Chapter 6 Conclusions and Future Directions

6.1 Conclusions

This thesis deals with two issues in highly available distributed storage systems: reliability and efficiency. To achieve reliability, two new families of MDS array codes are presented, whose optimal encoding operations are optimal. To improve efficiency (performance) of general distributed data systems, including storage systems, two problems and their solutions are proposed, namely the data distribution problem and data acquisition problem. A new efficient deterministic voting algorithm for distributed data is also presented.

The two new classes of MDS array codes of distance 3, the X-code and the B-Code, are the first two classes of array codes that place parity and information bits in the same column. The two codes are still *systematic* in the sense that all information bits can be directly obtained from their codewords without any computation. The encoding operations of both codes are optimal, i.e., their update complexity achieves the lower bound. In addition to encoding algorithms to these array codes, efficient decoding algorithms, both for erasure-correcting and for error-correcting, are also proposed.

The X-Code has a very simple geometrical structure, namely the parity bits are constructed along two groups of parallel *parity lines* of slopes 1 and -1 respectively, where its name comes from. This simple geometrical structure enables simple erasure-decoding and error-decoding algorithms, using only XORs and vector cyclic-shift operations.

The B-Code is related to a 3-decade old graph theory problem. It is proven in this thesis that constructing a B-Code of odd length is exactly equivalent to constructing a perfect one-factorization (or P1F) of a complete graph. Thus if the P1F conjecture is solved, then B-Codes of arbitrary length can be constructed. On the other hand, B-Codes of new lengths can lead to constructions of P1F of new complete graphs. An efficient error-correcting algorithm for the B-Code is also presented, based on the relations between the B-Code and its dual; this algorithm might give a hint of developing efficient decoding algorithms for other codes.

To show that in distributed systems, data redundancy should be actively introduced to improve system performance, a novel deterministic voting scheme that uses error-correcting codes is proposed, which generalizes all known simple deterministic voting algorithms. The new voting scheme greatly reduces communication complexity while still providing the correct deterministic voting result. The scheme can be tuned for optimal average case communication complexity by choosing the parameters of the error-correcting code, thus it is very adaptive to various application environments with different error rates.

It is also shown that in general distributed storage systems, proper data redundancy can improve the performance of the systems. Two problems are identified to improve the performance of general data server systems, namely the data distribution problem and the data acquisition problem. Solutions are proposed, as are general analytical results on the performance of (n, k) systems. A simple service time model of a practical disk-based distributed server system is given based on experimental results, which is used as a starting point for data distribution and data acquisition schemes. These results can be used in more sophisticated scheduling schemes that optimize or improve the performance of data server systems that serve multiple clients at the same time.

6.2 Future Directions

Much research still needs to be done to improve the availability of distributed storage systems. More error-correcting codes with low computational overhead that provide flexible reliability need to be designed. Array codes are a good class of codes that should get more researchers' attention. Distributed storage systems can become popular only with matching distributed file systems. Many issues in distributed file systems, such as efficiently reaching consistency of distributed data, maintaining data integrity in the presence of transient faults, and achieving graceful performance degradation while faulty parts of systems are replaced, should be solved. Also distributed storage systems should be easy to scale, easy to manage and easy to maintain. In short, there are many research problems to solve in order to build highly available distributed storage systems.

Many open problems directly related to the topics in the thesis have been already raised in the previous chapters. We summarize some important ones here:

- More MDS array codes: here *more* has many meanings. We need to find codes with distances greater than 3. X-like codes may be a solution. We also need to find codes with more lengths, ideally, arbitrary lengths. Codes that have parity bits mixed with information bits in the same column are typically difficult to shorten, thus one solution to this problem is to find more codes with more lengths. While they have optimal encoding operations and optimal erasure-decoding in terms of the total number of operations, the X-Code and the B-Code are *not* optimal in decoding in number of decoding steps if *parallel* decoding is used. Since in distributed storage systems, parallel decoding means reading data in parallel, more codes with fewer parallel decoding steps can improve the data-read performance of many applications. The *lower bound* of parallel erasure-decoding steps for a given code is *not* even known yet.

- The B-Code: it still needs to be proven that the B-Code of length $2n+1$ can always be constructed from the B-Code of length $2n$, thus the B-Code is totally equivalent to the P1F. Also how to construct a new B-Code from the known B-Codes is very useful, since this gives new P1F construction methods, in addition to more codes that can be obtained.

- Voting: the efficient voting algorithm in Chapter 4 uses 2 or 3 rounds. It is an open problem whether there is a voting scheme that always uses *only* 1 round, but can still reduce average communication complexity. Also the voting schemes discussed use a broadcast model, another interesting problem is how to reduce average communication complexity if a point-to-point communication model is used.

- General data servers: Chapter 5 deals with the case of only one client at a time. But in practical data server systems, it is common to have multiple clients at the same time or a group of requests that are already queued. How to schedule the processing of these multiple data requests is an interesting and difficult research problem.

We conclude this thesis by two more problems related to codes that can be used in distributed storage systems to improve availability.

6.2.1 Reed-Solomon Codes as Array Codes

Reed-Solomon codes are more flexible in lengths and distances. Their only shortcoming is their relatively complex encoding and decoding operations which use finite field operations. Since array codes are 2-dimensional, they are generalizations of 1-dimensional codes, and this generalization applies to any arbitrary 1-dimensional code. Thus Reed-Solomon codes can also be described as array codes. The following example shows an array code representation of a (7,2,6) Reed-Solomon code.

Example 6.1 *Array representation of (7,2,6) Reed-Solomon code*

Let α be a root of the primitive binary polynomial $x^3 + x + 1$ that generates the Galois field GF(8). Using 1, α and α^2 as a basis, the 8 elements of GF(8) can be represented as vectors:

0	α	α^2	α^3	α^4	α^5	α^6	1
000	010	001	110	011	111	101	100

Now a (7,2,6) Reed-Solomon code can be constructed using the following *generator polynomial* [19]:

$$g(x) = (x + \alpha)(x + \alpha^2)(x + \alpha^3)(x + \alpha^4)(x + \alpha^5)$$

i.e.,

$$g(x) = x^5 + \alpha^2 x^4 + (1 + \alpha)x^3 + (1 + \alpha^2)x^2 + (\alpha + \alpha^2)x + \alpha$$

Any information to be encoded can be represented by an information polynomial of degree 1, i.e.,

$$m(x) = (a_1 + a_2\alpha + a_3\alpha^2)x + (b_1 + b_2\alpha + b_3\alpha^2)$$

where the information bits, the a_i's and the b_i's, are in GF(2), i.e., they are binary. The above information polynomial can then be described as an array of size 3×2 over GF(2):

a_1	b_1
a_2	b_2
a_3	b_3

The code polynomial is then of degree 6:

$$c(x) = m(x)g(x)$$

After calculating and simplifying $c(x)$, a codeword of the (7,2,6) Reed-Solomon code can then be represented by an array of size 3×7, in a way similar to the representation of the information polynomial:

a_1	a_2+b_1	$a_1+a_3+b_2$	$a_1+a_2+b_1+b_3$	$a_2+a_3+b_1+b_2$	$a_3+b_2+b_3$	b_3
a_2	$a_2+a_3+b_2$	$a_1+a_2+a_3+b_2+b_3$	$a_3+b_1+b_2+b_3$	$a_1+a_2+b_3$	$a_1+a_3+b_1+b_2$	b_1+b_3
a_3	$a_1+a_3+b_3$	$a_2+a_3+b_1+b_3$	$a_1+b_2+b_3$	$a_1+a_2+a_3+b_1$	$a_2+b_1+b_2+b_3$	b_2

Table 6.1: An array representation of a (7,2,6) Reed-Solomon code. Total number of additions: 39.

□

Representing Reed-Solomon codes using arrays alone does *not* simplify encoding operations. Further simplifications need to be done. For the (7,2,6) Reed-Solomon code in Table 6.1, the last column certainly can be simplified to (b_3, b_1, b_2) instead of (b_3, b_1+b_3, b_2), without changing the code's MDS property, since the two vectors span the same space. We can simplify all other columns in a similar way, so that the density of each column is reduced to its minimum while the space spanned by the column remains unchanged. The following array is a simplified form of the (7,2,6) Reed-Solomon code, derived from its array form in the above example:

a_1	a_2+b_1	$a_1+a_3+b_2$	$a_2+a_3+b_3$	$a_2+a_3+b_1+b_2$	$a_3+b_2+b_3$	b_3
a_2	$a_2+a_3+b_2$	a_2+b_3	$a_1+a_3+b_1$	$a_1+a_2+b_3$	$a_1+b_1+b_3$	b_1
a_3	$a_1+a_3+b_3$	a_3+b_1	$a_1+b_2+b_3$	a_1+b_2	$a_1+a_2+b_2$	b_2

Table 6.2: A simplified array representation of a (7,2,6) Reed-Solomon code. Total number of additions: 27.

Though the above array form has been simplified a lot, i.e., its encoding operation is simpler than the original Reed-Solomon code, it can be further simplified as in Table 6.3 without changing the update complexity, i.e., some intermediate parity bits (s_i's) are calculated once and then reused in calculating other parity bits.

So this gives a way to design more MDS array codes with more choices of length and distance, based on Reed-Solomon codes. Of course, this method should apply with *any* other linear code that is not MDS, i.e., any linear code can be described by a simplified array code with simple encoding operations.

Even though the array in Table 6.3 has been simplified a lot, it is still not clear whether it is the *optimal* form in terms of encoding operations, since we can simplify multiple columns

a_1	s_2	a_1+s_3	a_3+s_5	s_2+s_3	b_3+s_3	b_3
a_2	a_2+s_3	s_5	a_1+s_4	a_2+s_1	b_1+s_1	b_1
a_3	a_3+s_1	s_4	b_2+s_1	s_6	a_2+s_6	b_2

Table 6.3: A further simplified array representation of a (7,2,6) Reed-Solomon code, where $s_1 = a_1 + b_3$, $s_2 = a_2 + b_1$, $s_3 = a_3 + b_2$, $s_4 = a_3 + b_1$, $s_5 = a_2 + b_3$, and $s_6 = a_1 + b_2$. Total number of additions: 17.

at the same time as long as these columns span the same space. There are many research problems to solve, related to representing arbitrary codes as array codes. To name a few: 1) given a linear code, what is its optimal array code representation in terms of encoding complexity? or, how does one determine whether an array description is optimal in terms of its component density, i.e., total number of information bits appearing? 2) how does one design an efficient erasure-correcting algorithm once an array code is derived from a Reed-Solomon code or from another code? 3) how does one design efficient *multiple* error-correcting algorithms for an array code?

6.2.2 Strong MDS Codes

For an MDS (l,k) array code of size $n \times l$, its MDS property means that the nk original information bits can be recovered from any k columns with each containing n bits. As shown in Chapter 5, if an (l, k) MDS array code is used in a data distribution scheme, a matching data acquisition scheme may read nk bits from m ($m \geq k$) servers, with the ith server sending a_i bits such that $\sum_{i=1}^{m} a_i = nk$, where of course $0 \leq a_i \leq n$ and $1 \leq i \leq m$. Mapping to array codes, this leads to a new MDS property, which is called the *strong MDS property*, as defined as follows:

Definition 6.1 *(strong MDS property) An array code of size $n \times l$ with nk raw information bits has the* strong MDS property, *if, for any given m columns and any given series of positive integers a_i, where $k \leq m \leq l$, $0 \leq a_i \leq n$, $1 \leq i \leq m$ and $\sum_{i=1}^{m} a_i = nk$, there* always *exist a_i bits from the ith column such that the nk raw information bits can be reconstructed from these nk bits from the m columns.*

The interested reader can verify that the Reed-Solomon code in Table 6.2 has the strong MDS property. For example, if $m = 3$, and we are given the first 3 columns, let $a_1 = a_2 =$

$a_3 = 2$, then we can choose either of the following 6 bits from the the 3 columns with 2 bits from each column:

a_1	$a_2 + b_1$	$a_2 + b_3$
a_2	$a_2 + a_3 + b_2$	$a_3 + b_1$

a_1	$a_2 + b_1$	$a_2 + b_3$
a_3	$a_2 + a_3 + b_2$	$a_3 + b_1$

Codes with the strong MDS property can provide more choices of which data to read from multiple servers in an optimal data acquisition scheme. They can also provide more flexibility between redundancy and efficiency when distributing data. So codes with the strong MDS property will have useful roles in many applications in distributed storage systems. From the definition, any code with the strong MDS property is of course MDS.

A final question is how to construct codes with the strong MDS property? Since any code can be represented in array form, we answer this question by the following conjecture and conclude this thesis:

Conjecture 6.1 *(Strong MDS Conjecture)*

All MDS codes have the strong MDS property.