SURVIVABLE DISTRIBUTED STORAGE WITH PROGRESSIVE DECODING*

Yunghsiang S. Han†, Soji Omiwade, and Rong Zheng

Department of Computer Science
University of Houston
Houston, TX, 77204, USA
http://www.cs.uh.edu

Technical Report Number UH-CS-09-09

September 9, 2009

Keywords: Network storage, Byzantine failures, Reed-Solomon code, Error-detection code

Abstract

To harness the ever growing capacity and decreasing cost of storage, it is important to provide an abstraction of survivable storage in presence of Byzantine failures due to the prevalence of computer virus and software bugs. In this paper, we propose a *storage-optimal and computation efficient primitive* to spread information from a single data source to a set of storage nodes, which allows recovery from both crash-stop and Byzantine failures. In presence of crash-stop and Byzantine failures, a progressive data retrieval scheme is employed, which retrieves just enough data from live storage nodes. It adapts the cost of successful data retrieval to the degree of errors in the system. The cost of communication in data retrieval is derived analytically and corroborated by Monte-Carlo simulation results. Implementation and evaluation studies demonstrate speed-up of the progressive data retrieval scheme, which is comparable to that in a genie-aid decoding process.

*Partial support for this work was provided by the Computer Systems Research program of the National Science Foundation (NSF) under Award No. CNS-0834750 and CNS-0546391. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

† Graduate Institute of Communication Engineering, National Taipei University

SURVIVABLE DISTRIBUTED STORAGE WITH PROGRESSIVE DECODING*

Yunghsiang S. Han†, Soji Omiwade, and Rong Zheng

Abstract

To harness the ever growing capacity and decreasing cost of storage, it is important to provide an abstraction of survivable storage in presence of Byzantine failures due to the prevalence of computer virus and software bugs. In this paper, we propose a *storage-optimal and computation efficient primitive* to spread information from a single data source to a set of storage nodes, which allows recovery from both crash-stop and Byzantine failures. In presence of crash-stop and Byzantine failures, a progressive data retrieval scheme is employed, which retrieves just enough data from live storage nodes. It adapts the cost of successful data retrieval to the degree of errors in the system. The cost of communication in data retrieval is derived analytically and corroborated by Monte-Carlo simulation results. Implementation and evaluation studies demonstrate speed-up of the progressive data retrieval scheme, which is comparable to that in a genie-aid decoding process.

Index Terms

Network storage, Byzantine failures, Reed-Solomon code, Error-detection code

I. INTRODUCTION

Cost of storage has decreased drastically over the years. Many companies such as Google, Yahoo, Amazon offer GB, TB online storage for free or at very low cost. Meanwhile, low-power storage media are widely used in embedded devices or mobile computers. However, to harness the ever growing capacity and decreasing cost of distributed storage, a number of challenges need to be addressed, (i) volatility of storage availability due to network (dis)connectivity, varying administrative restriction or user preferences, and nodal mobility (of mobile devices); (ii) (partial) failures of storage devices. For example, flash media are known to be engineered to trade-off error probabilities for cost reduction; (iii) software bugs or malicious attacks, where an adversary manages to compromise a node and causes it to misbehave.

To ensure availability despite failure or compromise of storage nodes, survivable storage systems spread data redundantly across a set of distributed storage nodes. At the core of a survivable storage system is a coding scheme that maps information bits to stored bits, and vice versa. Without loss of generality, we call the unit of such mapping, symbols. A (k, n) coding is defined by the following two primitives:

- **encode** $\mathbf{c} = (\mathbf{u}, k, n)$, which returns a coded vector $\mathbf{c} = [c_0, c_1, \ldots, c_{n-1}]$ of length n from k information symbols $\mathbf{u} = [u_0, u_1, \ldots, u_{k-1}]$. The coded symbols can be stored on separate storage nodes.
- **decode** $\mathbf{u} = (\mathbf{r}, k, n)$, which accesses a subset of storage nodes, and returns the information symbols from possibly corrupted symbols.

Many existing approaches to survivable storage assume crash-stop behaviors, i.e., a storage device becomes unavailable if failed (also called "erasure"). Solutions such as various RAID configurations [1] and their extensions are engineered for high read and write data throughput. Thus, typically low-complexity (replication or XOR-based) coding mechanisms are employed to recover from limited degree of erasure. We argue that Byzantine failures, where devices fail in arbitrary manner and cannot be trusted, are becoming more pertinent with the prevalence of cheap storage devices, software bugs and malicious attacks. Efficient encode and decode primitives that can detect data corruption and handle Byzantine failures serve as a fundamental building block to support higher level abstractions such as multi-reader multi-writer atomic register [2] and digital fingerprints [3] in distributed systems.

*Partial support for this work was provided by the Computer Systems Research program of the National Science Foundation (NSF) under Award No. CNS-0834750 and CNS-0546391. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

† Graduate Institute of Communication Engineering, National Taipei University

For fixed error correction capability, the efficiency of encode and decode primitives can be evaluated by three metrics, i) *storage overhead* measured as the ratio between the number of storage symbols and total information symbols (n/k); ii) *encoding and decoding computation time*; and iii) *communication overhead* measured in the number of bits transferred in the network for encode and decode. Communication overhead is of much importance in wide-area and/or low-bandwidth storage systems. In this paper, we propose a novel solution to spreading redundant information efficiently across distributed storage nodes using incremental Reed-Solomon (RS) decoding. By virtue of RS codes, our scheme is storage optimal. The key novelty of the proposed approach lies in a progressive data retrieval procedure, which retrieves just enough data from live storage nodes, and performs decoding incrementally. As a result, both communication and computation cost are minimized, and are made adaptive to the degree of actual errors in the system. We provide a theoretical characterization of the communication cost and success rate of data retrieval using the proposed scheme in presence of arbitrary errors in the system. Our implementation studies demonstrate up to 20 times speed-up of the progressive data retrieval scheme in computation time, relative to a classical scheme. Moreover, the proposed scheme is comparable to that of a genie-aid decoding process, which assumes knowledge of failure modes of storage nodes.

Main Contributions: In this paper, we make the following contributions:

- Design of a novel progressive data retrieval mechanism that is storage and communication optimal, and computationally efficient. It handles Byzantine failures in storage nodes gracefully as the probability of failures increases.
- Development of an analytical model to evaluate the communication cost of data retrieval.

The rest of the paper is organized as follows. Related work is given in Section II. The progressive data retrieval scheme is presented in Section III, with the details of the incremental RS decoding algorithm in Section IV. An analysis of our coding, communication and success rate complexity is provided in Section V. Evaluation results are presented in Section VI and then a discussion of the application of the proposed progressive data retrieval scheme in sensor network storage and peer-to-peer tuple space follows in Section VII. Finally, we conclude the paper in Section VIII.

II. Background and Related Work

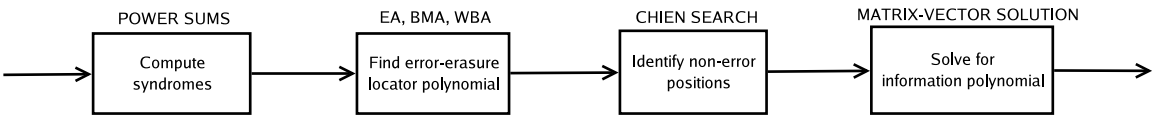


Fig. 1. Block diagram of RS decoding. The texts on top of the boxes correspond to existing algorithms.

In storage systems, ensuring reliability requires the introduction of redundancy. A file is divided into k pieces, encoded into n coded pieces and stored at n nodes. One important metric of coding efficiency is the redundancy-reliability tradeoff defined as n/k. The simplest form of redundancy is replication. As a generalization of replication, erasure coding offers better storage efficiency. The Maximum Distance Separable (MDS) codes are optimal as it provides largest separation among code words, and an (n, k) MDS code will be able to recover from any v errors if $v \leq \lfloor \frac{n-k-s}{2} \rfloor$, where s is the number of erasures (or unretrievable symbols).

A. Reed-Solomon codes

RS codes are the most well-known class of MDS codes. RS operates on symbols of m bits. An (n, k) RS code is a linear code, with each symbol in $GF(2^m)$, and parameters $n = 2^m - 1$ and $n - k = 2t$, where n is the total number of symbols in a codeword, k is the total number of information symbols, and t is the symbol-error-correcting capability of the code.

Encoding: Let the sequence of k information symbols in $GF(2^m)$ be $\boldsymbol{u} = (u_0, u_1, \ldots, u_{k-1})$ and $u(x)$ be the information polynomial of $\boldsymbol{u}$ represented as

$$u(x) = u_0 + u_1 x + \cdots + u_{k-1} x^{k-1} \ .$$

The codeword polynomial, $c(x)$, corresponding to $u(x)$ can be encoded as

$$c(x) = u(x)g(x) \ ,$$

where $g(x)$ is a generator polynomial of the RS code. It is well-known that $g(x)$ can be obtained as

$$\begin{aligned} g(x) &= (x-\alpha^b)(x-\alpha^{b+1})\cdots(x-\alpha^{b+2t-1}) \\ &= g_0 + g_1 x + g_2 x^2 + \cdots + g_{2t} x^{2t} \ , \end{aligned} \tag{1}$$

where α is a primitive element in $GF(2^m)$, b an arbitrary integer, and $g_i \in GF(2^m)$.

Decoding: The decoding process of RS codes is more complex. Complete description of decoding of RS codes can be found in [4].

Let $r(x)$ be the received polynomial and $r(x) = c(x) + e(x) + \gamma(x) = c(x) + \lambda(x)$, where $e(x) = \sum_{j=0}^{n-1} e_j x^j$ is the error polynomial, $\gamma(x) = \sum_{j=0}^{n-1} \gamma_j x^j$ the erasure polynomial, and $\lambda(x) = \sum_{j=0}^{n-1} \lambda_j x^j = e(x) + \gamma(x)$ the errata polynomial. Note that $g(x)$ and (hence) $c(x)$ have $\alpha^b, \alpha^{b+1}, \ldots, \alpha^{b+2t-1}$ as roots. This property is used to determine the error locations and recover the information symbols.

The basic procedure of RS decoding is shown in Figure 1. The last step of the decoding procedure involves solving a linear set of equations, and can be made efficient by the use of Vandermonde generator matrices [5].

In $GF(2^m)$, addition is equivalent to bitwise exclusive-or (XOR), and multiplication is typically implemented with multiplication tables or discrete logarithm tables. To reduce the complexity of multiplication, Cauchy Reed-Solomon (CRS) codes [6] have been proposed to use a different construction of the generator matrix, and convert multiplications to XOR operations for erasure. However, CRS codes incur the same complexity as RS codes for error corrections.

B. XOR-based erasure codes for storage

Several XOR-based erasure codes (in a field of GF(2)) [7]–[10] have been used in storage systems. In RAID-6 systems, each disk is partitioned into strips of fixed size. Two parity strips are computed using one strip from each data disk, forming a stripe together with the data strips. EVEN-ODD [9], Row Diagonal Parity (RDP) [7], and Minimal Density RAID-6 codes [8] use XOR operations, and are specific to RAID-6. A detailed comparison of the encoding and decoding performance of several open-source erasure coding libraries for storage is provided [11].

The gain in computation efficiency of XOR-based erasure codes is achieved by trading off fault tolerance. RAID-6 systems can recover from the loss of exactly two disks but cannot handle Byzantine failures.

III. Progressive Data Retrieval in Presence of Byzantine Failures

We use the abstractions of a data node which is a source of information that must be stored, and a storage node which corresponds to a storage device. Nodes are subject to both crash-stop failures, where data cannot be accessed and Byzantine failures, where arbitrary data may be returned. The communication cost of transferring one unit of data from the data source to a storage node is assumed to be constant independent of the location of the storage node.

A. Data storage

The data storage scheme consists of two steps. First, for data integrity, a message authentication code (MAC) is added to each data block generated by a data node before it is encoded. Many one-way hash functions such as MD5, SHA-1, SHA-2 can be used. For simplicity, we adopt CRC code for error detection with r redundant bits [4], [12]. It has been proven that the portion of errors that cannot be detected by a CRC code is dependent only on its number of redundant bits. That is, a CRC code with r redundant bits *cannot* detect $(\frac{1}{2^r})100\%$ portion of errors. If T_0 is the size of the original data, then the size of the resulting data is $T = T_0 + r$. The overhead can be amortized by combining multiple data blocks together.

In the second step, we partition a data block into information symbols of length m bits and apply RS codes. The data-generating node divides its data into $\lceil T/m \rceil$ pieces (symbols) such that each symbol represents an element in $GF(2^m)$. Next the $\lceil T/m \rceil$ symbols are divided into $\left\lceil \frac{\lceil T/m \rceil}{k} \right\rceil$ information groups each of k symbols. Let k symbols of the ith group be the components in information vector $\boldsymbol{u}_i = (u_{i0}, u_{i1}, \ldots, u_{i(k-1)})$, where $1 \leq i \leq \left\lceil \frac{\lceil T/m \rceil}{k} \right\rceil$. The node encodes $\boldsymbol{u}_i$ into $\boldsymbol{c}_i = (c_{i0}, c_{i1}, \ldots, c_{i(n-1)})$ with n symbols as

$$\boldsymbol{c}_i = \boldsymbol{u}_i \boldsymbol{G},$$

where

$$\boldsymbol{G} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ \alpha & \alpha^2 & \alpha^3 & \cdots & \alpha^n \\ \alpha^2 & (\alpha^2)^2 & (\alpha^3)^2 & \cdots & (\alpha^n)^2 \\ & & & \vdots & \\ \alpha^{k-1} & (\alpha^2)^{k-1} & (\alpha^3)^{k-1} & \cdots & (\alpha^n)^{k-1} \end{bmatrix}. \tag{2}$$

Recall that α is a primitive element (generator) of $GF(2^m)$ which can be determined in advance. The data-generating node then packs all $c_{i,j}$, $0 \leq i \leq \left\lceil \frac{\lceil T/m \rceil}{k} \right\rceil$, and sends them with their index j to $(j+1)$th storage node via the network.

B. Data retrieval

To reconstruct the source data, a collector needs to access sufficient number of storage nodes to ensure data integrity. Among n storage nodes, let the number of erasures, which includes the number of crash-stop nodes and the number of nodes that have not been accessed, be s. Identity of crash-stop nodes can be determined by the use of keep-alive messages. Additionally, there are v nodes with Byzantine failures. Neither v nor the identity of these nodes are known to the data collector.

It has been proven that $\boldsymbol{G}$ given in (2) is a generator matrix of a RS code [4] and thus an error-erasure decoding algorithm can recover all data if there is no error in at least k encoded symbols. Without loss of generality, we assume that the data collector retrieves encoded symbols from j_0th, j_1th,$\cdots$, and j_{k-1}th storage nodes. If no error is present, the k symbols in ith group of any data-generating node can be recovered by solving the following system of linear equations:

$$[u_{i0}, u_{i1}, \ldots, u_{i(k-1)}]\hat{\boldsymbol{G}} = [c_{ij_0}, c_{ij_1}, \ldots, c_{ij_{k-1}}] \ , \tag{3}$$

where

$$\hat{\boldsymbol{G}} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ \alpha^{j_0} & \alpha^{j_1} & \cdots & \alpha^{j_{k-1}} \\ (\alpha^{j_0})^2 & (\alpha^{j_1})^2 & \cdots & (\alpha^{j_{k-1}})^2 \\ & & \vdots & \\ (\alpha^{j_0})^{k-1} & (\alpha^{j_1})^{k-1} & \cdots & (\alpha^{j_{k-1}})^{k-1} \end{bmatrix}.$$

$\hat{\boldsymbol{G}}$ can be constructed by the primitive element and the index associated with c_{ij_d}, $0 \leq d \leq k-1$.

When the number of erroneous (or compromised) nodes is unknown but is bounded, the proposed progressive procedure for data retrieval minimizes communication cost without any *a priori* knowledge regarding failure models of nodes.

From Section II, we know that RS codes can recover from any v errors if $v \leq \lfloor \frac{n-k-s}{2} \rfloor$. Therefore, if the number of compromised nodes (v) is small, more erasures (s) can be tolerated, and less nodes need to be accessed (by treating them as unavailable). The data retrieval procedure proceeds in stages. At stage l, l errors are assumed. If RS decoding fails or the decoded information symbols fail the CRC check, there must exist more erroneous nodes than RS error-erasure decoding can handle at this stage. In order to correct *one* more error, *two* more symbols need to be collected, since the number of erasures allowed is reduced by two. Therefore, the total number of symbols retrieved at stage l is $k + 2l$.

This procedure is clearly optimal in communication costs as additional symbols are retrieved only when necessary. However, if applied naively, its computation cost can be quite high since RS decoding shall be performed at each

The last information group may have symbols less than k. In this case, zero symbols will be appended during the encoding procedure.

stage. For example, when $n = 1023$, $k = 401$, with 1% error probability (defined as probability that a storage node is faulty), our analytical results (Section V) show that on average 409.2 storage nodes need to be accessed. That is, the decoding needs to be done 10 times on average. On the other hand, consider an alternative (but naive) scheme that retrieves coded symbols from each of n storage nodes and decodes only once. The naive scheme may incur less computation cost but suffers from high communication cost. One may be tempted to think that such trade-offs between computation and communication are unavoidable. Instead, in Section IV, we devise an algorithm that can utilize intermediate computation results from previous stages and performs RS decoding incrementally. Combined with the incremental decoding of stored symbols, the proposed progressive data retrieval scheme (detailed in Algorithm 1) is both computation and communication efficient. For simplicity, Algorithm 1 is presented only for one group of encoded symbols. It is applied to all groups of encoded symbols to retrieve all data for the data-generating node.

Algorithm 1: Progressive Data Retrieval

```
begin
    i ← k;
    The data collector randomly chooses k storage nodes and retrieves encoded data, c_i = [c_{j_0}, c_{j_1}, ..., c_{j_{k-1}}];
    r_i = c_i
    repeat
        u = r_i Ĝ^{-1};
1       if CRCTest(u) = SUCCESS then
            Delete CRC checksum from u to obtain u_0;
            return u_0;
        else
            repeat
                i ← i + 2
                Two more encoded data from remaining nodes i_1, i_2, are retrieved
                c_i ← c_{i-2} ∪ {c_{i_1}, c_{i_2}}
2           until {(r_i = IncrRSDecode(c_i)) = SUCCESS ‖ i ≥ n − 1} ;
    until i ≥ n − 1 ;
    return FAIL;
end
```

In Algorithm 1, for each i (or accordingly stage $l = (i-k)/2$ where the number of errors $v > l$), the decoding process declares errors in one of two cases. In Line 2, the proposed incremental RS decoding algorithm ($IncrRSDecode()$) may fail to produce decoded symbols. Otherwise, in Line 1, the decoded symbols fail the CRC check. Our implementation (Section VI) shows that the former happens frequently. Thus, in most cases, CRC checking is carried out only once throughout the entire decoding process.

IV. Incremental RS Decoding

In this section, we present the incremental RS decoding algorithm. Compared to the classic RS decoding, it utilizes intermediate computation results and decodes incrementally as more symbols become available.

A. The basic algorithm

Given the received coded symbols $[r_0, r_1, \ldots, r_n]$ with erasures set to be zero, the generalized syndrome polynomial $S(x)$ can be calculated as [13],

$$S(x) = \sum_{j=0}^{n-1} r_j \alpha^{jb} \frac{T(x) - T(\alpha^j)}{x - \alpha^j} = \sum_{j=0}^{n-1} \lambda_j \alpha^{jb} \frac{T(x) - T(\alpha^j)}{x - \alpha^j} \ , \tag{4}$$

where $T(x)$ is an arbitrary polynomial with degree $(n-k)$. Assume that v errors occur in unknown locations $j_1, j_2, \ldots, j_v$ and s erasures in known locations $m_1, m_2, \ldots, m_s$ of the received polynomial. Then

$$e(x) = e_{j_1} x^{j_1} + e_{j_2} x^{j_2} + \cdots + e_{j_v} x^{j_v}$$

and

$$\gamma(x) = \gamma_{m_1} x^{m_1} + \gamma_{m_2} x^{m_2} + \cdots + \gamma_{m_s} x^{m_s} \ ,$$

where e_{j_ℓ} is the value of the ℓ-th error, $\ell = 1, \cdots, v$, and γ_{m_ℓ} is the value of the ℓ-th erasure, $\ell = 1, \cdots, s$. The decoding process is to find all j_ℓ, e_{j_ℓ}, m_ℓ, and γ_{m_ℓ}. Let $\mathbf{E} = \{j_1, \cdots, j_v\}$, $\mathbf{M} = \{m_1, \cdots, m_s\}$, and $\mathbf{D} = \mathbf{E} \cup \mathbf{M}$. Clearly, $\mathbf{E} \cap \mathbf{M} = \emptyset$. It has been shown that a key equation for decoding is

$$\Lambda(x)S(x) = \Psi(x)T(x) + \Omega(x) \ , \tag{5}$$

where

$$\begin{aligned}
\Lambda(x) &= \prod_{j \in \mathbf{D}} (x - \alpha^j) = \prod_{j \in \mathbf{E}} \left(x - \alpha^j\right) \prod_{j \in \mathbf{M}} \left(x - \alpha^j\right) \\
&= \Lambda_{\mathbf{E}}(x)\Lambda_{\mathbf{M}}(x) && (6) \\
\Psi(x) &= \sum_{j \in \mathbf{D}} \lambda_j \alpha^{jb} \prod_{\substack{i \in \mathbf{D} \\ i \neq j}} \left(x - \alpha^i\right) && (7) \\
\Omega(x) &= -\sum_{j \in \mathbf{D}} \lambda_j \alpha^{jb} T(\alpha^j) \prod_{\substack{i \in \mathbf{D} \\ i \neq j}} \left(x - \alpha^i\right) \ . && (8)
\end{aligned}$$

If $2v + s \leq n - k + 1$, then (5) has a unique solution $\{\Lambda(x), \Psi(x), \Omega(x)\}$. Instead of solving (5) by either the Euclidean or Berlekamp-Massey algorithm we introduce a reduced key equation [13] that can be solved by the Welch-Berlekamp (W-B) algorithm [4]. It will be demonstrated that by using W-B algorithm and the reduced key equation, the complexity of decoding can be reduced drastically. Let $\mathbf{T} = \{j | T(\alpha^j) = 0\}$. Let a set of coordinates $\mathbf{U} \subset \{0, 1, \ldots, n-1\}$ be defined by $\mathbf{U} = \mathbf{M} \cap \mathbf{T}$. A polynomial $\Lambda_{\mathbf{U}}(x)$ is then defined by $\Lambda_{\mathbf{U}}(x) = \prod_{j \in \mathbf{U}} \left(x - \alpha^j\right)$, which is known for the receiver since $T(x)$ and $\mathbf{M}$ are both known. Since $\Lambda_{\mathbf{U}}(x)$ divides both $\Lambda(x)$ and $T(x)$, according to (5), it also divides $\Omega(x)$. Hence, we have the following reduced key equation:

$$\tilde{\Lambda}(x)S(x) = \Psi(x)\tilde{T}(x) + \tilde{\Omega}(x) \ , \tag{9}$$

where

$$\begin{aligned}
\Lambda(x) &= \tilde{\Lambda}(x)\Lambda_{\mathbf{U}}(x) \\
T(x) &= \tilde{T}(x)\Lambda_{\mathbf{U}}(x) \\
\Omega(x) &= \tilde{\Omega}(x)\Lambda_{\mathbf{U}}(x) \ .
\end{aligned}$$

Note that $\tilde{\Lambda}(x)$ is still a multiple of the error location polynomial $\Lambda_{\mathbf{E}}(x)$. The reduced key equation can have a unique solution if

$$\deg(\tilde{\Omega}(x)) < \deg(\tilde{\Lambda}(x)) < \frac{n - k + 1 + s}{2} - |\mathbf{U}| \ , \tag{10}$$

where $\deg(\cdot)$ is the degree of a polynomial and $|\mathbf{U}|$ is the number of elements in set $\mathbf{U}$.

For all $j \in \mathbf{T} \backslash \mathbf{U}$, by (9), we have

$$\tilde{\Lambda}(\alpha^j)S(\alpha^j) = \tilde{\Omega}(\alpha^j) \tag{11}$$

since $\tilde{T}(\alpha^j) = 0$. Note that α^j is a sampling point and $S(\alpha^j)$ the sampled value for (11). The unique solution $\{\tilde{\Lambda}(x), \tilde{\Omega}(x)\}$ can then be found by the W-B algorithm with time complexity $O((n - k - |\mathbf{U}|)^2)$ [4]. Once all coefficients of the errata polynomial are found, the error locations j_ℓ can be determined by successive substitution through Chien search [14]. When the solution of (9) is obtained, the errata values can be calculated. Since there is no need to recover the errata values in our application we omit the calculations. In summary, there are three steps in the decoding of RS codes that must be implemented. First, the sampled values of $S(\alpha^j)$ for $j \in \mathbf{T} \backslash \mathbf{U}$ must be calculated. Second, the W-B algorithm is performed based on the pairs $\left(\alpha^j, S(\alpha^j)\right)$ in order to obtain a valid $\tilde{\Lambda}(x)$. If a valid $\tilde{\Lambda}(x)$ is obtained, then error locations are found by Chien search; otherwise, decoding failure is reported.

Since the received values in the erased positions are zero, $\gamma_{m_\ell} = -c_{m_\ell}$ for $\ell = 1, \cdots, s$.

$\backslash$ is the set difference.

B. Incremental computation of $S(x)$, $\tilde{\Lambda}(x)$, $\tilde{\Omega}(x)$

Let us choose

$$T(x) = (x - \alpha^{m_0})\,(x - \alpha^{m_1}) \cdots (x - \alpha^{m_{n-k-1}}) \ ,$$

where m_ℓ are those corresponding positions of missing data symbols after the data collector has retrieved encoded symbols from k storage nodes. In the decoding process, these are erased positions before the first iteration of error-erasure decoding. Let $\mathbf{U}_0 = \{m_0, \ldots, m_{n-k-1}\}$. It has been proven that the generator polynomial of the RS code encoded by (2) has $\alpha^{n-k}, \alpha^{n-k-1}, \ldots, \alpha$ as roots. The error-erasure decoding algorithm is mainly based on W-B algorithm which is an iterative rational interpolation method.

In the ℓth iteration, ℓ errors are assumed in the data and the number of erasures is $n-k-2\ell$. Let $(j_1^{(\ell)}+1)$th and $(j_2^{(\ell)}+1)$th nodes be the two storage nodes just accessed in the ℓth iteration. Let $\mathbf{U}_\ell = \mathbf{U}_{\ell-1} \backslash \{j_1^{(\ell)}, j_2^{(\ell)}\}$. Based on $\mathbf{U}_\ell$ the W-B algorithm will find $\tilde{\Lambda}^{(\ell)}(x)$ and $\tilde{\Omega}^{(\ell)}(x)$ which satisfy

$$\tilde{\Lambda}^{(\ell)}(\alpha^j) S^{(\ell)}(\alpha^j) = \tilde{\Omega}^{(\ell)}(\alpha^j) \text{ for all } j \in \mathbf{U}_0 \backslash \mathbf{U}_\ell \ ,$$

where $S^{(\ell)}(x)$ is the generalized syndrome with $r_i = 0$ for all $r_i \in \mathbf{U}_\ell$. It has been shown that $\deg(\tilde{\Lambda}^{(\ell)}(x)) > \deg(\tilde{\Omega}^{(\ell)}(x))$ for any ℓ by a property of W-B algorithm. Thus, if $\deg(\tilde{\Lambda}^{(\ell)}(x)) < \frac{n-k+1+|\mathbf{U}_\ell|}{2} - |\mathbf{U}_\ell| = \ell + 1/2$, then the unique solution will exist due to (10). By the definition of generalized syndrome polynomial in (4), for $i \in \mathbf{U}_0 \backslash \mathbf{U}_\ell$, we have

$$\begin{aligned} S^{(\ell)}(\alpha^i) &= \sum_{j=0}^{n-1} r_j \alpha^j \frac{T(\alpha^i) - T(\alpha^j)}{\alpha^i - \alpha^j} \\ &= \sum_{\substack{j=0 \\ j \notin \mathbf{U}_0}}^{n-1} r_j \alpha^j \frac{T(\alpha^j)}{\alpha^j - \alpha^i} + r_i \alpha^i T'(\alpha^i) \\ &= \sum_{\substack{j=0 \\ j \notin \mathbf{U}_0}}^{n-1} \frac{F_j}{\alpha^j - \alpha^i} + r_i \alpha^i T'(\alpha^i) \ , \end{aligned} \tag{12}$$

where $T'(x)$ is the derivative of $T(x)$ and $F_j = r_j \alpha^j T(\alpha^j)$. Note that $T'(\alpha^i) = \prod_{\substack{j \in \mathbf{U}_0 \\ m_j \neq i}} \left(\alpha^i - \alpha^{m_j}\right)$. It is easy to see that $S^{(\ell)}(\alpha^i)$ is not related to any r_j, where $j \in \mathbf{U}_0$ and $j \neq i$. Hence, $S^{(\ell-1)}(\alpha^i) = S^{(\ell)}(\alpha^i)$ for all $i \in \mathbf{U}_0 \backslash \mathbf{U}_{\ell-1}$. This fact implies that all sampled values in previous iterations can be directly used in current iteration of the W-B algorithm.

Define $\text{rank}[N(x), W(x)] = \max[2\deg(W(x)), 1 + 2\deg(N(x))]$. The incremental RS decoding algorithm is described in Algorithm 2. Upon success, the incremental RS decoding algorithm returns k non-error symbols. The procedure will report failure either as the result of mismatched degree of the error locator polynomial, or insufficient number of roots found by Chien search (Line 2). In both cases, no further erasure decoding is required. This reduces the decoding computation time.

V. Complexity Analysis

A. Encoding complexity

The communication cost incurred by the encoded data generated by a data-generating node is $nm \left\lceil \frac{\lceil T/m \rceil}{k} \right\rceil$. It is easy to see that the total bits stored in each storage node is $m \left\lceil \frac{\lceil T/m \rceil}{k} \right\rceil \approx \lceil T/k \rceil$.

Assuming a software implementation on field operations without look-up tables is used, the computation complexity of encoding can be estimated as follows. Given that computation of one multiplication in $GF(2^m)$ is of m^2 bit exclusive-ORs. At the data-generating node, $kn \left\lceil \frac{\lceil T/m \rceil}{k} \right\rceil$ multiplications are performed, which is equivalent to $kn \left\lceil \frac{\lceil T/m \rceil}{k} \right\rceil m^2$ bit exclusive-ORs.

Algorithm 2: Incremental RS Decoding $IncrRSDecode$

init : Calculate F_j given in (12) for all $j \notin \mathbf{U}_0$.
$\ell \leftarrow 0$; $\tilde{\Lambda}^{(0)}(x) \leftarrow 1$;
$\tilde{\Omega}^{(0)}(x) \leftarrow 0$; $\Phi^{(0)}(x) \leftarrow 0, \Theta^{(0)}(x) \leftarrow 1$.
input : stage l, two new symbols at the $(j_1^{(\ell)}+1)$th, and $(j_2^{(\ell)}+1)$th nodes
output: FAIL or non-error symbols $\boldsymbol{r}$
begin
  foreach $i = 1, 2$ **do**
1 $x_i^{(\ell)} \leftarrow \alpha^{j_i^{(\ell)}}$ and $y_i^{(\ell)} \leftarrow S^{(\ell)}(x_i^{(\ell)})$
  end
  for $i = 1$ **to** 2 **do**
    $b_i^{(\ell-1)} \leftarrow \tilde{\Omega}^{(\ell-1)}(x_i^{(\ell)}) - y_i^{(\ell)}\tilde{\Lambda}^{(\ell-1)}(x_i^{(\ell)})$;
    if $b_i^{(\ell-1)} = 0$ **then**
      $\tilde{\Lambda}^T(x) \leftarrow \tilde{\Lambda}^{(\ell-1)}(x)$; $\tilde{\Omega}^T(x) \leftarrow \tilde{\Omega}^{(\ell-1)}(x)$; $\Theta^T(x) \leftarrow (x - x_i^{(\ell)})\Theta^{(\ell-1)}(x)$; $\Phi^T(x) \leftarrow (x - x_i^{(\ell)})\Phi^{(\ell-1)}(x)$
    else
      $a_i^{(\ell-1)} \leftarrow \Theta^{(\ell-1)}(x_i^{(\ell)}) - y_i^{(\ell)}\Phi^{(\ell-1)}(x_i^{(\ell)})$; $\Theta^T(x) \leftarrow (x - x_i^{(\ell)})\tilde{\Omega}^{(\ell-1)}(x)$; $\Phi^T(x) \leftarrow (x - x_i^{(\ell)})\tilde{\Lambda}^{(\ell-1)}(x)$;
      $\tilde{\Omega}^T(x) \leftarrow b_i^{(\ell-1)}\Theta^{(\ell-1)}(x) - a_i^{(\ell-1)}\tilde{\Omega}^{(\ell-1)}(x)$; $\tilde{\Lambda}^T(x) \leftarrow b_i^{(\ell-1)}\Phi^{(\ell-1)}(x) - a_i^{(\ell-1)}\tilde{\Lambda}^{(\ell-1)}(x)$.
    end
    if *rank*$[\tilde{\Omega}^T(x), \tilde{\Lambda}^T(x)] >$ *rank*$[\Theta^T(x), \Phi^T(x)]$ **then**
      swap $[\tilde{\Omega}^T(x), \tilde{\Lambda}^T(x)] \leftrightarrow [\Theta^T(x), \Phi^T(x)]$.
    end
    if $i = 1$ **then**
      $\tilde{\Omega}^{(\ell-1)}(x) \leftarrow \tilde{\Omega}^T(x)$; $\tilde{\Lambda}^{(\ell-1)}(x) \leftarrow \tilde{\Omega}^T(x)$; $\Theta^{(\ell-1)}(x) \leftarrow \Theta^T(x)$, $\Phi^{(\ell-1)}(x) \leftarrow \Phi^T(x)$;
    else
      $\tilde{\Omega}^{(\ell)}(x) \leftarrow \tilde{\Omega}^T(x)$; $\tilde{\Lambda}^{(\ell)}(x) \leftarrow \tilde{\Omega}^T(x)$; $\Theta^{(\ell)}(x) \leftarrow \Theta^T(x)$; $\Phi^{(\ell)}(x) \leftarrow \Phi^T(x)$.
    end
  end
  if $\deg(\tilde{\Lambda}^{(\ell)}(x)) \neq \ell$ **then**
    return FAIL;
  end
  NumErrorLoc = ChienSearch($\tilde{\Lambda}^{(\ell)}(x)$).
2 **if** $NumErrorLoc > n - k \parallel NumErrorLoc \neq \deg(\tilde{\Lambda}^{(\ell)}(x))$ **then**
    return FAIL;
  end
  return k non-error symbols $\boldsymbol{r}$;
end

B. Computation complexity of decoding

In the subsequent complexity analysis, the worst case is assumed, namely, no failure on decoding is reported in Algorithm 2 (Line 2), and the algorithm runs to completion.

In CRC checking, one polynomial division is performed. Since the dividend is of degree $T-1$ and the divider is of degree r, the computation complexity of CRC checking is of $O(Tr)$.

Let v be the number of errors when the decoding procedure is completed. In the ℓth iteration, ℓ errors are assumed in the data and the number of erasures is $n-k-2\ell$. We first need to calculate two syndrome values. This can be obtained by the F_j calculated initially. For instance, in the first iteration, according to (12), the computation complexity is of $O(k(n-k))$ since there are k F_j's to be calculated and each is a product of $n-k$ terms. In the next iteration, two more symbols are added to (4). Hence, the updated syndrome values can be obtained by an extra $O(k) + O(n-k)$ computations. To find the error-locator polynomial, the W-B algorithm is performed two steps in each iteration with complexity $O(\ell)$. Since we only consider software implementation, the Chien search can be replaced by substituting a power of α into the error-locator polynomial. It needs to test for at most $k+\ell$ positions to locate k non-error positions such that it takes $O((k+\ell)\ell)$ computations. Finally, inversion of Vandermonde matrix $\hat{G}$ requires $O(k\log^2(k))$ time [15]. In summary, the computation in the ℓth iteration for $\ell > 1$ is

$$L_v(\ell) \quad = \quad O(k\log^2 k) + O(n-k) + O(k\ell + \ell^2) \ .$$

$$\begin{aligned}\bar{N}(n,k) &= \sum_{v=0}^{n-k}\binom{n}{v}p^v(1-p)^{n-v}\sum_{i=0}^{\min(v,\lfloor\frac{n-k}{2}\rfloor,n-v-k)}(k+2i)\frac{\binom{n-v}{i+k-1}\binom{v}{i}}{\binom{n}{2i+k-1}}\times\frac{k}{i+k}\times\frac{n-v-(i+k-1)}{n-(2i+k-1)}\\ &+ \sum_{v=0}^{n-k}n\binom{n}{v}p^v(1-p)^{n-v}\left(1-\sum_{i=0}^{\min(v,\lfloor\frac{n-k}{2}\rfloor,n-v-k)}\frac{\binom{n-v}{i+k-1}\binom{v}{i}}{\binom{n}{2i+k-1}}\times\frac{k}{i+k}\times\frac{n-v-(i+k-1)}{n-(2i+k-1)}\right)\\ &+ \sum_{v=n-k+1}^{n}n\binom{n}{v}p^v(1-p)^{n-v}\ . \end{aligned} \tag{14}$$

$$Pr_{suc}(n,k)=\sum_{v=0}^{n-k}\binom{n}{v}p^v(1-p)^{n-v}\sum_{i=0}^{\min(v,\lfloor\frac{n-k}{2}\rfloor,n-v-k)}\frac{\binom{n-v}{i+k-1}\binom{e}{i}}{\binom{n}{2i+k-1}}\times\frac{k}{i+k}\times\frac{n-v-(i+k-1)}{n-(2i+k-1)}\ . \tag{15}$$

Counting for v iterations and the complexity of calculating F_j we have

$$\begin{aligned}L_v &= O(vk\log^2 k)+O(k(n-k))+O(v^2k)\\ &\quad +O(v(n-k))+O(v^3)\ . \end{aligned} \tag{13}$$

Note the computation complexity is measured by finite field multiplications, which is equivalent to m^2 bit exclusive-ORs. Since the correctable number of errors v is at most $(n-k)/2$, the decoding complexity is at most $O(k(n-k)^2)$. For small v, The second term $O(k(n-k))$ dominates, which corresponds to syndrome computation.

C. Average communication cost of decoding

In this section, we provide a probabilistic analysis of the cost of communication by determining the number of stages the algorithm needs to take, and the probability of successful execution. Given n storage nodes and (n,k) RS codes, it is easy to see that the fewest number of storage nodes to be accessed in the proposed scheme is k and the most is n. We assume that the CRC checking can always detect an error if it occurs. Without loss of generality, we assume that all failures are Byzantine failures. s crash-stop failures can be easily modeled by replacing n with $n-s$. An important metric of the decoding efficiency is the average number of accessed storage nodes when the probability of compromising each storage node is p. Failure to recover data correctly may occur in two cases. First, $v>n-k$, i.e., there are insufficient number of healthy storage nodes. Second, $\lfloor\frac{n-k}{2}\rfloor<v<n-k$, in which the sequence of accessing determines the outcome (success of failure) of the decoding process. For example, if the first v nodes accessed are all compromised nodes, correct decoding is impossible. In both cases, the decoding algorithm stops after n accesses and declares a failure. The communication cost is n.

The main result is summarized in the following theorem.

Theorem 1: With the progressive data retrieval scheme, the average number of access is given in Eq. (14). Eq. (15) gives the probability of successful decoding.

The proof is omitted due to space limitations. Interested readers are referred to [16].

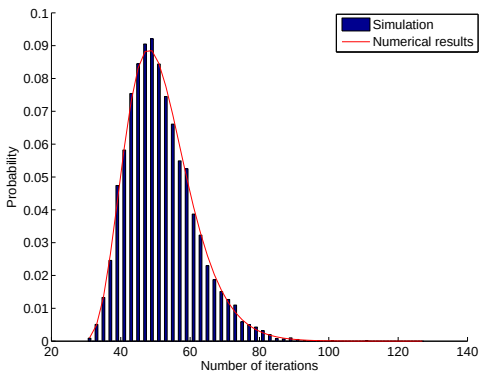


Fig. 2. Distribution of the number of node accesses from simulations and analysis. $n=127$, $k=31$ and $p=0.2$.

Numerical results: We verify the correctness of the analytical model using Monte-Carlo simulations implemented in Matlab. Figure 2 shows the distribution of the number of storage nodes accessed when the algorithm terminates. The bar chart gives the histogram from the Monte-Carlo simulations with 5000 runs, and the curve represents the result from the analytical model. We choose $n = 127$, $k = 31$ and $p = 0.2$ so that 5000 runs give sufficient statistics in the simulations. From Figure 2, it can be observed that the analytical results agree well with the simulation.

Next, we fix $n = 1023$, and vary the number of data-generating nodes k from 101 to 401 and the error probability p from 0 – 0.3. Figure 3 shows the increasing communication cost as the probability of failure increases. The number of crash-stop failure is set to be zero, and all Byzantine failures result in incorrect data. Clearly, when the error probability p is small, the communication cost is close to k and monotonically increases as p increases.

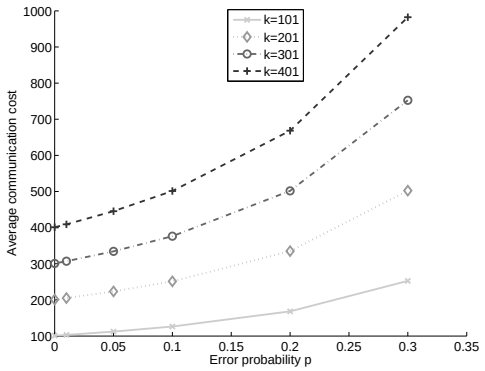


Fig. 3. Number of accesses vs. probability of failures. $n = 1023$ nodes, k = 101 – 401.

VI. Evaluation

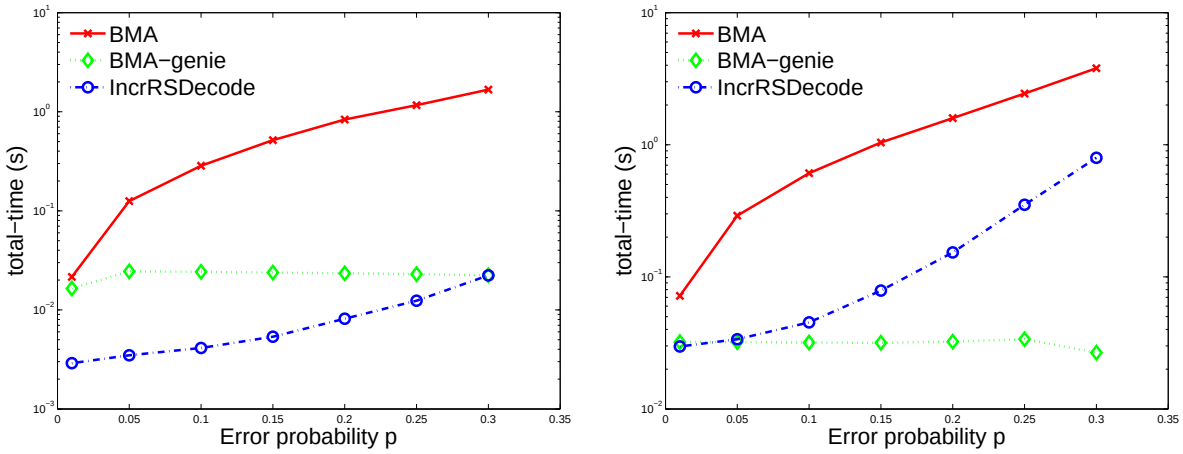
We have implemented the proposed and baseline algorithms in C. Evaluations are done on a desktop PC with a 2.66GHz Intel Xeon CPU, 4096 KB cache and available RAM of 2GB. Three algorithms are considered.

- *BMA* implements the Berlekamp-Massey (BMA) algorithm [4] for RS decoding. Similar to Algorithm 1, BMA *progressively* retrieves data from each storage node and performs decoding until the decoded symbols passes the CRC checks or failure is declared. However, decoding cannot be performed incrementally.
- *BMA-genie* knows *a priori* how many symbols are needed to successfully decode. BMA-genie decodes only once after retrieving sufficient number of symbols. Note that BMA-genie is impossible to implement in practice, and is included for comparison purpose only.
- *IncrRSDecode* implements the proposed progressive data retrieval scheme with the incremental decoding algorithm.

In place of BMA, either the Euclidean or Welch-Berlekamp algorithm could have been used. They have the same asymptotic time complexity. Figures 4–5 show the time it takes to correctly decode a data block. A randomly generated message is first partitioned into k information symbols and then encoded into $n = 1023$ coded symbols of length 10230 bits. Thus, the field size is $2^{10} = 1024$. A stored symbol is corrupted with an error probability p independently. Each point in the figures is an average of 50 runs.

Total computation time: Figure 4(a) and (b) illustrate the computation time (in log scale) spent in decoding when $k = 101$ and $k = 401$, respectively. The storage overhead n/k is 10.13 and 2.55 with the maximum number of errors correctable being 461 and 311. From Figure 4, we observe that the BMA and IncrRSDecode computation time increases as p increases. But the rate of increment in IncrRSDecode is much slower. When k is small or the redundancy is higher (Figure 4(a)), IncrRSDecode is faster than the genie-aided BMA. This is because in the genie-aided BMA, the computations of erasure polynomials (with $O((n-k)^2)$) dominate the decoding time in the case of small number of errors. In contrast, in IncrRSDecode, no erasure polynomials are computed.

Breakdown of the decoding computation: We break down the decoding computation time to understand the dominant operations in the algorithms as well as how the time spent in each stage of the algorithms changes as the error probability increases. The break down includes the time to find the error-locator polynomial (*elp-time*),

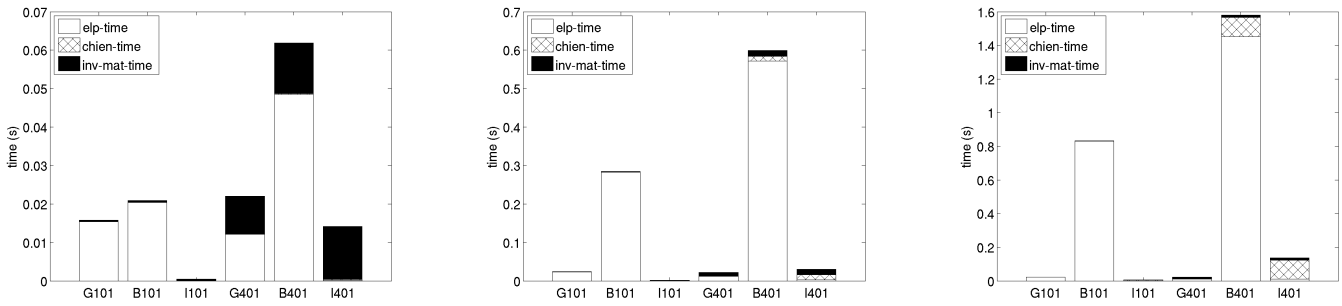


(a) k = 101 (b) k = 401

Fig. 4. Average computation time for decoding one group of encoded symbols

find the error locations (*chien-time*) and solve for the information polynomial (*inv-mat-time*). In Figure 1, the 1st and 2nd block shows the elp-time, and the 3rd and 4th blocks give chien-time and inv-mat-time, respectively.

When the error probability is low (Figure 5(a)), computation of error location polynomials appears to dominate for small k, while the matrix inversion time becomes significant when k is large. In our implementation, the cost of matrix inversion is quadratic in the number of symbols decoded. Chien search though asymptotically is the most time consuming procedure, it can be performed quite fast. When the error probability is high, computation of error location polynomials appear to dominate except in IncrRSDecode. Comparing Figure 5(a), (b) and (c), we observe that the computation time in matrix inversion is almost negligible (on the order of tens of milliseconds) in BMA and IncrRSDecode, and is comparable to that in BMA-genie (recall that BMA-genie knows the number of errors in advance and thus performs matrix inversion only once). This is because even though there are more errors with larger p (and thus more iterations), the decoding algorithm is likely to fail in or before Chien search (e.g., Algorithm 2 (Line 2)). Thus, in most cases, BMA and IncrRSDecode perform matrix inversion once.



(a) $p = 1\%$ (b) $p = 10\%$ (c) $p = 20\%$

Fig. 5. Average computational time breakdown for decoding one codeword

A couple of observations can be made from the evaluation results. First, IncrRSDecode is very efficient since it utilizes intermediate results from the previous iteration. Up to 20 times speed up can be attained, relative to classic RS decoding. Second, the computation complexity in Section V only provides the worst-case order analysis. In practice, the computation time in the average case can differ significantly in part due to hidden constant factors.

VII. Applications

In this section, we discuss the applications of the proposed progressive data retrieval scheme.

TABLE I

THE PERFORMANCE COMPARISON BETWEEN DECENTRALIZED ERASURE CODES [17] AND OUR PROPOSED SCHEMES.

	Decentralized erasure codes	Our schemes
Storage complexity	n	n
Communication complexity in storage	$5nln(k)$	n
Communication complexity in retrieving one unit of data	k	1
Communication complexity in retrieving all data	k	k
Computation complexity for encoding	$5\ln(k)T_0^2 n$	$kTmn$
Computation complexity for decoding (worst case)	$T_0^2 O(k^3)$	$f(v,n,k)$
Can detect error?	no	yes
Can correct error?	no	yes
Type of guarantee?	probabilistic	deterministic

[a] T_0 is the packet size generated by each data node and $GF(2^m)$ is the finite field the proposed scheme operates. Usually, T_0 can be 100 or even 1000 times larger than m.

[b] There are v errors occurred in the decoding procedure. If $v = 0$, then $f(v,n,k) = m^2 O(k \log^2 k)$; otherwise, $f(v,n,k) = m^2(O(vk \log^2 k) + O(k(n-k)) + O(v^2 k) + O(v(n-k)) + O(v^3))$.

A. Storage in sensor networks with multiple data sources

Recently, decentralized erasure codes have been applied in wireless sensor networks. Given k data nodes, and n storage nodes, the objective is to retrieve *all data* when a data collector accesses *any* k out of n storage nodes. In [17], randomized linear codes are used, where each data node routes its packet to $d(k) = \frac{5n}{k}\ln(k)$ storage nodes. Each storage node selects random and independent coefficient f_i in a finite field $\mathbb{F}_q$, and stores a linear combination of the received data (modulo q). In [10], a distributed implementation of fountain codes is proposed. Instead of pulling the data from candidate source nodes, a deterministic and probabilistic scheme are devised to push data from the source nodes to storage nodes. Both schemes share the common approach of coding at the storage nodes. As a result, sparsity of the coding matrix is necessary to reduce communication cost. This is the main argument made by the authors of [17] for not using RS codes. If RS codes are used and coding is done at the storage nodes, every data node needs to send its data to almost all n storage nodes, resulting in a communication complexity of $\Theta(nk)$. However, such a high complexity can be avoided if i) the source data is divisible into smaller coding units, and ii) encoding is performed at the data-generating nodes.

The proposed progressive data retrieval scheme can be directly applied when there are multiple data sources. Information from each data generating nodes is stored independently (rather than mixed at the storage nodes). Therefore, in addition to its ability to correct errors and tolerate Byzantine failures, the progressive data retrieval scheme also has the added benefit that partial retrieval of a subset of data sources is allowed. In contrast, due to the mixing at the storage node, decentralized erasure codes mandate "all-or-none", namely, either all storage nodes need to be accessed or none of the individual data sources can be retrieved. Table I provides a quantitative comparison between the cost of decentralized erasure codes and our proposed scheme. From Table I, we see that the proposed scheme outperforms decentralized erasure codes in almost all metrics.

B. Peer-to-peer tuple space

A tuple space is an implementation of the associative memory paradigm for parallel/distributed computing [18]. It provides a repository of tuples that can be accessed concurrently, and may be thought as a form of distributed shared memory. Implementations of tuple spaces have also been developed for Smalltalk, Java (JavaSpaces), Python, Ruby, TCL, Lua, Lisp, Prolog and the .NET framework. Typically, the tuple space is stored at a reliable central server. Processors produce pieces of data and use the data via *get* and *put* operations, respectively, based on unique identifiers of the data. In presence of a large number of processors (nodes), the tuple space server becomes a single point of failure and performance bottleneck. One way to alleviate this problem is to use peer-to-peer storage systems to host the tuple space. However, volatility and failure of storage nodes need to be considered. The distributed progressive retrieval scheme provides fault tolerance to erasures (unavailability of nodes) and Byzantine failures (corrupted data), and thus can be used to construct a reliable tuple space from an unreliable peer-to-peer storage system.

VIII. Conclusions

In this paper, we developed a solution using RS codes to spread information distributedly and redundantly for the handling of Byzantine failures. The data retrieval procedure is carried out in a progressive manner such that the communication cost is minimized while intermediate computation results can be utilized to greatly reduce computation cost. The efficient encode and decode primitives serve as a fundamental building block for survivable distribution storage systems. As future work, we will explore the applications of the proposed algorithms in practical systems.

References

[1] "RAID, Redundant Array of Independent Disks," http://en.wikipedia.org/wiki/Redundant_array_of_independent_disks.

[2] G. R. Goodson, J. J. Wylie, G. R. Ganger, and M. K. Reiter, "Efficient byzantine-tolerant erasure-coded storage," in *DSN '04: Proceedings of the 2004 International Conference on Dependable Systems and Networks*. Washington, DC, USA: IEEE Computer Society, 2004, p. 135.

[3] H. Krawczyk, "Distributed fingerprints and secure information dispersal," in *PODC '93: Proceedings of the twelfth annual ACM symposium on Principles of distributed computing*. New York, NY, USA: ACM, 1993, pp. 207–218.

[4] T. Moon, *Error Correction Coding: Mathematical Methods and Algorithms*. Hoboken, NJ: John Wiley & Sons, Inc., 2005.

[5] H. William, S. Teukolsky, W. Vetterling, and B. Flannery, *Numerical Recipes in C: The art of scientific computing*. Cambridge university press New York, NY, USA, 1988.

[6] J. Blomer, M. Kalfane, R. Karp, M. Karpinski, M. Luby, and D. Zuckerman, "An xor-based erasure-resilient coding scheme," ICSI Technical Report TR-95-048, 1995.

[7] P. Corbett, B. English, A. Goel, T. Grcanac, S. Kleiman, J. Leong, and S. Sankar, "Row-diagonal parity for double disk failure correction," in *Proceedings of the 3rd USENIX Symposium on File and Storage Technologies (FAST)*, 2004, pp. 1–14.

[8] M. Blaum, R. Roth, I. Div, A. Center, and C. San Jose, "On lowest density MDS codes," *IEEE Trans. Inform. Theory*, vol. 45, no. 1, pp. 46–59, 1999.

[9] M. Blaum, J. Brady, J. Bruck, J. Menon, and A. Vardy, *The EVENODD code and its generalization*. IEEE and Wiley Press, New York, 2001, pp. 187–208.

[10] Y. Lin, B. Liang, and B. Li, "Data persistence in large-scale sensor networks with decentralized fountain codes," in *Proceedings of the 26th IEEE INFOCOM*, 2007, pp. 6–12.

[11] J. S. Plank, J. Luo, C. D. Schuman, L. Xu, and Z. Wilcox-O'Hearn, "A performance evaluation and examination of open-source erasure coding libraries for storage," in *FAST '09: Proccedings of the 7th conference on File and storage technologies*. Berkeley, CA, USA: USENIX Association, 2009, pp. 253–265.

[12] I. S. Reed and X. Chen, *Error-Control coding for Data Networks*. Boston, MA: Kluwer Academic, 1999.

[13] K. Araki, M. Takada, and M. Morii, "On the efficient decoding of Reed-Solomon codes based on GMD criterion," in *Proc. of the International Symposium on Multiple-Valued Logic*, Sendai, Japan, May 1992, pp. 138–145.

[14] S. Lin and D. J. Costello, Jr., *Error Control Coding: Fundamentals and Applications*, 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, Inc., 2004.

[15] I. Gohberg and V. Olshevsky, "Fast algorithms with preprocessing for matrix-vector multiplication problem," *J. Complexity*, vol. 10, pp. 411–427, December 1994.

[16] Y. S. Han, S. Omiwade, and R. Zheng, "Survivable distributed storage with progressive decoding," Technical Report UH-CS-09-17, Department of Computer Science, University of Houston, 2009.

[17] A. G. Dimakis, V. Prabhakaran, and K. Ramchandran, "Decentralized erasure codes for distributed networked storage," *IEEE Trans. Inform. Theory*, vol. 52, no. 6, pp. 2809–2816, June 2006.

[18] D. Gelernter and N. Carriero, "Coordination languages and their significance," *Commun. ACM*, vol. 35, no. 2, pp. 97–107, 1992.