BASIC Regenerating Code: Binary Addition and Shift for Exact Repair

Hanxu Hou[†§], Kenneth W. Shum[‡], Minghua Chen[§] and Hui Li[†*]
[†] Shenzhen Eng. Lab of Converged Networks Technology, Shenzhen Key Lab of Cloud Computing Tech. & App.
Shenzhen Graduate School, Peking University, Shenzhen, 518055, China.
[‡] Institute of Network Coding, the Chinese University of Hong Kong
[§] Department of Information Engineering, the Chinese University of Hong Kong

***Abstract*— Regenerating code is a class of storage codes that achieve the optimal trade-off between storage capacity and repair bandwidth, which are two important performance metrics in data storage systems. However, existing constructions of regenerating codes rely on expensive computational operations such as finite field multiplication. The high coding and repair complexity limit their applications in large-scale practical storage systems. In this paper, we show that it is possible to achieve the full potential of regenerating codes with low computational complexity. In particular, we propose a new class of regenerating codes, called BASIC codes, that can achieve two specific points (i.e., minimum-bandwidth and minimum-storage regenerating points) on the storage and repair bandwidth trade-off curve, using only binary addition and shift operations in the coding and repair processes. Although in this paper we focus on constructing and analyzing BASIC codes for two specific exact-repair settings, our framework can be generalized to develop BASIC codes for more general exact- and functional-repair regenerating codes.**

***Index Terms*—Distributed storage system, regenerating codes, convolutional codes.**

I. Introduction

Online data storage is an increasingly popular service that allows users to access their data anywhere and anytime. Large-scale distributed cloud storage systems are key to the success of such service. Commercial distributed cloud storage systems, such as those run by Google [1] and Facebook [2], usually consist of a large number of inexpensive and individually unreliable storage nodes interconnected via a large distributed network. The systems are unreliable in nature because nodes in the system can leave frequently and unexpectedly due to various reasons like hardware failures.

A redundancy-and-repair paradigm is commonly adopted to store user data reliably in face of system unreliability [3]. In particular, classical erasure codes are employed to exploit the power of redundancy to protect the data from being lost when nodes fail. Here redundancy is measured in terms of storage capacity overhead necessary for data protection, and it is important to minimize the capacity overhead without jeopardizing the data protection capability. Meanwhile, to maintain a target high reliability across time, the system is repaired whenever a node fails by replacing it with a new one. As the node failures in distributed cloud storage systems are “norm rather than exceptional” [1], keeping the repair process efficient by minimizing the amount of data that a new node needs to download to repair the system (called repair bandwidth) is also a critical system design objective.

There is a fundamental trade-off between storage capacity overhead and repair bandwidth in designing distributed cloud storage systems, as revealed by the authors in [4]. They also introduced a new class of codes, called *regenerating codes*, that achieve any point in this *capacity-bandwidth* trade-off curve. There has been a large body of follow-up works in the literature since then.

While existing regenerating codes can attain any point in the optimal capacity-bandwidth trade-off curve, computational complexity should be reduced for practical implementation. In [5], the minimum-bandwidth regenerating code in [6] is implemented, and the performance is compared with traditional RAID-5 and RAID-6. The complexity of product-matrix construction in [7] is studied in [8]. In [9], the implementation of a new coding scheme for Hadoop Distributed File System (HDFS) is reported. The importance of designing a low-complexity scheme is highlighted in these works. These observations naturally lead to the following open question: *Can one achieve the full benefit of regenerating codes with the minimum computational complexity*?

There have been several recent works that provide partial answers to the above question. In [10], [11], the authors proposed a class of minimum-bandwidth regenerating (MBR) code that is formed by the concatenation of two component codes: an outer maximum-distance separable (MDS) code to ensure the repaired MDS property for distributed storage system, and an inner repetition code characterized by an efficient and uncoded repair process with repetition degree ρ that can tolerate up to $\rho-1$ nodes failing together. In [12], they formulated a repair-by-transfer problem based on functional minimum-storage regenerating (FMSR) codes and provide a deterministic FMSR code construction that can significantly speed up the repair process. However, in all of the works

This work was partially supported by the National Basic Research Program of China (No. 2012CB315904), NSFC61179028, General Research Funds No. 411011, and by a grant from the University Grants Committee of the Hong Kong Special Administrative Region, China (Project No. AoE/E-02/08).

[*] Corresponding author.

mentioned above, finite field arithmetic is involved. As finite field multiplication is very often the bottleneck in the design of algorithms for coding and cryptography, applications of regenerating codes in large-scale practical storage systems is a challenging design issue.

Encouraged by these exciting progress in [10], [11], [12], we further explore answers to the question. In this paper, we propose a new class of low-complexity regenerating codes called BASIC codes (in short of *Binary Addition and Shift Implementable Convolutional* codes). There are two distinctive features in the proposed codes. Firstly, the concepts of "striping" and "data chunk" are not required. The data file to be stored is first divided into a small number of bit streams, before encoding is carried out. This introduces a new paradigm for the design of regenerating codes. Secondly, finite field multiplication is replaced by bitwise shifting the bit streams. Expensive finite field multiplication is avoided. This leads to low-complexity design of regenerating codes. Similar design is presented in [13], but repair bandwidth is not considered in [13].

For two exact-repair settings (i.e., exact minimum bandwidth repair and exact minimum storage repair), we show how to construct BASIC codes by applying a new *convolutional design framework* to existing product-matrix code construction [7]. The resulting BASIC codes use only binary addition and shift operations in their coding and repair processes and thus have very low computational complexity. We further show that the constructed BASIC codes achieve the optimal capacity-bandwidth trade-off asymptotically as size of the stored file goes to infinity.

Our convolutional design framework was partially inspired by the MDS convolutional code [14]. We expands its usefulness to the regenerating code design by taking the repair bandwidth into consideration. Although in this paper we focus on two specific exact-repair settings, our framework can be generalized to construct BASIC codes for more general exact- and functional-repair scenarios.

This paper is organized as follows. We review some preliminaries of formal power series and its application in convolutional codes in Section II. A simple example that illustrates the main ideas is given in Section III. In Section IV, the two product-matrix constructions in [7] are adapted to BASIC regenerating codes, and the computational complexity of encoding and node repair is discussed.

II. Preliminaries

A. Formal Power Series and Polynomials

Let a and b be two infinite sequences of bits,

$$\mathsf{a} = (a_0, a_1, a_2, a_3, \ldots), \quad \mathsf{b} = (b_0, b_1, b_2, b_3, \ldots).$$

We can define addition and multiplication on infinite sequences. The sum of a and b is performed by adding the corresponding components using $\mathbb{F}_2$ arithmetic,

$$\mathsf{a} + \mathsf{b} := (a_0 + b_0, a_1 + b_1, a_2 + b_2, \ldots).$$

The product of a and b is defined by "convolution",

$$\mathsf{a} \cdot \mathsf{b} := \big(a_0 b_0, a_0 b_1 + a_1 b_0, \ldots, \sum_{\ell=0}^{j} a_\ell b_{j-\ell}, \ldots\big).$$

A formal power series is just another way of writing an infinite sequence,

$$(a_0, a_1, a_2, a_3, \ldots) \Leftrightarrow a(z) = a_0 + a_1 z + a_2 z^2 + a_3 z^3 + \ldots.$$

The j-th bit in the infinite sequence on the left is the coefficient of the j-th term on the right. An advantage of representing an infinite sequence in the form of a formal power series is that the addition and multiplication operations can be carried out naturally. The indeterminate z represents the *right-shift operator*, as we have

$$z(a_0 + a_1 z + a_2 z^2 + a_3 z^3 + \ldots) = 0 + a_0 z + a_1 z^2 + a_2 z^3 + \ldots$$

A formal power series with finitely many nonzero terms is nothing but a polynomial. The set of all formal power series with coefficients in $\mathbb{F}_2$ is denoted by $\mathbb{F}_2[[z]]$, and the set of polynomials over $\mathbb{F}_2$ is denoted by $\mathbb{F}_2[z]$. In this paper, the coefficients of polynomials or formal power series take values in $\mathbb{F}_2$.

We can divide a power series $b(z)$ by $a(z)$ if the constant term of $a(z)$ is equal to 1. If $a_0 = 1$, we can define the quotient $b(z)/a(z)$ as a formal power series $f(z) = \sum_j f_j z^j$ whose coefficients are obtained by equating the coefficients on both sides of

$$\sum_{j=0}^{\infty} \Big(\sum_{\ell=0}^{j} a_\ell f_{j-\ell}\Big) z^j = \sum_{j=0}^{\infty} b_j z^j.$$

The coefficients of $f(z)$ can be computed recursively by

$$f_j = b_j + \sum_{\ell=1}^{j} a_\ell f_{j-\ell} \tag{1}$$

for $j = 0, 1, 2, \ldots$.

B. Convolutional Coding

We identify the bits in a data packet with the coefficients of a formal power series. When we say "a node stores a formal power series", we mean the coefficients of the formal power series are stored. We will use the terms "packet", "power series" and "formal power series" interchangeably. In practice, the total number of bits in a packet is a large but finite number. A packet is thus represented by a polynomial in $\mathbb{F}_2[z]$ with very large degree.

Let $s_1(z)$, $s_2(z), \ldots, s_k(z)$ be the source packets. A *coded packet* is a linear combination of the source packets,

$$c_1(z)s_1(z) + c_2(z)s_2(z) + \ldots + c_k(z)s_k(z).$$

The $c_i(z)$'s are polynomials in $\mathbb{F}_2[z]$ and are called the *global encoding coefficients* of the coded packet. The vector

$$[c_1(z) \; c_2(z) \; \cdots \; c_k(z)],$$

with the j-th component equal to the polynomial $c_j(z)$, is called the *global encoding vector*. For example, let $a(z)$ and

$b(z)$ be two source packets, each containing L bits. The bits in the coded packet $a(z) + zb(z)$ are

$$(a_0, a_1 + b_0, a_2 + b_1, a_3 + b_2, \dots, a_{L-1} + b_{L-2}, b_{L-1}),$$

and the global coding vector is $[1\ z]$. Due to bitwise shifting, the length of the coded packet is L plus the maximal degree of the global encoding coefficients. The maximal degree of the coefficients is thus the storage overhead. This overhead is negligible when the packet size L is large.

Proposition 1. *Suppose that $s_1(z)$ to $s_k(z)$ are source packets, and $p_1(z)$ to $p_k(z)$ are coded packets with global encoding vectors v_1 to v_k, respectively. Let $\mathbf{E}$ be the $k \times k$ matrix whose rows are precisely the global encoding vectors. Then we can compute $s_1(z)$ to $s_k(z)$ from $p_1(z)$ to $p_k(z)$ provided that the determinant of $\mathbf{E}$ is a nonzero polynomial.*

Proof. Suppose that $\det(\mathbf{E})$ is a non-zero polynomial with indeterminate z. We can write $\det(\mathbf{E})$ as $z^e(1 + f(z))$, where e is a non-negative integer and $f(z)$ is a polynomial with zero constant term, so that the multiplicative inverse of $1 + f(z)$ is well-defined in $\mathbb{F}_2[[z]]$. Recall that the adjoint of $\mathbf{E}$, denoted by $adj(\mathbf{E})$, is defined as the transpose of the matrix of cofactors of $\mathbf{E}$ [15, p.353], and has the property that

$$adj(\mathbf{E}) \cdot \mathbf{E} = \det(\mathbf{E})\mathbf{I}.$$

We can compute $z^e s_1(z)$ to $z^e s_k(z)$ by

$$\begin{bmatrix} z^e s_1(z) \\ z^e s_2(z) \\ \vdots \\ z^e s_k(z) \end{bmatrix} = (1 + f(z))^{-1} \cdot adj(\mathbf{E}) \cdot \begin{bmatrix} p_1(z) \\ p_2(z) \\ \vdots \\ p_k(z). \end{bmatrix}.$$

Then $s_1(z)$ to $s_k(z)$ can be obtained by shifting $z^e s_1(z)$, $z^e s_2(z), \dots, z^e s_k(z)$ to the left by e bits. □

Some remarks on implementation are in order. It is more convenient to perform byte-wise shift instead of bit-wise shift. We can further subdivide a bit stream into 8 sub-streams, and store them in an interleaving manner, so that the first bit in a byte belongs to the first sub-stream, the second bit in a byte belongs to the second sub-stream, and so on. Shifting a byte is equivalent to shifting the 8 sub-streams simultaneously. Also, it is not necessary to physically shift the bits in the memory. By creating pointers to the data bits, the shifting operations are performed by simply incrementing the pointers.

III. An Illustrating Example

We present a simple example similar to the example given by Wu and Dimakis in [16]. There are four source packets $s_1(z)$ to $s_4(z)$. Suppose that each packet carries L bits, i.e., the packets are represented by polynomials with degree strictly less than L. Nodes 1 and 2 are systematic nodes, while nodes 3 and 4 are parity-check nodes. The encoding of data is given in Table I.

TABLE I
An Example of BASIC Regenerating Codes for $n = 4$ Nodes.

Node	Packets
1	$s_1(z)$, $s_2(z)$
2	$s_3(z)$, $s_4(z)$
3	$s'_1(z) = s_1(z) + s_3(z)$, $s'_2(z) = s_2(z) + zs_4(z)$
4	$s'_3(z) = s_1(z) + zs_3(z)$, $s'_4(z) = s_2(z) + s_4(z)$

We first check that the MDS property is satisfied. Suppose that a data collector connects to one systematic node and one parity-check node, say nodes 2 and 3. The packets $s_3(z)$ and $s_4(z)$ are available in uncoded form, and $s_1(z)$ and $s_2(z)$ can be computed by adding $s'_1(z) + s_3(z)$ and $s'_2(z) + zs_4(z)$.

Suppose that a data collector connects to nodes3 and 4. From packets $s_1(z) + s_3(z)$ and $s_1(z) + zs_3(z)$, the data collector computes

$$z(s_1(z) + s_3(z)) + s_1(z) + zs_3(z) = (z + 1)s_1(z).$$

The coefficients of $s_1(z)$ can be obtained from $(z+1)s_1(z)$, by the recursive formula in (1). The other source packets $s_2(z)$, $s_3(z)$ and $s_4(z)$ can be obtained similarly.

We note that there is a symmetry in the encoding. The packets $s_1(z)$ to $s_4(z)$ in nodes 1 and 2 can be obtained from $s'_1(z)$ to $s'_4(z)$ in nodes 3 and 4 by

$$\begin{aligned} s_1(z) &:= (1 + z)^{-1}(zs'_1(z) + s'_3(z)) \\ s_2(z) &:= (1 + z)^{-1}(s'_2(z) + zs'_4(z)) \\ s_3(z) &:= (1 + z)^{-1}(s'_1(z) + s'_3(z)) \\ s_4(z) &:= (1 + z)^{-1}(s'_1(z) + s'_3(z)). \end{aligned}$$

We can alternately treat nodes 3 and 4 as the systematic nodes and nodes 1 and 2 as the parity-check node.

Because of the symmetry, it is sufficient to see how to repair nodes 1. Suppose that node 1 fails, and we want to repair it by downloading one packet from each of the surviving nodes. Node 2 sends the sum

$$s_3(z) + s_4(z)$$

to the new node. Node 3 shifts packet $s_1(z) + s_3(z)$ to the right by one bit, adds to $s_2(z)+zs_4(z)$, and sends the resulting packet

$$zs_1(z) + s_2(z) + z(s_3(z) + s_4(z))$$

to the new node. Node 3 shifts packet $s_2(z) + s_4(z)$ to the right by one bit, adds to $s_1(z)+zs_3(z)$, and sends the resulting packet

$$s_1(z) + zs_2(z) + z(s_3(z) + s_4(z)).$$

The new node can remove $s_3(z) + s_4(z)$ and obtain $zs_1(z) + s_2(z)$ and $s_1(z) + zs_2(z)$. Next, the new node computes

$$\begin{aligned} z(zs_1(z) + s_2(z)) + s_1(z) + zs_2(z) &= (1 + z^2)s_1(z) \\ zs_1(z) + s_2(z) + z(s_1(z) + zs_2(z)) &= (1 + z^2)s_2(z), \end{aligned}$$

from which $s_1(z)$ and $s_2(z)$ can be decoded by (1).

The number of bits transmitted in the repair of node 1 is slightly larger than $3L$, due to the shift-and-add operation. The percentage increase of repair bandwidth approaches zero when L approaches infinity. In practice, the value of L is very large. We thus see that the optimal repair bandwidth at the minimum-storage regenerating point is practically attained.

IV. BASIC Product-Matrix Regenerating Codes

We convert the product-matrix minimum-bandwidth regenerating (MBR) and minimum-storage regenerating (MSR) codes in [7] to BASIC regenerating codes.

Let $\mathbf{u} = [s_1(z)\ s_2(z)\ \dots\ s_B(z)]^\top$ be a column vector of length B containing the source packets. (We use superscript $^\top$ to denote the transpose operator.) Each packet contains L bits. The product-matrix construction is specified by two matrices, $\mathbf{\Psi}$ and $\mathbf{M}$. The matrix $\mathbf{\Psi}$ has size $n \times m$ and $\mathbf{M}$ has size $m \times \alpha$. The entries of $\mathbf{\Psi}$ are fixed polynomials and independent of the source symbols. The i-th row of $\mathbf{\Psi}$ is referred to as the encoding vector of node i. The entries of $\mathbf{M}$ are either the source packets or constant, and are regarded as formal power series. For $i = 1, 2, \dots, n$, node i stores the i-th row of the product $\mathbf{\Psi M}$.

A. BASIC Product-Matrix MSR Code

In the following, we construct the BASIC product-matrix MSR code for $d = 2k - 2$. As in [7], the construction can be extended naturally to $d \geq 2k - 2$, but we will only discuss the basic case for $d = 2k - 2$, $\alpha = k - 1$, $B = k\alpha = k(k-1)$.

Divide the data file into B source packets, and divide each of the B source packets into two equal groups. For each group, create a $(k-1) \times (k-1)$ symmetric matrix by filling the upper-triangular part of the matrix by the $k(k-1)/2$ source packets in the group, and obtain the lower-triangular part by reflection. Let the symmetric matrix obtained from group j be denoted by $\mathbf{S}_j$, for $j = 1, 2$, and let $\mathbf{M}$ be the $d \times (k-1)$ matrix

$$\mathbf{M} = \begin{bmatrix} \mathbf{S}_1 \\ \mathbf{S}_2 \end{bmatrix}.$$

The design parameter m is set to d.

Define the encoding matrix $\mathbf{\Psi}$ be the $n \times d$ Vandermonde matrix, with the i-th row defined as

$$\boldsymbol{\psi}_i^\top := \begin{bmatrix} 1 & z^{i-1} & z^{2(i-1)} & \dots & z^{(d-1)(i-1)} \end{bmatrix}, \qquad (2)$$

for $i = 1, 2, \dots, n$. The i-th node stores $\alpha = k - 1$ packets in $\boldsymbol{\psi}_i^\top \mathbf{M}$. The protocol of repairing a failed node is the same as in [7], except that we are now working over $\mathbb{F}_2[[z]]$ instead of finite field.

B. BASIC Product-Matrix MBR Code

We divide the data file into

$$B = \frac{k(k+1)}{2} + k(d-k) \qquad (3)$$

source packets. Each node stores $\alpha = d$ coded packets. Create a $d \times d$ matrix $\mathbf{M}$

$$\mathbf{M} := \begin{bmatrix} \mathbf{S} & \mathbf{T} \\ \mathbf{T}^\top & \mathbf{0} \end{bmatrix}.$$

The matrix $\mathbf{S}$ is a symmetric $k \times k$ matrix obtained by first filling the upper-triangular part by source packets $s_j(z)$, for $j = 1, 2, \dots, k(k+1)/2$, and then obtain the lower-triangular part by reflection along the diagonal. The rectangular matrix $\mathbf{T}$ has size $k \times (d-k)$, and the entries in $\mathbf{T}$ are source packets $s_j(z)$, $j = k(k+1)/2, \dots, B$, listed in some fixed but arbitrary order. The matrix $\mathbf{T}^\top$ is the transpose of $\mathbf{T}$ and the matrix $\mathbf{0}$ is a $(d-k) \times (d-k)$ all-zero matrix. For $i = 1, 2, \dots, n$, let the encoding vector of node i be defined as in (2). Node i stores the d packets in $\boldsymbol{\psi}_i^\top \mathbf{M}$.

We give an example for $n = 6$, $k = 3$, $d = 4$, which contains all the essential features.

There are $B = 9$ source packets $s_1(z)$ to $s_9(z)$. The matrix

$$\mathbf{M} = \left[\begin{array}{ccc|c} s_1(z) & s_2(z) & s_3(z) & s_7(z) \\ s_2(z) & s_4(z) & s_5(z) & s_8(z) \\ s_3(z) & s_5(z) & s_6(z) & s_9(z) \\ \hline s_7(z) & s_8(z) & s_9(z) & 0 \end{array}\right]$$

is a symmetric matrix with entries taken from $\mathbb{F}_2[z]$. For $i = 1, 2, \dots, 6$, the encoding vector of node i is

$$\boldsymbol{\psi}_i^\top = \begin{bmatrix} 1 & z^{i-1} & z^{2(i-1)} & z^{3(i-1)} \end{bmatrix}.$$

For any four distinct node indices i_1, i_2, i_3 and i_4 between 1 to 6, the corresponding encoding vectors $\boldsymbol{\psi}_{i_1}^\top$, $\boldsymbol{\psi}_{i_2}^\top$ $\boldsymbol{\psi}_{i_3}^\top$ and $\boldsymbol{\psi}_{i_4}^\top$ form a non-singular 4×4 Vandermonde matrix.

For $i = 1, 2, \dots, 6$, node i stores the four packets in the i-th row of $\mathbf{\Psi M}$, namely

$$\begin{aligned} &s_1(z) + z^{i-1}s_2(z) + z^{2(i-1)}s_3(z) + z^{3(i-1)}s_7(z), \\ &s_2(z) + z^{i-1}s_4(z) + z^{2(i-1)}s_5(z) + z^{3(i-1)}s_8(z), \\ &s_3(z) + z^{i-1}s_5(z) + z^{2(i-1)}s_6(z) + z^{3(i-1)}s_9(z), \\ &s_7(z) + z^{i-1}s_8(z) + z^{2(i-1)}s_9(z). \end{aligned}$$

Each of the coded packets can be obtained by right-shifting and adding the source packets appropriately.

Suppose that a data collector connects to nodes 1, 2 and 3. We can solve for $s_7(z)$, $s_8(z)$ and $s_9(z)$ from

$$\begin{bmatrix} s_7(z) + s_8(z) + s_9(z) \\ s_7(z) + zs_8(z) + z^2 s_9(z) \\ s_7(z) + z^2 s_8(z) + z^4 s_9(z) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & z & z^2 \\ 1 & z^2 & z^4 \end{bmatrix} \begin{bmatrix} s_7(z) \\ s_8(z) \\ s_9(z) \end{bmatrix}.$$

It can be as in the proof of Prop. 1. Then, $s_1(z)$ to $s_6(z)$ can be decoded from

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & z & z^2 \\ 1 & z^2 & z^4 \end{bmatrix} \begin{bmatrix} s_1(z) & s_2(z) & s_3(z) \\ s_2(z) & s_4(z) & s_5(z) \\ s_3(z) & s_5(z) & s_6(z) \end{bmatrix}.$$

The solution can be computed by the adjoint formula in the proof of Prop. 1 as well.

Suppose node 5 fails, and we want to regenerate it from node 1, 2, 3 and 4. The coded packet sent from helper node i to the newcomer is $\boldsymbol{\psi}_i^\top \mathbf{M} \boldsymbol{\psi}_5$. If we put the packets received by the newcomer as a column vector, then the column vector can be written as

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & z & z^2 & z^3 \\ 1 & z^2 & z^4 & z^6 \\ 1 & z^3 & z^6 & z^9 \end{bmatrix} \cdot \mathbf{M} \cdot \boldsymbol{\psi}_5.$$

Since the matrix on the left has non-zero determinant, the newcomer can compute $\mathbf{M} \cdot \boldsymbol{\psi}_5$ by the adjoint formula in Prop. 1. As $\mathbf{M}$ is symmetric, this is exactly equal to the content of the failed node.

The repair of other nodes can be done similarly.

C. Storage Overheads and Computational Complexity

We give an estimate of the storage overhead and computational complexity of decoding and repairing the BASIC product-matrix MBR code. The computational complexity of BASIC product-matrix MSR code is similar and is omitted. As we have mentioned earlier, we can access the bits in a bit stream by pointers, pointing to the bits we want to access. The shifting operation can performed by incrementing the pointers, and hence is very easy to implement. Hence, we will not count the number of shift operations, but we will give an upper bound on the number of additions over $\mathbb{F}_2$, or equivalently, the number of exclusive or (XOR) operations.

If Vandermonde matrices are used in the BASIC product-matrix MSR and MBR codes, the maximal degree of the encoding coefficients is $(d-1)(n-1)$. Let M be the total number of bits in the data file to be distributed, and L be the size of the largest packets in the storage nodes. Each source packet contains M/B bits, where B is given in (3). The number of bits in each packets stored in the storage nodes is no more than $L < M/B+(d-1)(n-1)$. The storage overhead due to bitwise shifting is negligible if $M \gg B(d-1)(n-1)$.

In the encoding of the BASIC product-matrix MBR code, a coded packet is obtained by adding some shifted versions of the source packets. The total number of XOR operations in computing a coded packet is upper bounded by dL. During the repair of the BASIC product-matrix MBR code, a helping node adds some shifted versions of the coded packets. The total number of XOR operations in each helping node is upper bounded by dL.

For the computational complexity at the new node, we first give an upper bound on the number of bit-wise XOR required in solving a system of $m \times m$ equations $\mathbf{A}\mathbf{w} = \mathbf{b}$. The matrix $\mathbf{A}$ is an $m \times m$ matrix whose entries are polynomials $\mathbb{F}_2[z]$. Suppose that the largest degrees of the entries in $\mathbf{A}$ is D. The vector $\mathbf{w}$ is an $m \times 1$ column matrix whose components are unknown polynomials. The vector $\mathbf{b}$ is an $m \times 1$ column matrix whose components belong to $\mathbb{F}_2[z]$. Suppose that the degrees of the components of $\mathbf{b}$ is no larger than L. We consider the case where the value of L is much larger than the the value of D. We solve for $\mathbf{w}$ using the adjoint formula in the proof of Prop. 1.

The cofactors of $\mathbf{A}$ are polynomials in $\mathbb{F}_2[z]$, whose degrees are strictly less than mD. The m entries in $adj(\mathbf{A})\mathbf{b}$ can be computed by no more than m^3DL XOR's. The computation of the adjoint of $\mathbf{A}$ is independent of the packet size L. Since $L \gg D$, the computational complexity of the calculation of adjoint can be ignored.

The determinant of $\mathbf{A}$ is a polynomial in $\mathbb{F}_2[z]$, whose degree is less than mD. The degree of each components in $adj(\mathbf{A})\mathbf{b}$ is at most $L + mD$. The number of XOR's required in solving $\det(\mathbf{A})\mathbf{w} = adj(\mathbf{A})\mathbf{b}$ by (1) is no more than $mD(L+mD)$. Hence, the number of XOR's in solving $\mathbf{A}\mathbf{w} = \mathbf{b}$ is $O(m^3DL)$.

The new node needs to solve a $d \times d$ system of equations, and it takes $O(d^4nL)$ XOR operations. A data collector need to solve d systems of equations, each of them is $k \times k$. The number of XOR's required in decoding the data file is $O(dk^3(d-1)(n-1)L) = O(d^2k^3nL)$.

V. Summary

A BASIC approach for constructing a new class of regenerating codes is proposed in this paper. This approach has the advantage that finite field multiplication is avoided, by replacing finite field multiplication by bitwise shifting. The constructed codes are vector-linear, with each symbol represented by a polynomial over $\mathbb{F}_2[z]$. The encoding of source data and the operation of the helper nodes during the repair process are especially simple. The low-complexity of the encoding and repair operation is amenable to practical implementation.

References

[1] S. Ghemawat, H. Gobioff, and S. Leung, "The google file system," in *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5. ACM, 2003, pp. 29–43.

[2] N. Ellison, C. Steinfield, and C. Lampe, "The benefits of facebook friends: social capital and college students use of online social network sites," *Journal of Computer-Mediated Communication*, vol. 12, no. 4, pp. 1143–1168, 2007.

[3] S. Rhea, C. Wells, P. Eaton, D. Geels, B. Zhao, H. Weatherspoon, and J. Kubiatowicz, "Maintenance-free global data storage," *IEEE Internet Computing*, vol. 5, no. 5, pp. 40–49, 2001.

[4] A. Dimakis, P. Godfrey, Y. Wu, M. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," *IEEE Trans. Inf. Theory*, vol. 56, no. 9, pp. 4539–4551, Sep. 2010.

[5] Y. Hu, C.-M. Yu, Y. K. Li, P. P. C. Lee, and J. C. S. Lui, "NCFS: On the practicality and extensibility of a network-coding-based distributed file system," in *Proc. of Int. Symp. on Network Coding (Netcod)*, Beijing, Jul. 2011, pp. 1–6.

[6] N. B. Shah, K. V. Rashmi, P. V. Kumar, and K. Ramchandran, "Distributed storage codes with repair-by-transfer and non-achievability of interior points on the storage-bandwidth tradeoff," *IEEE Trans. Inf. Theory*, vol. 58, no. 3, pp. 1837–1852, Mar. 2012.

[7] K. V. Rashmi, N. B. Shah, and P. V. Kumar, "Optimal exact-regenerating codes for distributed storage at the MSR and MBR points via a product-matrix construction," *IEEE Trans. Inf. Theory*, vol. 57, no. 8, pp. 5227–5239, Aug. 2011.

[8] S. Jiekak, A.-M. Kermarrec, N. Le Scouarnec, G. Straub, and A. van Kempen, "Regenerating codes: A system perspective," arXiv:1204.5028 [cs.DC], 2012.

[9] M. Sathiamoorthy, M. Asteris, D. Papailiopoulos, A. G. Dimakis, R. Vadali, S. Chen, and D. Borthakur, "XORing elephants: Novel erasure codes for big data," in *Proc. of the 39th Int. Conf. on Very Large Data Bases*, Trento, Aug. 2013.

[10] S. El Rouayheb and K. Ramchandran, "Fractional repetition codes for repair in distributed storage systems," in *the 48th Annual Allerton Conference on Communication, Control, and Computing (Allerton),*, 2010, pp. 1510–1517.

[11] S. Pawar, N. Noorshams, S. El Rouayheb, and K. Ramchandran, "Dress codes for the storage cloud: Simple randomized constructions," in *Proc. IEEE Int. Symp. Inf. Theory*, 2011, pp. 2338–2342.

[12] Y. Hu, P. P. C. Lee, and K. W. Shum, "Analysis and construction of functional regenerating codes with uncoded repair for distributed storage systems," in *Proc. IEEE Int. Conf. on Computer Comm. (INFOCOM)*, Turin, Apr. 2013, pp. 1–6.

[13] J. Qureshi, C. H. Foh, and J. Cai, "Optimal solution for the index coding problem using network coding over GF(2)," in *Proc. IEEE Conf. on Sensor Mesh and Ad Hoc Comm. and Networks (SECON)*, Seoul, Jun. 2012, pp. 1–5.

[14] P. Piret and T. Krol, "MDS convolutional codes," *IEEE Trans. Inf. Theory*, vol. 29, no. 2, pp. 224–232, Mar. 1983.

[15] T. W. Hungerford, *Algebra*, 2nd ed. New York: Springer-Verlag, 1974.

[16] Y. Wu and A. G. Dimakis, "Reducing repair traffic for erasure coding-based storage via interference alignment," in *Proc. IEEE Int. Symp. Inf. Theory*, Seoul, Jul. 2009, pp. 2276–2280.