Update-Efficient Regenerating Codes with Minimum Per-Node Storage

Yunghsiang S. Han*, Hung-Ta Pai†, Rong Zheng‡ and Pramod K. Varshney§
*Dep. of Electrical Eng. National Taiwan University of Science and Technology, Taipei, Taiwan
†Dep. of communication Eng. National Taipei University, Taipei, Taiwan
‡Dep. of Computing and Software, McMaster University, Hamilton, ON, Canada
§Dep. of EECS, Syracuse University, Syracuse, USA

***Abstract*—Regenerating codes provide an efficient way to recover data at failed nodes in distributed storage systems. It has been shown that regenerating codes can be designed to minimize the per-node storage (called MSR) or minimize the communication overhead for regeneration (called MBR). In this work, we propose a new encoding scheme for $[n, d]$ error-correcting MSR codes that generalizes our earlier work on error-correcting regenerating codes. We show that by choosing a suitable diagonal matrix, any generator matrix of the $[n, \alpha]$ Reed-Solomon (RS) code can be integrated into the encoding matrix. Hence, MSR codes with the least update complexity can be found. An efficient decoding scheme is also proposed that utilizes the $[n, \alpha]$ RS code to perform data reconstruction. The proposed decoding scheme has better error correction capability and incurs the least number of node accesses when errors are present.**

I. Introduction

Cloud storage is gaining popularity as an alternative to enterprise storage where data is stored in virtualized pools of storage typically hosted by third-party data centers. Reliability is a key challenge in the design of distributed storage systems that provide cloud storage. Both crash-stop and Byzantine failures (as a result of software bugs and malicious attacks) are likely to be present during data retrieval. A crash-stop failure makes a storage node unresponsive to access requests. In contrast, a Byzantine failure responds to access requests with erroneous data. To achieve better reliability, one common approach is to replicate data files on multiple storage nodes in a network. Erasure coding is employed to encode the original data and then the encoded data is distributed to storage nodes. Typically, more than one storage nodes need to be accessed to recover the original data. One popular class of erasure codes is the maximum-distance-separable (MDS) codes. With $[n, k]$ MDS codes such as Reed-Solomon (RS) codes, k data items are encoded and then distributed to and stored at n storage nodes. A user or a data collector can retrieve the original data by accessing *any* k of the storage nodes, a process referred to as *data reconstruction*.

Any storage node can fail due to hardware or software damage. Data stored at the failed nodes need to be recovered (regenerated) to remain functional to perform data reconstruction. The process to recover the stored (encoded) data at a storage node is called *data regeneration*. *Regenerating codes* first introduced in the pioneering works by Dimakis *et al.* in [1], [2] allow efficient data regeneration. To facilitate data regeneration, each storage node stores α symbols and a total of d surviving nodes are accessed to retrieve $\beta \le \alpha$ symbols from each node. A trade-off exists between the storage overhead and the regeneration (repair) bandwidth needed for data regeneration. Minimum Storage Regenerating (MSR) codes first minimize the amount of data stored per node, and then the repair bandwidth, while Minimum Bandwidth Regenerating (MBR) codes carry out the minimization in the reverse order. There have been many works that focus on the design of regenerating codes [3]–[10]. Recently, Rashmi *et al.* proposed optimal exact-regenerating codes that recover the stored data at the failed node exactly (and thus the name exact-regenerating) [10]; however, the authors only consider crash-stop failures of storage nodes. Han *et al.* extended Rashmi et al.'s work to construct error-correcting regenerating codes for exact regeneration that can handle Byzantine failures [11]. In [11], the encoding and decoding algorithms for both MSR and MBR error-correcting codes were also provided. Specifically, the decoding algorithm of an $[n, k, d]$ MBR code given in [11] has an error correction capability of $\lfloor \frac{n-k+1}{2} \rfloor$. In [12], the code capability and resilience were discussed for error-correcting regenerating codes. The authors also discovered that it is possible to decode an $[n, k, d]$ MBR code up to $\lfloor \frac{n-k+1}{2} \rfloor$ errors, and further claimed that any $[n, k, d \ge 2k - 2]$ MSR code can be decoded up to $\lfloor \frac{n-k+1}{2} \rfloor$ errors. However, no explicit decoding (or data reconstruction) procedure was provided giving rise to the open problem as to whether efficient decoding algorithms can be devised with such an error correction capability. In this paper, we address this problem and develop a decoding algorithm that achieves the stated error correction capability.

In addition to bandwidth efficiency and error correction capability, another desirable feature for regenerating codes is *update complexity* [13], defined as the maximum number of encoded symbols that must be updated while a single data symbol is modified. Low update complexity is desirable in scenarios where updates are frequent. Clearly, the update complexity of a regenerating code is determined by the number of non-zero elements in the row of the encoding matrix with the maximum Hamming weight. The smaller the number, the lower is the update complexity. The update efficiency of MDS codes is discussed in [14], [15]. Update-efficient regenerating codes were presented in [16], [17]. However, existing update-efficient codes only handle crash-stop (erasure) failures and do not handle Byzantine failures, which is the main focus of this paper.

We summarize the contribution of this work as follows. First, an efficient and practical decoding procedure is given for an $[n, k, d = 2k-2]$ MSR code. Second, we propose a general encoding scheme for MSR codes with least-update-complexity. Third, the assumption in [12] that all α symbols are erroneous for each compromised node is eliminated and the proposed decoder can handle any number of errors at the compromised nodes.

II. ERROR-CORRECTING MSR REGENERATING CODES

In this section, we give a brief overview of data regenerating codes and the MSR code construction presented in [11].

A. Regenerating Codes

Let α be the number of symbols stored at each storage node and $\beta \leq \alpha$ the number of symbols downloaded from each storage during regeneration. To repair the stored data at the failed node, a helper node accesses d surviving nodes. The design of regenerating codes ensures that the total regenerating bandwidth be much less than that of the original data, B. A regenerating code must be capable of reconstructing the original data symbols and regenerating coded data at a failed node. An $[n, k, d]$ regenerating code requires at least k and d surviving nodes to ensure successful data reconstruction and regeneration [10], respectively, where n is the number of storage nodes and $k \leq d \leq n-1$.

The cut-set bound given in [2], [3] provides a constraint on the repair bandwidth. By this bound, any regenerating code must satisfy the following inequality:

$$B \leq \sum_{i=0}^{k-1} \min\{\alpha, (d-i)\beta\} \ . \tag{1}$$

From (1), α or β can be minimized achieving either the minimum storage requirement or the minimum repair bandwidth requirement, but not both. The two extreme points in (1) are referred to as the minimum storage regeneration (MSR) and minimum bandwidth regeneration (MBR) points, respectively. The values of α and β for the MSR point can be obtained by first minimizing α and then minimizing β:

$$\begin{aligned} \alpha &= d-k+1 \\ B &= k(d-k+1) = k\alpha \ , \end{aligned} \tag{2}$$

where we normalize β as 1.[1]

There are two types of approaches to regenerate data at a failed node. If the replacement data is exactly the same as that previously stored at the failed node, we call it the *exact regeneration*. Otherwise, if the replacement data only guarantees the correctness of data reconstruction and regeneration properties, it is called *functional regeneration*. In practice, exact regeneration is more desirable since in this case there is no need to inform each node in the network regarding the replacement. Furthermore, it is easy to keep the codes systematic via exact regeneration, where partial data can be retrieved without accessing all k nodes. The codes designed in [10], [11] allow exact regeneration.

B. MSR Regenerating Codes With Error Correction Capability

Next, we describe the MSR code construction given in [11]. In the rest of the paper, we assume $d = 2\alpha$. The information sequence $\boldsymbol{m} = [m_0, m_1, \ldots, m_{B-1}]$ can be arranged into an information vector $U = [Z_1 Z_2]$ with size $\alpha \times d$ such that Z_1 and Z_2 are symmetric matrices with dimension $\alpha \times \alpha$. An $[n, d = 2\alpha]$ RS code is adopted to construct the MSR code [11]. Let a be a generator of $GF(2^m)$. In the encoding of the MSR code, we have

$$U \cdot G = C, \tag{3}$$

where

$$\begin{aligned} G &= \\ &\begin{bmatrix} 1 & 1 & \cdots & 1 \\ a^0 & a^1 & \cdots & a^{n-1} \\ (a^0)^2 & (a^1)^2 & \cdots & (a^{n-1})^2 \\ & & \vdots & \\ (a^0)^{\alpha-1} & (a^1)^{\alpha-1} & \cdots & (a^{n-1})^{\alpha-1} \\ (a^0)^{\alpha}1 & (a^1)^{\alpha}1 & \cdots & (a^{n-1})^{\alpha}1 \\ (a^0)^{\alpha}a^0 & (a^1)^{\alpha}a^1 & \cdots & (a^{n-1})^{\alpha}a^{n-1} \\ (a^0)^{\alpha}(a^0)^2 & (a^1)^{\alpha}(a^1)^2 & \cdots & (a^{n-1})^{\alpha}(a^{n-1})^2 \\ & & \vdots & \\ (a^0)^{\alpha}(a^0)^{\alpha-1} & (a^1)^{\alpha}(a^1)^{\alpha-1} & \cdots & (a^{n-1})^{\alpha}(a^{n-1})^{\alpha-1} \end{bmatrix} \\ &= \begin{bmatrix} \bar{G} \\ \bar{G}\Delta \end{bmatrix}, \end{aligned} \tag{4}$$

and C is the codeword vector with dimension $(\alpha \times n)$. $\bar{G}$ contains the first α rows in G and Δ is a diagonal matrix with $(a^0)^\alpha$, $(a^1)^\alpha$, $(a^2)^\alpha, \ldots,$ $(a^{n-1})^\alpha$ as diagonal elements. Note that if the RS code is over $GF(2^m)$ for $m \geq \lceil \log_2 n\alpha \rceil$, then it can be shown that $(a^0)^\alpha$, $(a^1)^\alpha$, $(a^2)^\alpha, \ldots,$ $(a^{n-1})^\alpha$ are all distinct. After encoding, the ith column of C is distributed to storage node i for $1 \leq i \leq n$.

III. ENCODING SCHEMES FOR ERROR-CORRECTING MSR CODES

RS codes are known to have very efficient decoding algorithms and exhibit good error correction capability. From (4) in Section II-B, a generator matrix G for MSR codes needs to satisfy:

1) $G = \begin{bmatrix} \bar{G} \\ \bar{G}\Delta \end{bmatrix}$, where $\bar{G}$ contains the first α rows in G and Δ is a diagonal matrix with distinct elements in the diagonal.
2) $\bar{G}$ is a generator matrix of the $[n, \alpha]$ RS code and G is a generator matrix of the $[n, d = 2\alpha]$ RS code.[2]

OrNext, we present a sufficient condition for $\bar{G}$ and Δ such that G is a generator matrix of an $[n, d]$ RS code.

[1] It has been proved that when designing $[n, k, d]$ MSR codes for $k/(n+1) \leq 1/2$, it suffices to consider those codes with $\beta = 1$ [10].

[2] The construction of $[n, d]$ MSR code given in [10] only requires that every α columns of $\bar{G}$ and every d columns of G are linearly independent. The usage of RS codes was first proposed in [11].

Theorem 1: Let $\bar{G}$ be a generator matrix of the $[n,\alpha]$ RS code C_α that is generated by the generator polynomial with roots $a^1, a^2, \ldots, a^{n-\alpha}$. Let the diagonal elements of Δ be $(a^0)^\alpha$, $(a^1)^\alpha$, ..., $(a^{n-1})^\alpha$, where $m \geq \lceil \log_2 n \rceil$ and $\gcd(2^m-1, \alpha) = 1$. Then G is a generator matrix of $[n,d]$ RS code C_d that is generated by the generator polynomial with roots $a^1, a^2, \ldots, a^{n-d}$.

Proof: We need to show that each row of $\bar{G}\Delta$ is a codeword of C_d, and all rows in G are linearly independent. Let $\boldsymbol{c} = (c_0, c_1, \ldots, c_{n-1})$ be any row in $\bar{G}$. Then the polynomial representation of $\boldsymbol{c}\Delta$ is

$$\sum_{i=0}^{n-1} c_i (a^i)^\alpha x^i = \sum_{i=0}^{n-1} c_i (a^\alpha x)^i \ . \tag{5}$$

Since $\boldsymbol{c} \in C_\alpha$, $\boldsymbol{c}$ has roots $a^1, a^2, \ldots, a^{n-\alpha}$. Then it can be seen that (5) has roots $a^{-\alpha+1}$, $a^{-\alpha+2}$,..., $a^{n-2\alpha}$ that clearly contain $a^1, a^2, \ldots, a^{n-2\alpha}$. Hence, $\boldsymbol{c}\Delta \in C_d$.

In order to show that all rows in G are linearly independent, it is sufficient to show that $\boldsymbol{c}\Delta \notin C_\alpha$ for all nonzero $\boldsymbol{c} \in C_\alpha$. Assume that $\boldsymbol{c}\Delta \in C_\alpha$. Then $\sum_{i=0}^{n-1} c_i (a^\alpha x)^i$ must have roots $a^1, a^2, \ldots, a^{n-\alpha}$. It follows that $c(x)$ must have $a^{\alpha+1}, a^{\alpha+2}, \ldots, a^n$ as roots. Recall that $c(x)$ also has roots $a^1, a^2, \ldots, a^{n-\alpha}$. Since $n-1 \geq d = 2\alpha$, we have $n - \alpha \geq \alpha + 1$. Hence, $c(x)$ has n distinct roots of $a^1, a^2, \ldots, a^n$. This is impossible since the degree of $c(x)$ is at most $n-1$. Thus, $\boldsymbol{c}\Delta \notin C_\alpha$. ■

One advantage of the proposed scheme is that it can now operate on a smaller finite field than that of the scheme in [11]. Another advantage is that one can choose $\bar{G}$ (and Δ accordingly) freely as long as it is the generation matrix of an $[n,\alpha]$ RS code. Next, we present a least-update-complexity generator matrix that satisfies (4).

Corollary 1: Let Δ be the one given in Theorem 1. Let $\bar{G}$ be the generator matrix of a systematic $[n,\alpha]$ RS code, namely,

$$\bar{G} = [D|I]$$

where

$$D = \begin{bmatrix} b_{00} & b_{01} & b_{02} & \cdots & b_{0(n-\alpha-1)} \\ b_{10} & b_{11} & b_{12} & \cdots & b_{1(n-\alpha-1)} \\ b_{20} & b_{21} & b_{22} & \cdots & b_{2(n-\alpha-1)} \\ \vdots & & \vdots & & \vdots \\ b_{(\alpha-1)0} & b_{(\alpha-1)1} & b_{(\alpha-1)2} & \cdots & b_{(\alpha-1)(n-\alpha-1)} \end{bmatrix} \tag{6}$$

I is the $(\alpha \times \alpha)$ identity matrix, and

$$x^{n-\alpha+i} = u_i(x)g(x) + b_i(x) \text{ for } 0 \leq i \leq \alpha - 1 \ .$$

Then, $G = \begin{bmatrix} \bar{G} \\ \bar{G}\Delta \end{bmatrix}$ is a least-update-complexity generator matrix.

Proof: The result holds since each row of $\bar{G}$ is a nonzero codeword with the minimum Hamming weight $n - \alpha + 1$. ■

IV. Efficient Decoding Scheme for error-correcting MSR Codes

Unlike the decoding scheme in [11] that uses $[n,d]$ RS code, we propose to use the subcode of the $[n,d]$ RS code, the $[n, \alpha = k-1]$ RS code generated by $\bar{G}$, to perform the data reconstruction. The advantage of using the $[n, k-1]$ RS code is two-fold. First, its error correction capability is higher (namely, it can tolerate $\lfloor \frac{n-k+1}{2} \rfloor$ instead of $\lfloor \frac{n-d}{2} \rfloor$ errors). Second, it only requires the access of two additional storage nodes (as opposed to $d - k + 2 = k$ nodes) for the first error to correct.

Without loss of generality, we assume that the data collector retrieves encoded symbols from $k + 2v$ ($v \geq 0$) storage nodes, $j_0, j_1, \ldots, j_{k+2v-1}$. We also assume that there are v storage nodes whose received symbols are erroneous. The stored information of the $k+2v$ storage nodes are collected as the $k+2v$ columns in $Y_{\alpha \times (k+2v)}$. The $k+2v$ columns of G corresponding to storage nodes $j_0, j_1, \ldots, j_{k+2v-1}$ are denoted as the columns of G_{k+2v}. First, we discuss data reconstruction when $v = 0$. The decoding procedure is similar to that in [10].

No Error: In this case, $v = 0$ and there is no error in $Y_{\alpha \times k}$. Then,

$$Y_{\alpha \times k} \ = \ [Z_1 \bar{G}_k + Z_2 \bar{G}_k \Delta] \ . \tag{7}$$

Multiplying $\bar{G}_k^T$ and $Y_{\alpha \times k}$ in (7), we have [10],

$$\bar{G}_k^T Y_{\alpha \times k} \ = \ P + Q\Delta \ . \tag{8}$$

Since Z_1 and Z_2 are symmetric, P and Q are symmetric as well. The (i,j)th element of $P + Q\Delta$, $1 \leq i, j \leq k$ and $i \neq j$, is

$$p_{ij} + q_{ij} a^{(j-1)\alpha} \ , \tag{9}$$

and the (j,i)th element is given by

$$p_{ji} + q_{ji} a^{(i-1)\alpha} \ . \tag{10}$$

Since $a^{(j-1)\alpha} \neq a^{(i-1)\alpha}$ for all $i \neq j$, $p_{ij} = p_{ji}$, and $q_{ij} = q_{ji}$, combining (9) and (10), the values of p_{ij} and q_{ij} can be obtained. Note that we only obtain $k-1$ values for each row of P and Q since no elements in the diagonal of P or Q are obtained.

To decode P, recall that $P = \bar{G}_k^T Z_1 \bar{G}_k$. P can be treated as a portion of the codeword vector, $\bar{G}_k^T Z_1 \bar{G}$. By the construction of $\bar{G}$, it can be seen that $\bar{G}$ is a generator matrix of the $[n, k-1]$ RS code. Hence, each row in the matrix $\bar{G}_k^T Z_1 \bar{G}$ is a codeword. Since we know $k-1$ components in each row of P, it is possible to decode $\bar{G}_k^T Z_1 \bar{G}$ by the error-and-erasure decoder of the $[n, k-1]$ RS code.[3]

Since one cannot locate any erroneous position from the decoded rows of P, the decoded α codewords are accepted as $\bar{G}_k^T Z_1 \bar{G}$. By collecting the last α columns of $\bar{G}$ as $\bar{G}_\alpha$ to find its inverse (here it is an identity matrix), one can recover $\bar{G}_k^T Z_1$ from $\bar{G}_k^T Z_1 \bar{G}_k$. Note that $\alpha = k - 1$. Since any α rows in $\bar{G}_k^T$ are independent and thus invertible, we can pick any α of them to recover Z_1. Z_2 can be obtained similarly by Q.

[3] The error-and-erasure decoder of an $[n, k-1]$ RS code can successfully decode a received vector if $s + 2v < n - k + 2$, where s is the erasure (no symbol) positions, v is the number of errors in the received portion of the received vector, and $n - k + 2$ is the minimum Hamming distance of the $[n, k-1]$ RS code. If $s + 2v \geq n - k + 2$, the decoder may fail. In this case, the decoding result is taken as the received vector.

It is not trivial to extend the above decoding procedure to handle errors. The difficulty arises from the fact that any error in $Y_{\alpha\times n}$ will propagate to many elements in P and Q, due to the operations in (8), (9), and (10), such that many rows of P and Q cannot be decoded successfully or correctly (Please refer to Lemma 1). In the following, we present how to locate erroneous columns in Y using an RS decoder.

Multiple Errors: Before presenting the proposed decoding algorithm, we first prove that a decoding procedure can always successfully decode Z_1 and Z_2 if $v \le \lfloor\frac{n-k+1}{2}\rfloor$ and all storage nodes are accessed. Due to space limitation, all proofs are omitted except for the main theorem in this section. The detailed proofs can be found in [18].

Assume the storage nodes with errors correspond to the ℓ_0th, ℓ_1th, ..., ℓ_{v-1}th columns in the received matrix $Y_{\alpha\times n}$. Then,

$$\bar{G}^T Y_{\alpha\times n} \;=\; [\bar{G}^T Z_1 \bar{G} + \bar{G}^T Z_2 \bar{G}\Delta] + \bar{G}^T E \;, \quad (11)$$

where

$$E = \left[\mathbf{0}_{\alpha\times(\ell_0-1)} | \boldsymbol{e}_{\ell_0}^T | \mathbf{0}_{\alpha\times(\ell_1-\ell_0-1)} | \cdots | \boldsymbol{e}_{\ell_{v-1}}^T | \mathbf{0}_{\alpha\times(n-\ell_{v-1})}\right] \;.$$

Lemma 1: There are at least $n-k+2$ errors in each of the ℓ_0th, ℓ_1th, ..., ℓ_{v-1}th columns of $\bar{G}^T Y_{\alpha\times n}$.

When the $n-k+2$ errors in each erroneous column propagate to their respective rows through operations given in (9) and (10), these rows might not be decoded correctly since the error-correction capability of the RS code is $\lfloor(n-k+2)/2\rfloor$. We next have the main theorem to perform data reconstruction even when many rows of P and Q cannot be decoded correctly.

Theorem 2: Let $\bar{G}^T Y_{\alpha\times n} = \tilde{P} + \tilde{Q}\Delta$. Furthermore, let $\hat{P}$ be the corresponding portion of decoded codeword vector to $\tilde{P}$ and $E_P = \hat{P} \oplus \tilde{P}$ be the error pattern vector. Assume that the data collector accesses all storage nodes and there are v, $1 \le v \le \lfloor\frac{n-k+1}{2}\rfloor$, of them with errors. Then, there are at least $n-k+2-v$ nonzero elements in ℓ_jth column of E_P, $0 \le j \le v-1$, and at most v nonzero elements in the rest of columns of E_P.

Proof: Let us focus on the ℓ_jth column of E_P. By Lemma 1, there are at least $n-k+2$ errors in the ℓ_jth column of $\bar{G}^T Y_{\alpha\times n}$. $\tilde{P}$ is constructed from $\bar{G}^T Y_{\alpha\times n}$ based on (9) and (10). If there is only one value of (9) and (10) that is erroneous, then the constructed p_{ij} and q_{ij} will be wrong. However, when both values are erroneous, p_{ij} and q_{ij} might accidentally be correct. Among those $n-k+2$ erroneous positions, there are at least $n-k+2-v$ positions in error after constructing $\tilde{P}$ since at most v errors can be corrected in the process. It can be seen that at least $n-k+2-v$ positions are in error that are not among any of the ℓ_0th, ℓ_1th, ..., ℓ_{v-1}th elements in the ℓ_jth column. These errors are in the rows that can be decoded correctly. Hence, there are at least $n-k+2-v$ errors that can be located in ℓ_jth column of $\tilde{P}$ such that there are at least $n-k+2-v$ nonzero elements in the ℓ_jth column of E_P. There are at most v rows in $\tilde{P}$ that cannot be decoded correctly since each contains more than v errors. Hence, other than those columns with errors in the original matrix $\bar{G}^T Y_{\alpha\times n}$, at most v errors will be found in each of the remaining columns of $\tilde{P}$. ■

The above theorem allows us to design a decoding algorithm that can correct up to $\lfloor\frac{n-k+1}{2}\rfloor$ errors.[4] In particular, we need to examine the erroneous positions in $\bar{G}^T E$. Since $1 \le v \le \lfloor\frac{n-k+1}{2}\rfloor$, we have $n-k+2-v \ge \lfloor\frac{n-k+1}{2}\rfloor + 1 > v$. Thus, the way to locate all erroneous columns in $\tilde{P}$ is to find out all columns in E_P where the number of nonzero elements in them are greater than or equal to $\lfloor\frac{n-k+1}{2}\rfloor + 1$. After we locate all erroneous columns, we can follow a procedure similar to that given in the no error case to recover Z_1 from $\hat{P}$.

The above decoding procedure guarantees the recovery of Z_1 when all n storage nodes are accessed. However, it is not very efficient in terms of bandwidth usage. Next, we present a progressive decoding version of the proposed algorithm that only accesses enough extra nodes when necessary. Before presenting it, we need the following corollary.

Corollary 2: Consider that one accesses $k+2v$ storage nodes, among which v nodes are erroneous and $1 \le v \le \lfloor\frac{n-k+1}{2}\rfloor$. There are at least $v+2$ nonzero elements in ℓ_Jth column of E_P, $0 \le j \le v-1$, and at most v among the remaining columns of E_P.

Based on Corollary 2, we can design a progressive decoding algorithm [19] that retrieves extra data from remaining storage nodes when necessary. To handle Byzantine fault tolerance, it is necessary to perform integrity check after the original data is reconstructed. Two verification mechanisms have been suggested in [11]: cyclic redundancy check (CRC) and cryptographic hash function. Both mechanisms introduce redundancy to the original data before they are encoded and are suitable to be used in combination with the decoding algorithm.

The progressive decoding algorithm starts by accessing k storage nodes. Error-and-erasure decoding succeeds only when there is no error. If the integrity check passes, then the data collector recovers the original data. If the decoding procedure fails or the integrity check fails, then the data collector retrieves two more blocks of data from the remaining storage nodes. Since the data collector has $k+2$ blocks of data, the error-and-erasure decoding can correctly recover the original data if there is only one erroneous storage node among the $k+1$ nodes accessed. If the integrity check passes, then the data collector recovers the original data. If the decoding procedure fails or the integrity check fails, then the data collector retrieves two more blocks of data from the remaining storage nodes. The data collector repeats the same procedure until it recovers the original data or runs out of the storage nodes. The detailed decoding procedure is summarized in Algorithm 1. Another alternative decoding procedure, after locating erroneous columns in P, is that the decoder can collect k columns of $Y_{\alpha\times(k+2v)}$ with no errors to recover Z_1 and Z_2 by using the decoding procedure with no errors.

V. Conclusion

In this work, we proposed a new encoding scheme for the $[n, 2\alpha]$ error-correcting MSR codes from the generator matrix

[4] In constructing $\tilde{P}$ we only get $n-1$ values (excluding the diagonal). Since the minimum Hamming distance of an $[n, k-1]$ RS code is $n-k+2$, the error-and-erasure decoding can only correct up to $\lfloor\frac{n-1-k+2}{2}\rfloor$ errors.

Algorithm 1: Decoding of MSR Codes Based on $(n, k-1)$ RS Code for Data Reconstruction

```
begin
    $v = 0$; $j = k$;
    The data collector randomly chooses $k$ storage nodes
    and retrieves encoded data, $Y_{\alpha\times j}$;
    while $v \leq \lfloor \frac{n-k+1}{2} \rfloor$ do
        Collect the $j$ columns of $\bar{G}$ corresponding to
        accessed storage nodes as $\bar{G}_j$;
        Calculate $\bar{G}_j^T Y_{\alpha\times j}$;
        Construct $\tilde{P}$ and $\tilde{Q}$ by using (9) and (10);
        Perform progressive error-and-erasure decoding
        on each row in $\tilde{P}$ to obtain $\hat{P}$;
        Locate erroneous columns in $\hat{P}$ by searching for
        columns of them with at least $v + 2$ errors;
        assume that $\ell_e$ columns found in the previous
        action;
        Locate columns in $\hat{P}$ with at most $v$ errors;
        assume that $\ell_c$ columns found in the previous
        action;
        if ($\ell_e = v$ and $\ell_c = k + v$) then
            Copy the $\ell_e$ erronous columns of $\hat{P}$ to their
            corresponding rows to make $\hat{P}$ a symmetric
            matrix;
            Collect any $\alpha$ columns in the above $\ell_c$
            columns of $\hat{P}$ as $\hat{P}_\alpha$ and find its
            corresponding $\bar{G}_\alpha$;
            Multiply the inverse of $\bar{G}_\alpha$ to $\hat{P}_\alpha$ to recover
            $\bar{G}_j^T Z_1$;
            Recover $Z_1$ by the inverse of any $\alpha$ rows of
            $\bar{G}_j^T$;
            Recover $Z_2$ from $\tilde{Q}$ by the same procedure;
            Recover $\tilde{\boldsymbol{m}}$ from $Z_1$ and $Z_2$;
            if integrity-check($\tilde{\boldsymbol{m}}$) = SUCCESS then
                return $\tilde{\boldsymbol{m}}$;
        $j \leftarrow j + 2$;
        Retrieve 2 more encoded data from remaining
        storage nodes and merge them into $Y_{\alpha\times j}$;
        $v \leftarrow v + 1$;
    return FAIL;
```

of any $[n, \alpha]$ RS codes. It generalizes the previously proposed MSR codes in [11] and has several salient advantages. It allows the construction of least-update-complexity codes with a properly chosen systematic generator matrix. More importantly, the decoding scheme leads to an efficient decoding scheme that can tolerate more errors at the storage nodes, and accesses additional storage nodes only when necessary. A progressive decoding scheme was thereby devised with low communication overhead.

Acknowledgment

This work was supported in part by CASE: The Center for Advanced Systems and Engineering, a NYSTAR center for advanced technology at Syracuse University; the NSC of Taiwan under grant no. NSC 101-2221-E-011-069-MY3; US NSF under grant no. CNS-1117560, NSERC RGPIN/435601-2013, and McMaster University new faculty startup fund.

References

[1] A. G. Dimakis, P. B. Godfrey, M. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," in *Proc. of 26th IEEE International Conference on Computer Communications (INFOCOM)*, Anchorage, Alaska, May 2007, pp. 2000–2008.

[2] A. G. Dimakis, P. B. Godfrey, Y. Wu, M. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," *IEEE Trans. Inform. Theory*, vol. 56, pp. 4539 – 4551, September 2010.

[3] Y. Wu, A. G. Dimakis, and K. Ramchandran, "Deterministic regenerating codes for distributed storage," in *Proc. of 45th Annual Allerton Conference on Control, Computing, and Communication*, Urbana-Champaign, Illinois, September 2007.

[4] Y. Wu, "Existence and construction of capacity-achieving network codes for distributed storage," *IEEE Journal on Selected Areas in Communications*, vol. 28, pp. 277 – 288, February 2010.

[5] D. F. Cullina, "Searching for minimum storage regenerating codes," California Institute of Technology Senior Thesis, 2009.

[6] Y. Wu and A. G. Dimakis, "Reducing repair traffic for erasure coding-based storage via interference alignment," in *Proc. IEEE International Symposium on Information Theory*, Seoul, Korea, July 2009, pp. 2276–2280.

[7] K. V. Rashmi, N. B. Shah, P. V. Kumar, and K. Ramchandran, "Explicit construction of optimal exact regenerating codes for distributed storage," in *Proc. of 47th Annual Allerton Conference on Control, Computing, and Communication*, Urbana-Champaign, Illinois, September 2009, pp. 1243–1249.

[8] S. Pawar, S. E. Rouayheb, and K. Ramchandran, "Securing dynamic distributed storage systems against eavesdropping and adversarial attacks," arXiv:1009.2556v2 [cs.IT] 27 Apr 2011.

[9] F. Oggier and A. Datta, "Byzantine fault tolerance of regenerating codes," arXiv:1106.2275v1 [cs.DC] 12 Jun 2011.

[10] 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. Inform. Theory*, vol. 57, pp. 5227–5239, August 2011.

[11] Y. S. Han, R. Zheng, and W. H. Mow, "Exact regenerating codes for byzantine fault tolerance in distributed storage," in *Proc. of the IEEE INFOCOM 2012*, Orlendo, FL, March 2012.

[12] K. Rashmi, N. Shah, K. Ramchandran, and P. Kumar, "Regenerating codes for errors and erasures in distributed storage," in *Proc. of the 2012 IEEE International Symposium on Information Theory*, Cambridge, MA, July 2012.

[13] A. S. Rawat, S. Vishwanath, A. Bhowmick, and E. Soljanin, "Update efficient codes for distributed storage," in *Proc. of the 2011 IEEE International Symposium on Information Theory*, Saint Petersburg, Russia, July 2011.

[14] I. Tamo, Z. Wang, and J. Bruck, "Zigzag codes: MDS array codes with optimal rebuilding," arXiv:1112.0371 [cs.IT] 2 December 2011.

[15] ——, "MDS array codes with optimal rebuilding," arXiv:1103.3737 [cs.IT] 19 March 2011.

[16] D. S. Papailiopoulos, A. G. Dimakis, and V. R. Cadambe, "Repair optimal erasure codes through Hadamard designs," arXiv:1106.1634 [cs.IT] 8 June 2011.

[17] V. R. Cadambe, C. Huang, S. A. Jafar, and J. Li, "Optimal repair of MDS codes in distributed storage via subspace interference alignment," arXiv:1106.1250 [cs.IT] 7 June 2011.

[18] Y. S. Han, H.-T. Pai, R. Zheng, and P. K. Varshney, "Update-efficient error-correcting regenerating codes," arXiv:1301.4620 [cs.IT] 20 January 2013.

[19] Y. S. Han, S. Omiwade, and R. Zheng, "Progressive data retrieval for distributed networked storage," *IEEE Trans. on Parallel and Distributed Systems*, vol. 23, pp. 2303–2314, December 2012.