in deciding the error-correcting capability of a code $\mathcal{C}$ since a conventional decoder can correct up to $\lfloor \frac{d(\mathcal{C})-1}{2} \rfloor$ errors. Within this error-correcting bound, the decoder can find at most one codeword $c \in \mathcal{C}$ such that $d(c, r) \leq \lfloor \frac{d(\mathcal{C})-1}{2} \rfloor$ where $d(c, r)$ is the distance between the codeword c and the received word r. So such decoding problem is called unique-decoding. Another notion called list-decoding, which was introduced independently by Elias[1] and Wozencraft[16] in the late 1950s, deals with the problem that the number of errors is beyond the traditional "error-correction radius" (i.e. half the minimum distance). A list-decoder will output all of the possible codewords within a specified distance with the received word. Until now, the list-decoding algorithms for some error-correcting codes are invented and codes that have efficient list-decoding algorithms are called *list-decodable codes*.

The list decoding algorithm of Reed-Solomon codes was shown by Madhu Sudan[13] in 1997. It inspires the research in this area. Then, the list decoding algorithms for other well-known codes, such as Reed-Muller codes[10], Chinese remainder codes[6], algebraic-geometric codes[5], concatenated codes[4] and algebraic soft decoding[8] were invented. There was also a new combinatorial approach to list decoding resulting a linear time encodable and list-decodable codes by Venkatesan Guruswami and Piotr Indyk[7] in 2003.

Same idea in coding theory can be applied to applications in data storage. Plank[11] gave a tutorial on how to use RS code to build a RAID-like System with unique-decoding. Teng et al. [14] used the extractor codes to build a EC-RAID system, which offers high fault tolerance, while there is no specific implementation.

In this thesis, we show an architecture of RAID-like system by using the *list-decodable codes*. The RAID-like system can provide higher fault tolerance and reliability due to the nice property of list-decoding. We not only describe the full details about how to store and retrieve a document in our system, but also build a simulation environment in Java programming language so that we can demonstrate the reliability of the storage system with visualized testing. Our RAID-like simulation system through the web can be accessed via http://www.csie.nctu.edu.tw/∼myliu/java2/.

Since a list-decoder can correct more errors than a traditional unique-decoder, it usually takes more time. But if we have other useful information (e.g. erasures), we can reduce the decoding time. Similar information are found when we try to reconstruct the source document in our system. The information can be used not only to speedup the decoding time, but also to filter impossible decoding results. So the *retrieve* function provided by our system is more efficient than just using a list-decoder directly.

As pointed by McEliece[9], we find similar results in our experiments that the list size is almost unique. Even so we have used the fingerprint to help the list decoding algorithm reconstruct the unique answer from the lists. Our system can be used to develop new list decoding techniques. With the object oriented nature of Java, users can design and test their new coding methods on our system.

The organization of the rest of this thesis is as follows. Chapter 2 introduces the list-decodable codes we used. Chapter 3 shows an application in data storage with the list-decodable codes. Chapter 4 gives some experimental results on the codes and the system. At last, we make a conclusion in chapter 5.

Chapter 2

List-decodable codes

In this thesis, we use the most widely used Reed-Solomon codes to be our list-decodable code. So we will give a detail introduction of the code, including unique-decoding, list-decoding and decoding with erasures.

2.1 Reed-Solomon codes

Reed-Solomon code is one of the most practical and well-known error-correcting codes. They were introduced by Irving S. Reed and Gustave Solomon[12]. Until now, they have a wide range of applications in digital communication and data storage. We define the codes first.

Definition 2.1.1 (Reed-Solomon codes) *Let $\Sigma = \mathbb{F}_q$ be a finite field and $\alpha_1, \ldots, \alpha_n$ be distinct elements of $\mathbb{F}_q$. Given n, k and $\mathbb{F}_q$ such that $k \leq n \leq q$, we define the encoding function for Reed-Solomon codes as: $C : \Sigma^k \to \Sigma^n$ where on message $m = \langle m_0, m_1, \ldots, m_{k-1} \rangle$ consider the polynomial $p(X) = \sum_{i=0}^{k-1} m_i X^i$ and $C(m) = \langle p(\alpha_1), p(\alpha_2), \ldots, p(\alpha_n) \rangle$ is the codeword.*

From the above definition, we know that a Reed-Solomon code is a mapping from a vector space of dimension k over a finite field $\mathbb{F}_q$ into a vector space of higher dimension $n > k$ over the same field. The following theorem shows that it matches the singleton bound.

Theorem 2.1.2 *Reed-Solomon code is a* $[n, k, n-(k-1)]_q$*-code.*

Proof. Let p_1 and p_2 be any two polynomials with both degree $\leq k-1$. Their codewords agree at the i-th coordinate iff $(p_1 - p_2)(\alpha_i) = 0$. Since the degree of $(p_1 - p_2)$ is $\leq k-1$, it has at most $k-1$ zeros. So the minimum distance $d \geq n-(k-1)$. With the singleton bound, we have the minimum distance $d = n-(k-1)$.

2.2 Unique-decoding of Reed-Solomon codes

The well-known decoding algorithm for Reed-Solomon codes is given by Berlekamp and Welch[15]. Before we describe the algorithm, we need to define the *error-locating polynomial.*

Definition 2.2.1 (Error-locating polynomial) *Let* e, k *be some parameters. Let* $\alpha_1, \ldots, \alpha_n$ *and* $m_1, \ldots, m_n$ *be such that there exists a polynomial* p *of degree* $\leq k-1$ *such that* $p(\alpha_i) \neq m_i$ *for* $\leq e$ *values of* i. $E(x)$ *is an* error-locating polynomial *for the above input if we have*

1. $E(\alpha_i) = 0$ *if* $p(\alpha_i) \neq m_i$.

2. $\deg(E) \leq n-k$, *and* $E \neq 0$.

Note the given E, we know the location where these errors occur by finding the i's for which $E(\alpha_i) = 0$. Then we replace m_i with ? for those i's and run erasure decoding on the resulting vector of m_i's. This works because there are at most $n-k \leq d-1$ i's for which $E(\alpha_i) = 0$.

Now we define $N(x) = E(x) \cdot p(x)$. It is clear that $(m_i - p(\alpha_i))E(\alpha_i) = 0$ for all i. Then,

$$N(\alpha_i) = E(\alpha_i)p(\alpha_i) = m_i E(\alpha_i), \text{ for all } i.$$

The BW decoding algorithm is as follows.

Berlekamp-Welch decoding algorithm

Input: $\alpha_1, \ldots, \alpha_n, m_1, \ldots, m_n$ where $\alpha_i, m_i \in \mathbb{F}_q$. k and $e \leq \lfloor (n-k)/2 \rfloor$

Procedure:

1. Find polynomial $N(x)$ and $E(x)$ such that
   (a) $N(\alpha_i) = m_i E(\alpha_i)$ for all i.
   (b) $deg(N) \leq e + k - 1$ and $N(x) \neq 0$
   (c) $deg(E) \leq e$ and $E(x) \neq 0$
2. Let $p(x) = N(x)/E(x)$

Output: The coefficients of the polynomial $p(x)$.

Theorem 2.2.2 *Let $\alpha_1, \ldots, \alpha_n$ and $m_1, \ldots, m_n$ be such that there exists a polynomial p of degree at most $k-1$ such that $p(\alpha_i) \neq m_i$ for $e \leq \lfloor (n-k)/2 \rfloor$ values of i. Then we can find the polynomial p by the Berlekamp-Welch decoding algorithm efficiently.*

Proof.

1. The polynomials $N(x)$ and $E(x)$ exist.

   Define $E(x) = \prod_{i:p(\alpha_i) \neq m_i} (x - \alpha_i)$. If there are e errors, $deg(E) = e \leq \lfloor (n-k)/2 \rfloor$ and $E(x)$ is what we call *error-locating polynomial* for our input. Then define $N(x) = E(x) \cdot p(x)$. We saw earlier that for such $N(x)$, $N(\alpha_i) = m_i E(\alpha_i)$ for all i. So the polynomials $N(x)$ and $E(x)$ both exist.

2. The polynomial p found by the BW decoding algorithm is unique.

   To prove the uniqueness of the polynomial p, we assume that there are two pairs (N, E) and (N', E') which both satisfy the condition in step 1. If $\frac{N}{E} = \frac{N'}{E'}$, we prove the uniqueness of the polynomial p.

   Let's consider two cases.

case 1: $m_i = 0$: Then $N(\alpha_i) = N'(\alpha_i) = 0$, so $\frac{N}{E} = \frac{N'}{E'} = 0$.

case 2: $m_i \neq 0$: Then $N(\alpha_i)N'(\alpha_i) = m_i E(\alpha_i)N'(\alpha_i) = N(\alpha_i)m_i E'(\alpha_i)$.

Dividing through by m_i, we get $N'(\alpha_i)E(\alpha_i) = N(\alpha_i)E'(\alpha_i)$ for all i. Since $N' \cdot E$ and $N \cdot E'$ have degree $\leq 2e + k - 1$, while $n > 2e + k - 1$, this implies that $N' \cdot E = N \cdot E'$ as polynomial. So the polynomial p is unique.

3. The algorithm is efficient.
   In step 1, it is solving a homogeneous linear system of n equations

$$\sum_{j=0}^{e+k-1} N_j \alpha_i^j = m_i \sum_{j=0}^{e} E_j \alpha_i^j$$

   in the unknowns $N_0, \ldots, N_{e+k-1}, E_0, \ldots, E_{e-1}$, and $E_e = 1$. There are $2 \cdot e + k \leq n$ unknowns and we have n equations. The straightforward way of solving the linear system works in time $O(n^3)$.

   In step 2, the division of the polynomials works in time $O(n)$.
   So the BW decoding algorithm is efficient.

2.3 List-decoding of Reed-Solomon codes

The first list-decoding algorithm for Reed-Solomon codes is invented by Madhu Sudan in 1997[13]. We describe the algorithm in this section.

Definition 2.3.1 (weighted degree) *For weights $w_x, w_y \in \mathcal{Z}^+$, the (w_x, w_y)-weighted degree of a monomial $q_{ij}x^iy^j$ is $iw_x + jw_y$. The (w_x, w_y)-weighted degree of a polynomial $Q(x,y) = \sum_i \sum_j q_{ij}x^iy^j$ is the maximum, over the monomials with non-zero coefficients, of the (w_x, w_y)-weighted degree of the monomial.*

Sudan-List-decoding algorithm for RS codes

Input: n, k, t and $\{(\alpha_1, m_1), \ldots, (\alpha_n, m_n)\}$
Procedure:

1. Let $d = k-1$, $m = \lceil d/2 - 1 \rceil$ and $l = \lceil \sqrt{(2(n+1)/d)} \rceil - 1$.
2. Find any bivariate polynomial $Q(\alpha, m)$ such that
   (a) $Q(\alpha, m)$ has $(1, d)$-weighted degree at most $m + ld$.
   (b) $Q(\alpha_i, m_i) = 0$ for all i.
   (c) $Q \neq 0$.
3. Factor Q into irreducible factors.

Output: All the polynomials p such that $(m - p(\alpha))$ is a factor of Q and $p(\alpha_i) = m_i$ for at least t values of i.

Theorem 2.3.2 *Let $\mathbb{F}$ be a field and $\{\alpha_1, m_1\}, \{\alpha_2, m_2\}, \ldots, \{\alpha_n, m_n\}$ be n distinct pairs where $\alpha_i, m_i \in \mathbb{F}$. Then we can find all polynomials $p : \mathbb{F} \to \mathbb{F}$ of degree at most $k-1$, such that $p(\alpha_i) = m_i$ for $t \geq (k-1)\lceil \sqrt{\frac{2(n+1)}{k-1}} \rceil - \lfloor \frac{(k-1)}{2} \rfloor$ values of i by the above list-decoding algorithm efficiently.*

Proof.

1. The bivariate polynomial Q in step 2 exists.

   We show how to find Q directly, which implies the existence. Let $Q(\alpha, m) = \sum_{j=0}^{l} \sum_{k=0}^{m+(l-j)d} q_{kj} \alpha^k m^j$. We want to find coefficients q_{kj} satisfying the constraints $\sum_{j=0}^{l} \sum_{k=0}^{m+(l-j)d} q_{kj} \alpha_i^k m_i^j = 0$ for all i. The homogenous linear system has $(m+1)(l+1) + d \cdot l(l+1)/2 > n$ unknowns and n equations. Hence, a non-zero solution exists.

2. If $t \geq (k-1)\lceil \sqrt{\frac{2(n+1)}{k-1}} \rceil - \lfloor \frac{(k-1)}{2} \rfloor$ and p is a polynomial such that $p(\alpha_i) = m_i$ for at least t values of i, the polynomial p must be in the output list of the list-decoding algorithm.

It is equivalent to prove that $(m - p(\alpha))$ divides $Q(\alpha, m)$. First, let $f(\alpha) = Q(\alpha, p(\alpha))$. Since $Q(\alpha, m) = \sum_{j=0}^{l} \sum_{k=0}^{m+(l-j)d} q_{kj}\alpha^k m^j$ and it has $(1, d)$-weighted degree at most $m+ld$, we have that $k+jd \leq m+ld$. And our $f(\alpha) = \sum_{j=0}^{l} \sum_{k=0}^{m+(l-j)d} q_{kj}\alpha^k (f(\alpha)^j)$ has degree at most $k+jd$. Thus, we know that $f(\alpha)$ has degree at most $m + ld$.

Since p is a polynomial such that $p(\alpha_i) = m_i$ for at least t values of i, we have $f(\alpha_i) = Q(\alpha_i, p(\alpha_i))$ is zero for greater than $t \geq (k-1)\lceil\sqrt{\frac{2(n+1)}{k-1}}\rceil - \lfloor\frac{(k-1)}{2}\rfloor = d(l+1) - \lfloor\frac{d}{2}\rfloor > m + ld$ points. But $f(\alpha)$ has degree at most $m + ld$, so $f(\alpha) = Q(\alpha, p(\alpha))$ is identical zero, which means $(m - p(\alpha))$ divides $Q(\alpha, m)$.

The above is the first list decoding algorithm for RS codes. Actually, an improve list-decoding algorithm for RS code has been invented in 1999 by Guruswami and Sudan[5]. It can correct up to $N - \sqrt{(K-1)N}$ for a (N, K) RS code. So more errors can be corrected with the GS-decoder. But since the polynomial Q during decoding has much higher degree than the original algorithm, resulting in taking much more time, we choose to implement Sudan's list decoding algorithm. To be consistent with the implementation and the experimental results, we will use Sudan's algorithm throughout this paper.

2.4 Decoding of RS codes with erasures

Consider a (N, K) RS code, let $C(M)_{RS} = \langle m_1, m_2, \ldots, m_N \rangle$ denote the codeword for some message M. If the codeword suffers $e \leq N - K$ erasures in the last e tuples, the receive word is $\langle m_1, m_2, \ldots, m_{N-e}, *, \ldots, * \rangle$. Then we can do error-correcting on the first $N - e$ tuples by viewing the first $N - e$ tuples as a new receive word $\langle m_1, m_2, \ldots, m_{N-e} \rangle$ corresponding to a $(N - e, K)$ RS code. From theorem **??**, we know that a $(N - e, K)$ RS code can correct up to $\lfloor (N - e - K)/2 \rfloor$ errors with unique-decoding. And from theorem 2.3.2, a $(N - e, K)$ RS code can correct up to $(N - e) - (K-1)\lceil\sqrt{\frac{2(N-e+1)}{K-1}}\rceil - \lfloor\frac{(K-1)}{2}\rfloor$ errors with list-decoding. So, we have that a (N, K) RS code can be unique decoded from $e \leq N - K$ erasures and $\lfloor (N - e - K)/2 \rfloor$ errors by using the BW unique-decoding algorithm or list decoded from $e \leq N-K$ erasures and $(N-e)-(K-1)\lceil \sqrt{\frac{2(N-e+1)}{K-1}} \rceil + \lfloor \frac{(K-1)}{2} \rfloor$ errors by using Sudan's list-decoding algorithm.

2.5 An example

Now we show an example of how to do list-decoding of a RS code. As an example, let C_{RS} be a $[16, 2, 15]_{2^4}$ Reed-Solomon code. Then $C_{RS} : \Sigma^2 \to \Sigma^{16}$ where Σ is $GF(2^4)$. We use "0" to denote 0000_2, "1" to denote 0001_2, "2" to denote 0010_2, "3" to denote 0011_2, ..., "14" to denote 1110_2, "15" to denote 1111_2. Table 2.1 and 2.2 show the definition of operations $+$ and $\cdot$ over $GF(2^4)$.

Consider the message $m = \langle 6, 8 \rangle$ and its corresponding polynomial is $p(x) = 6 + 8x$. Then

$$\begin{aligned} C_{RS}(m) &= \langle\, p(1),\, p(2),\, p(3),\, \ldots,\, p(13),\, p(14),\, p(15),\, p(0)\,\rangle \\ &= \langle\, 14,\, 5,\, 13,\, 0,\, 8,\, 3,\, 11,\, 10,\, 2,\, 9,\, 1,\, 12,\, 4,\, 15,\, 7,\, 6\,\rangle \end{aligned}$$

From theorem 2.3.2, we know that the original message m must be in the output list of the list-decoding algorithm if there are at least 6 elements in $C_{RS}(m)$ not changed. It means that the number of errors can be up to 10, which is greater than the traditional "error-correction radius" $e = \lfloor \frac{D-1}{2} \rfloor = \lfloor \frac{15-1}{2} \rfloor = 7$.

Let $C'_{RS}(m)$ be the word that after $C_{RS}(m)$ occurs some errors.

$$\begin{array}{rll} & C_{RS}(m) & \langle\, 14,\, 5,\, 13,\, 0,\, 8,\, 3,\, 11,\, 10,\, 2,\, 9,\, 1,\, 12,\, 4,\, 15,\, 7,\, 6\,\rangle \\ \mapsto & C'_{RS}(m) & \langle\, 14,\, \underline{6},\, 13,\, \underline{1},\, \underline{9},\, \underline{4},\, 11,\, \underline{11},\, 2,\, \underline{10},\, 1,\, \underline{13},\, \underline{5},\, \underline{0},\, 7,\, \underline{7}\,\rangle \end{array}$$

Then we need to find any non-zero bivariate polynomial Q with $(1, 1)$-weighted degree at most 5 and it passes through $(1, 14)$, $(2, 6)$, $(3, 13)$, $(4, 1)$, $(5, 9)$, $(6, 4)$, $(7, 11)$, $(8, 11)$, $(9, 2)$, $(10, 10)$, $(11, 1)$, $(12, 13)$, $(13, 5)$, $(14, 0)$,

+	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	0	3	2	5	4	7	6	9	8	11	10	13	12	15	14
2	2	3	0	1	6	7	4	5	10	11	8	9	14	15	12	13
3	3	2	1	0	7	6	5	4	11	10	9	8	15	14	13	12
4	4	5	6	7	0	1	2	3	12	13	14	15	8	9	10	11
5	5	4	7	6	1	0	3	2	13	12	15	14	9	8	11	10
6	6	7	4	5	2	3	0	1	14	15	12	13	10	11	8	9
7	7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8
8	8	9	10	11	12	13	14	15	0	1	2	3	4	5	6	7
9	9	8	11	10	13	12	15	14	1	0	3	2	5	4	7	6
10	10	11	8	9	14	15	12	13	2	3	0	1	6	7	4	5
11	11	10	9	8	15	14	13	12	3	2	1	0	7	6	5	4
12	12	13	14	15	8	9	10	11	4	5	6	7	0	1	2	3
13	13	12	15	14	9	8	11	10	5	4	7	6	1	0	3	2
14	14	15	12	13	10	11	8	9	6	7	4	5	2	3	0	1
15	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Table 2.1: + operation over $GF(2^4)$

$\cdot$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2	0	2	4	6	8	10	12	14	3	1	7	5	11	9	15	13
3	0	3	6	5	12	15	10	9	11	8	13	14	7	4	1	2
4	0	4	8	12	3	7	11	15	6	2	14	10	5	1	13	9
5	0	5	10	15	7	2	13	8	14	11	4	1	9	12	3	6
6	0	6	12	10	11	13	7	1	5	3	9	15	14	8	2	4
7	0	7	14	9	15	8	1	6	13	10	3	4	2	5	12	11
8	0	8	3	11	6	14	5	13	12	4	15	7	10	2	9	1
9	0	9	1	8	2	11	3	10	4	13	5	12	6	15	7	14
10	0	10	7	13	14	4	9	3	15	5	8	2	1	11	6	12
11	0	11	5	14	10	1	15	4	7	12	2	9	13	6	8	3
12	0	12	11	7	5	9	14	2	10	6	1	13	15	3	4	8
13	0	13	9	4	1	12	8	5	2	15	11	6	3	14	10	7
14	0	14	15	1	13	3	2	12	9	7	6	8	4	10	11	5
15	0	15	13	2	9	6	4	11	1	14	12	3	8	7	5	10

Table 2.2: $\cdot$ operation over $GF(2^4)$

$(15,7)$ and $(0,7)$. Since the bivariate polynomial Q has 21 unknowns and we have 16 equations, the polynomial Q exists. After solving the homogenous linear system, we choose one of the non-zero solutions be our polynomial Q.

$$\begin{aligned} Q(x,y) &= 4+2x+4y+6x^2+12xy+4y^2+10x^3+4x^2y+13xy^2+y^3+x^4 \\ &\quad +9x^3y+xy^3+y^4+12x^5+3x^4y+13x^3y^2+x^2y^3+xy^4+y^5 \end{aligned}$$

After factoring the polynomial Q into irreducible polynomials, we get

$$\begin{aligned} Q(x,y) &= 4+2x+4y+6x^2+12xy+4y^2+10x^3+4x^2y+13xy^2+y^3+x^4 \\ &\quad +9x^3y+xy^3+y^4+12x^5+3x^4y+13x^3y^2+x^2y^3+xy^4+y^5 \\ &= (6+8x+y)\cdot(7+8x+y) \\ &\quad \cdot(4+4x+5x^2+8xy+x^3+13x^2y+xy^2+y^3) \end{aligned}$$

Both of the polynomials $p_1(x)=6+8x$ and $p_2(x)=7+8x$ satisfy the output constraints of the list-decoding algorithm. So we get two possible messages $\langle 6,8\rangle$ and $\langle 7,8\rangle$.

We can check them again.

$$\begin{aligned} C'_{RS}(m) &= \langle 14, 6, 13, 1, 9, 4, 11, 11, 2, 10, 1, 13, 5, 0, 7, 7\rangle \\ C_{RS}(\langle 7,8\rangle) &= \langle \underline{15}, \underline{4}, \underline{12}, 1, 9, \underline{2}, \underline{10}, 11, \underline{3}, \underline{8}, \underline{0}, 13, 5, \underline{14}, \underline{6}, 7\rangle \end{aligned}$$

and

$$\begin{aligned} C'_{RS}(m) &= \langle 14, 6, 13, 1, 9, 4, 11, 11, 2, 10, 1, 13, 5, 0, 7, 7\rangle \\ C_{RS}(\langle 6,8\rangle) &= \langle 14, \underline{5}, 13, \underline{0}, \underline{8}, \underline{3}, 11, \underline{10}, 2, \underline{9}, 1, \underline{12}, \underline{4}, \underline{15}, 7, \underline{6}\rangle \end{aligned}$$

Both agree with C'_{RS} in 6 positions.

Chapter 3

The RAID-like system

From the previous chapter, we have shown that the error-correcting capability of Reed-Solomon codes can be more powerful with the list-decoding algorithm. In this chapter, we show an application in data storage with the list decoding.

3.1 Background

A RAID-like system consists of individual storage devices and each of them works independently as shown in figure 3.1.

Today, if we want to safeguard a document in a RAID-like system, the most trivial method is to replicate it, and to store the different copies in different storage devices. Suppose the RAID-like system has n devices, and each of them holds one copy. Then we can reconstruct the original document by doing majority vote on the copies collected from each device if no more than half of the system incur arbitrary failures, including alterations to the data. However, the method has a main drawback that the required storage space grows linearly with the number of storage devices.

Rather than using the method of replication, those who are familiar with the coding theory may use an *error-correcting code* to encode the document. Suppose the size of the document is s and we use a $[N, K, D]_q$ Reed-Solomon

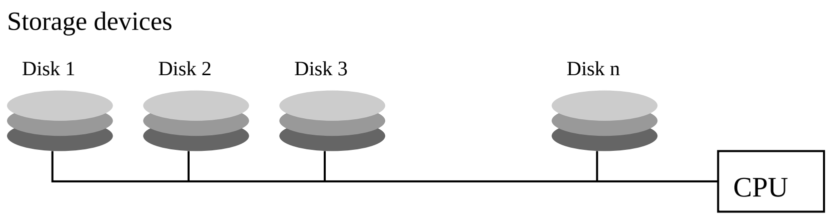


Figure 3.1: The RAID-like system

code where $q = 2^{s/K}$ and $N = n$, the number of storage devices. The document can be viewed as a vector space of dimension K over a field $\mathbb{F}$ of size q. By the encoding function of the code, the vector space of dimension K maps into a vector space of dimension N over the same field and each device holds the data of one dimension. With the unique-decoding of Reed-Solomon codes, we can reconstruct the original document when up to $\lfloor (D-1)/2 \rfloor = \lfloor (N-K)/2 \rfloor$ storage devices incur arbitrary failures, including alterations to data stored in them.

The above describes a traditional setting when using an *error-correcting code* with traditional unique-decoding to safeguard a document in a RAID-like system. Since the distance D of a code is at most equal to the block length N, the fraction of the failure storage devices in the system is at most $\approx 1/2$ when $N \to \infty$. That is, the above method can never satisfy the failure model that the fraction of failure storage devices is more than $1/2$, whatever the *error-correcting codes* we use. So it is a challenge to construct a RAID-like system with high fault tolerance that more than half of the storage devices incur arbitrary failures.

3.2 The high fault-tolerant RAID-like system

The RAID-like system needs to provide two important functions: *store* and *retrieve*. The *store* function takes a document as the input, and transforms the document into several data pieces. On the other hand, the *retrieve* function takes several data pieces as the inputs, and tries to reconstruct the

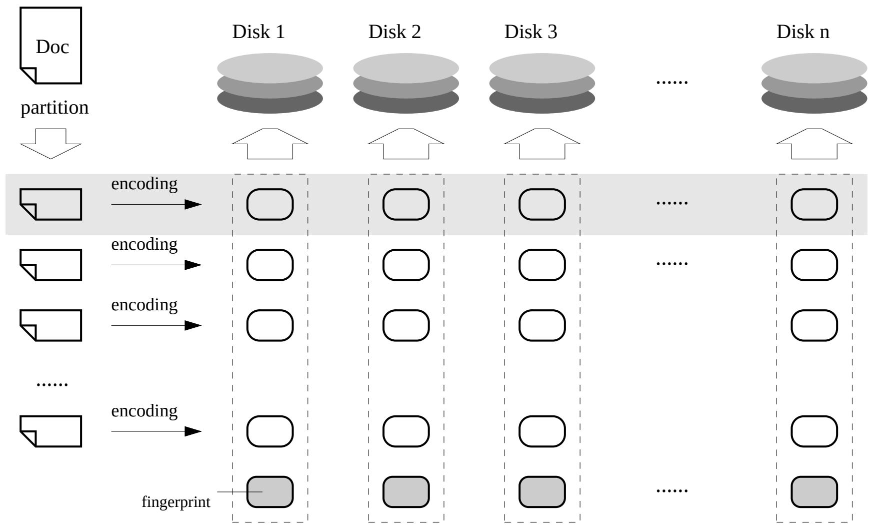


Figure 3.2: The *store* function

original document based on the information in those pieces.

In this chapter, we assume that there are n storage devices in the system, the size of the document is s bits and the number of faulty storage devices in the system is at most e where $e \geq n/2$. We use $ErrBound_U(e)$ to denote the error-correcting bound of unique-decoding with e erasures. $ErrBound_L(e)$ denotes the error-correcting bound of list-decoding with e erasures.

3.2.1 The store function

First, we need to choose a proper $(N, K)_q$ Reed-Solomon code where $N = n$ and q is as small as possible such that the code can correct up to e errors. So the code can encode $M = K * \log_2 q$ bits. Then, we partition the document into s/M fragments and encode each fragment separately. After encoding, each fragment is transformed into n data pieces and we store them on the n storage devices separately. At last, each device holds s/M data pieces and an extra hash value of the original document as fingerprint. Figure 3.2 shows the *store* function.

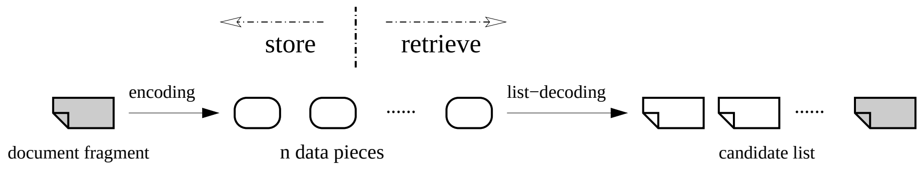


Figure 3.3: Get the candidate list for a document fragment

3.2.2 The retrieve function

The goal of the *retrieve* function is to reconstruct the original document. First, we can get a candidate list for each document fragment by performing the list-decoding algorithm on its corresponding data pieces. If there are at most e storage devices failure, every candidate list must contain the original corresponding document fragment. See figure 3.3. Now we show how to choose the correct one from the candidate list for all document fragments.

Step 1. Let I be an empty set that is used to collect the indices of the failure storage devices.

Step 2. For all candidate lists, do as follows: if the candidate list has only one message m, let $C(m)_{RS}$ denote the $(N,K)_q$ RS code of m and $D_0, D_1, \ldots, D_{n-1}$ denote the corresponding data pieces of m.

$$I = I \cup \{i : C(m)_{RS_i} \neq D_i\} \text{ where } 0 \leq i \leq n-1$$

Then, the set I has some indices of failure storage devices.

Step 3. For all candidate lists of size which is greater than one, do as follows: If $e - |I| \leq ErrBound_U(|I|)$, we can use unique-decoding with erasures to get the correct document fragment. Otherwise, assume the candidate list has messages $m_1, m_2, \ldots, m_l$. Let $C(m)_{RS}$ denote the $(N,K)_q$ RS code of some message m and $D_0, D_1, \ldots, D_{n-1}$ denote the corresponding data pieces of m. For the message m_j where $1 \leq j \leq l$, if

$$\mid I \cup \{i : C(m_j)_{RS_i} \neq D_i\} \mid > e \text{ where } 0 \leq i \leq n-1$$

we delete the message m_j from the candidate list.

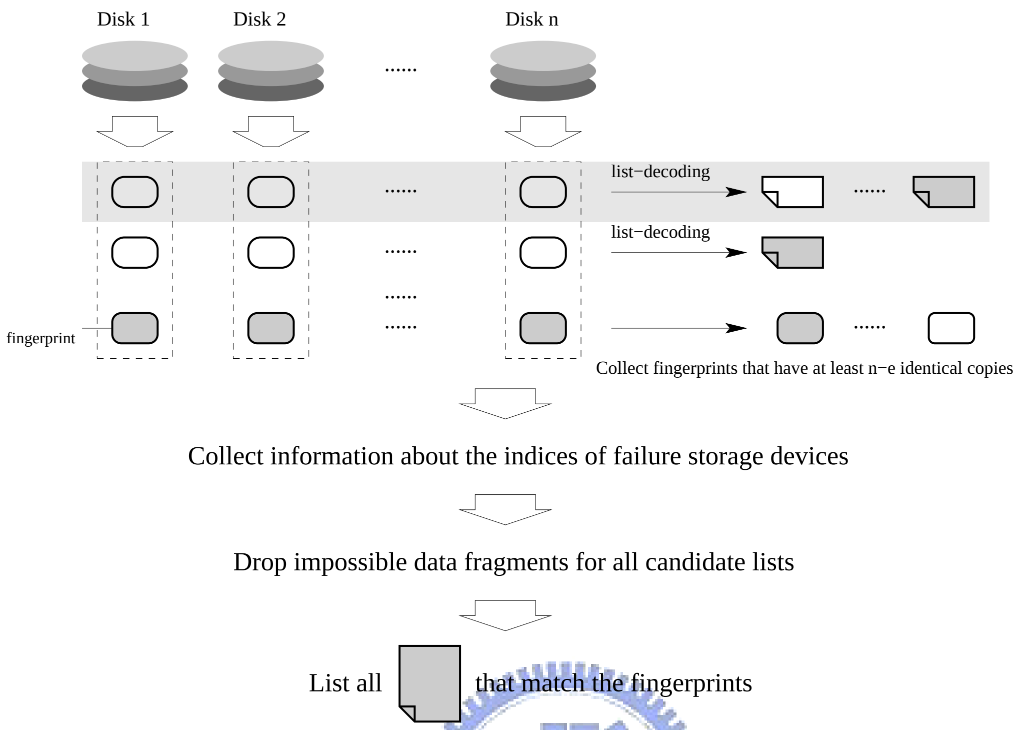

Disk 1
Disk 2
Disk n
list−decoding
list−decoding
fingerprint
Collect fingerprints that have at least n−e identical copies
Collect information about the indices of failure storage devices
Drop impossible data fragments for all candidate lists
List all
that match the fingerprints

Figure 3.4: The *retrieve* function

Step 4. If the sizes of all candidate lists are one. Concatenate them and we get the original document. Otherwise, compare all combinations with the fingerprints that have at least $n-e$ identical copies in the n storage devices to get all possible documents.

We summarize all of the procedures in figure 3.4.

Since the list-decoding algorithm takes more time, we can reorder some procedures in figure 3.4 to speed up the retrieve function. That is, every time we get the candidate list for some document fragment, we can update the set I, if the list size is one. Then, we can discard the indices in I as the positions of erasures. If $e - |I| \leq ErrBound_U(|I|)$, we use a unique-decoder with erasures to decode the next document fragment. Otherwise, $Errbound_U(|I|) < e - |I| \leq ErrBound_L(|I|)$, we use a list-decoder with erasures. The more information about the erasures, the less time taken by the list-decoding algorithm. Figure 3.5 shows a speedup version of the retrieve

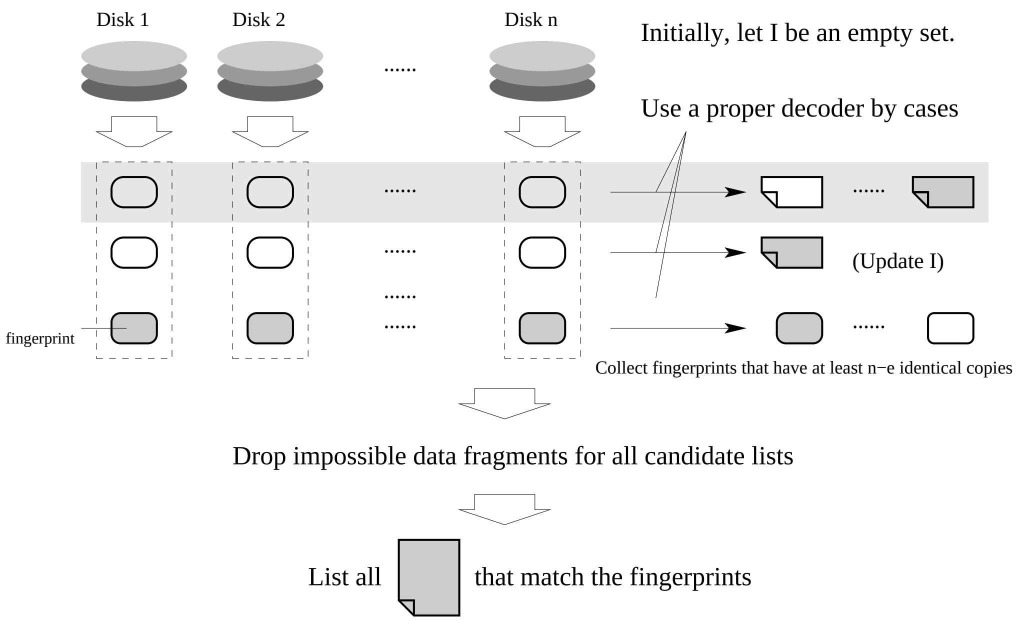

Disk 1
Disk 2
Disk n
Initially, let I be an empty set.
Use a proper decoder by cases
(Update I)
fingerprint
Collect fingerprints that have at least n−e identical copies
Drop impossible data fragments for all candidate lists
List all
that match the fingerprints

Figure 3.5: The *efficient-retrieve* function

function.

3.3 Choose a proper $(N, K)_q$ RS code

In the *store* function, we need a proper $(N, K)_q$ RS code as the encoder. Here, we show how to choose a code with error-correcting rate 0.5.

First, N is set to n, the number of storage devices in the system and q is set to $2^{\lceil \lg_2 n \rceil}$. From theorem 2.3.2, we know that an (N, K) RS code with list-decoding can correct up to $N - (K-1)\lceil\sqrt{\frac{2(N+1)}{K-1}}\rceil + \lfloor\frac{(K-1)}{2}\rfloor$ errors. If we want our system to correctly retrieve the stored document with half of the system failed, that is, we need to choose K with some fixed N to satisfy

$$(K-1)\lceil\sqrt{\frac{2(N+1)}{K-1}}\rceil + \lfloor\frac{(K-1)}{2}\rfloor \leq N/2.$$

It is clear that if $K \leq \lfloor 5N + 7 - 2\sqrt{2(N+1)(3N+4)}\rfloor$, then the above inequality holds.

N	K	Information rate	Blowup factor	Message size
$2^4 = 16$	2	$\frac{2}{16} = 0.1250$	8	8 bits
$2^5 = 32$	4	$\frac{4}{32} = 0.1250$	8	20 bits
$2^6 = 64$	7	$\frac{7}{64} \approx 0.1094$	9.14	42 bits
$2^7 = 128$	13	$\frac{13}{128} \approx 0.1016$	9.85	91 bits
$2^8 = 256$	26	$\frac{26}{256} \approx 0.1016$	9.85	208 bits
$2^9 = 512$	52	$\frac{52}{512} \approx 0.1016$	9.85	468 bits
$2^{10} = 1024$	104	$\frac{104}{1024} \approx 0.1016$	9.85	1040 bits
$2^{11} = 2048$	207	$\frac{207}{2048} \approx 0.1010$	9.89	2277 bits
$2^{12} = 4096$ ...	414 ...	$\frac{414}{4096} \approx 0.1010$ ...	9.89 ...	4968 bits ...
∞		≈ 0.1010	9.90	

Table 3.1: Some (N, K) RS codes with 1/2 error-correcting rate

Table 3.1 lists some RS codes that can be list-decoded from $N/2$ errors with maximum rate when using the Sudan's list decoding algorithm.

Chapter 4

Experimental results

In the previous chapter, we have described the design of our RAID-like system. To study the reliability of the storage system, we show some experimental results in this chapter.

4.1 The list size of the list-decoder

The list size of the list-decoding algorithm for RS codes plays an important role in the design of our RAID-like system. The previous work[9] on this topic by R. J. McEliece shows that list-decoder almost always returns a list of size one. In this section, we show this by experimental results. We may choose a $(N, K)_q$ RS code and assume it can correct up to e errors with the list-decoding algorithm. In each experiment, we will choose randomly a message and encode the message with the chosen RS code. After encoding, we get a codeword and randomly replace the values of e' positions for the codeword where $(N - K + 1) - e \leq e' \leq e$. Finally, we decode the codeword by the list-decoding algorithm and collect the outputs into the set L. For all e', we do the experiment t times.

Table 4.1 shows the experimental results for the $(16, 2)_{2^4}$ RS code which can correct up $e = 10$ errors with the list-decoding algorithm. Table 4.2 is for the $(32, 4)_{2^5}$ code with $e = 18$.

$t = 100,000$ for each e'

e': # of errors	$\#\{\lvert L\rvert > 1\}$	The fraction of $\lvert L\rvert > 1$
10	2389	0.02389
9	1408	0.01408
8	720	0.00720
7	340	0.00340
6	99	0.00099
5	24	0.00024

Table 4.1: The list size of the list-decoder for the $(16,2)_{2^4}$ RS code

$t = 1,000,000$ for each e'

e': # of errors	$\#\{\lvert L\rvert > 1\}$	The fraction of $\lvert L\rvert > 1$
18	0	0
17	0	0
16	0	0
15	0	0
14	0	0
13	0	0
12	0	0
11	0	0

Table 4.2: The list size of the list-decoder for the $(32,4)_{2^5}$ RS code

The results in Table 4.2 may seems unusual. It shows that the output list of the list-decoder is size one with very high probability. It indicates that in most cases the list-decoder behaves just like a conventional decoder(unique-decoding algorithm).

4.2 The RAID-like system

Before we show the experimental results of the system, we introduce its simulation environment first.

4.2.1 The simulation environment

The snapshots of the simulation environment are shown in Figure 4.1 and Figure 4.2. The main window is divided into three areas, which are named "Step 1", "Step 2", and "Step 3".

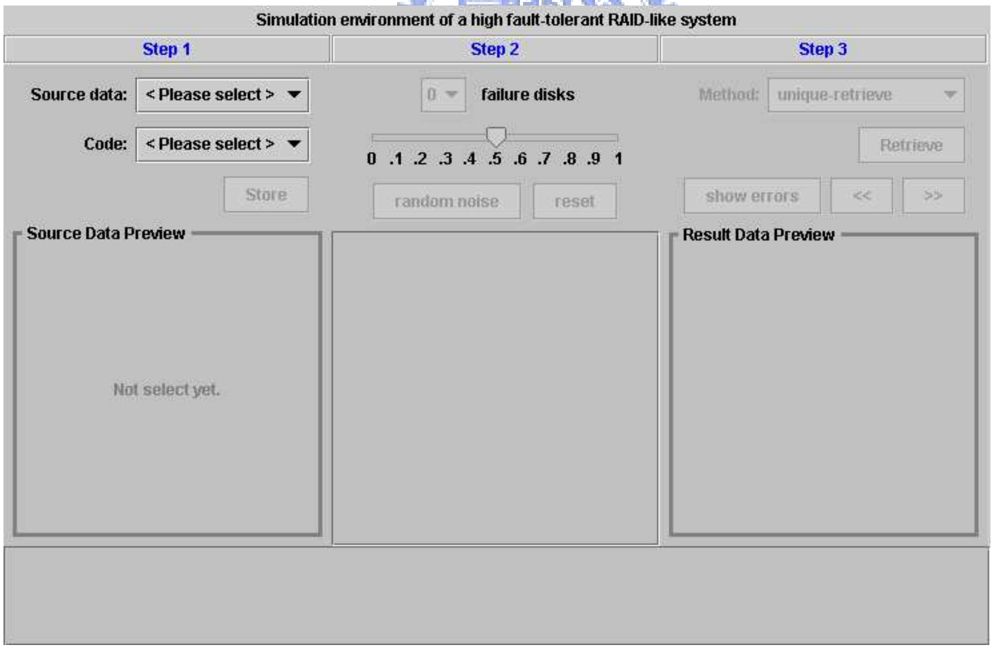
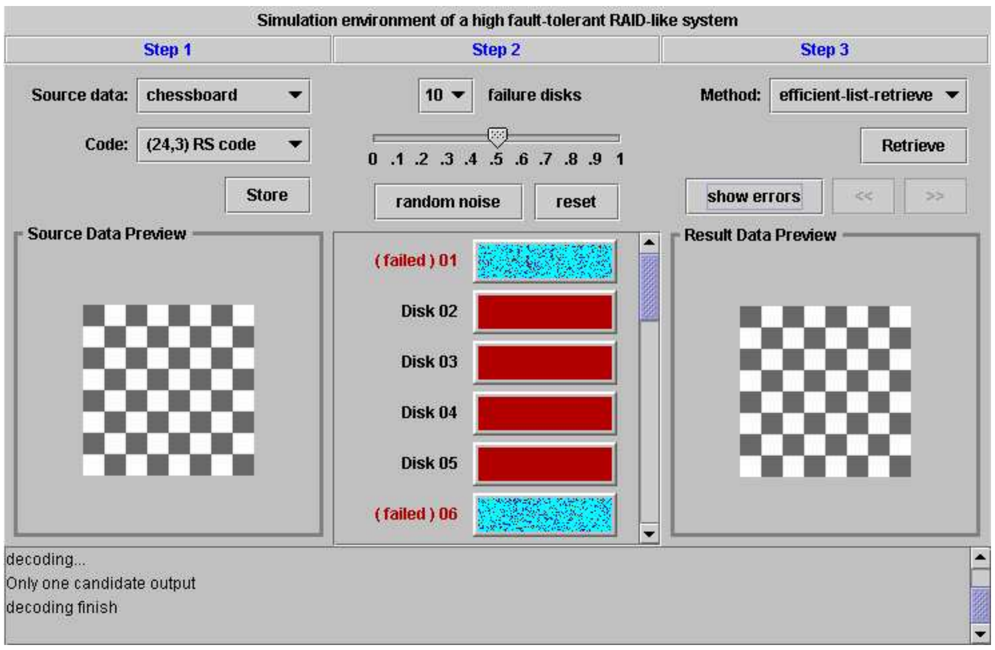


Figure 4.1: Snapshot 1 of the simulation environment

"Step 1" Area:



Source data: chessboard
Code: (24,3) RS code
10 failure disks
Method: efficient-list-retrieve

```
decoding...
Only one candidate output
decoding finish
```

Figure 4.2: Snapshot 2 of the simulation environment

In this area, users can select a source data (document) and a code as the encoder. Once both of them are selected, users can press the button named "**Store**" to perform the *store* function. In order to easily compare the original document in "Step 1" area with the retrieved document in "Step 3" area, we use images as the sources. Once select an image, users can preview the image on the bottom of this area.

"Step 2" Area:

Once press the "**Store**" button in "Step 1" area, all disks of the system will store the encoded data. Then, we can use the button named "**random noise**" to add random errors to the disks. The "**reset**" button is used to reset all disks to a state of error free.

Before pressing the "**random noise**" button, users can decide the number of failure disks and an error probability. Now suppose the number of failure disks is e, and the error probability is p. After pressing the "**random noise**" button, it will randomly select e disks as failure disks and the error probability for each data piece and each fingerprint on these failure disks will be p.

"Step 3" Area:

In this area, users can select a retrieving method to retrieve the document from the encoded data stored in those disks. Once select the retrieving method, users can press the button named "**Retrieve**" to perform the *retrieve* function and the result will be shown on the bottom of this area. If users just want to see where the errors occur, then press the button named "**show errors**". If we have more than one outputs, the button named "$\ll$" and "$\gg$" can be used to switch between them.

Here we provide two retrieving method. The first retrieving method is called *unique-retrieve*. It uses the unique-decoding algorithm as the decoder and its error-correction bound is half of the minimum distance of the code. Another retrieving method is called *efficient-retrieve*. It uses the list-decoding algorithm as the decoder and hence it has higher error-correction capability.

4.2.2 Simulation results

We use the *Lena* (or *Lenna*) picture, which is one of the most widely used images for data compression, to be our source data (document) and the $(16, 2)_{2^4}$ RS code, which can correct up to 10 errors, to be our encoder. The number of failure disks is set to 10. Figure 4.3 shows the result when the error probability p is set to 0.3. Figure 4.4 is the result when $p = 0.5$. Figure 4.5 is the result when $p = 0.7$.

The $(16, 2)_{2^4}$ RS code can correct up to 7 errors with unique-decoding and 10 errors with list-decoding. So once the encoded data pieces for some document fragment has more than 7 errors, the unique-retrieve method cannot reconstruct the original fragment and some parts of the retrieved picture will be corrupted. But for the efficient-retrieve method, it can correctly retrieve the original picture under this setting.

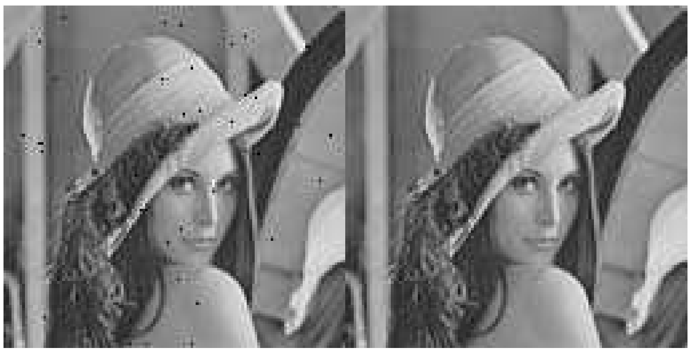


Figure 4.3: Unique-retrieve (left) and efficient-retrieve (right) with $p = 0.3$.

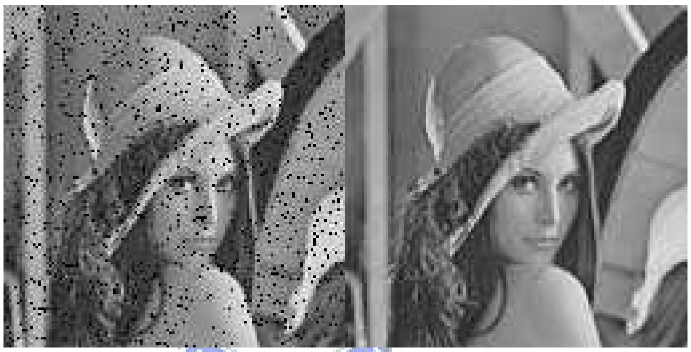


Figure 4.4: Unique-retrieve (left) and efficient-retrieve (right) with $p = 0.5$.

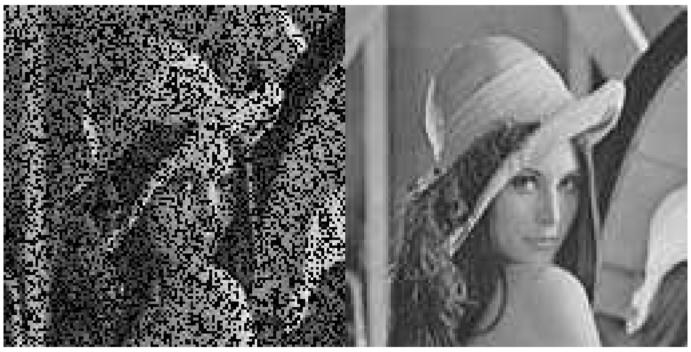


Figure 4.5: Unique-retrieve (left) and efficient-retrieve (right) with $p = 0.7$.

Chapter 5

Conclusion

In the thesis, we have shown how to use one of the *list-decodable codes*, Reed-Solomon codes with list-decoding to build a RAID-like system with high fault tolerance. Moreover, the system can offer traditional error-correcting capability if we replace a unique-decoder with the list-decoder. Then we can save much time in decoding if there are only few disks failure.

Since list-decoding of RS codes can be viewed as bivariate interpolation and factorization problem[2, 3], the *retrieve* function in this system may be too slow if the size of the document is too large. To overcome this problem, we can use other list-decodable codes (e.g. expander codes) that have low complexity in time to replace RS codes although we may need more disk space to store the encoded data.

Bibliography

[1] P. Elias, “List decoding for noisy channels,” *Institute of Radio Engineers*, 94-104, 1957.

[2] J. von zur Gathen and E. Kaltofen, “Polynomial time factorization of multivariate polynomials over finite fields,” *Math. Comput*, 45:251–261, 1985.

[3] Shuhong Gao and Alan G.B. Lauder, “Hensel lifting and bivariate polynomial factorisation over finite fields,” *Mathematics of Computation*, Volume 71, Issue 240, October 2002.

[4] V. Guruswami and M. Sudan, “List decoding algorithms for certain concatenated codes,” *Proceedings of the 32nd annual ACM symposium on Theory of computing*, 181–190, May 2000.

[5] V. Guruswami and M. Sudan, “Improved Decoding of Reed-Solomon Codes and Algebraic Geometry Codes,” *IEEE Transactions on Information Theory*, vol. 45, no. 6, pp. 1757-1767, September 1999.

[6] V. Guruswami, A. Sahai and M. Sudan, “Soft-Decision Decoding of Chinese Remainder Codes,” *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, 2000.

[7] V. Guruswami and P. Indyk, “Linear Time Encodable and List Decodable Codes,” *STOC*, 2003.

[8] R. Koetter and A. Vardy, "Algebraic soft-decision decoding of Reed-Solomon codes," *IEEE Transactions on Information Theory*, August 31 2001

[9] R. J. McEliece, "On the average list size for the Guruswami-Sudan Decoder," *International Symposium on Communication Theory and Applications*, July 2003.

[10] R. Pellikaan and Xin-Wen Wu, "List decoding of q-ary Reed-Muller codes," *IEEE Transactions on Information Theory*, April 2004.

[11] J. S. Plank, "A Tutorial on Reed-Solomon Coding for Fault-Tolerance in RAID-like Systems," *Software, Practice and Experience*, 27(9):995-1012, September 1997.

[12] Irving S. Reed and Gustave Solomon, "Polynomial Codes over Certain Finite Fields," *Journal of the Society for Industrial and Applied Mathematics*, 1960.

[13] Madhu Sudan, "Decoding of Reed-Solomon codes beyond the error-correction bound," *Journal of Complexity*, 13(1):180–193, 1997.

[14] C.-Y. Teng, R.-J. Chen, M.-Y. Liu and S.-C. Tsai, "Extractor Codes with Applications," *NCS 2003*.

[15] L. R. Welch and E. R. Berlekamp, "Error correction for algebraic block codes," *US patent*, Number 4, 633, 470, 1986.

[16] J. M. Wozencraft, "List decoding," *Quarterly Progress Report, Research Laboratory of Electronics, MIT*, 48:90-95, 470, 1958.