Sparsity Exploiting Erasure Coding for Resilient Storage and Efficient I/O Access in Delta based Versioning Systems

J. Harshan, Frédérique Oggier
School of Physical and Mathematical Sciences,
Nanyang Technological University, Singapore
Email:{jharshan,frederique}@ntu.edu.sg

Anwitaman Datta
School of Computer Engineering,
Nanyang Technological University, Singapore
Email:anwitaman@ntu.edu.sg

***Abstract*— In this paper we study the problem of storing reliably an archive of versioned data. Specifically, we focus on systems where the differences (deltas) between subsequent versions rather than the whole objects are stored - a typical model for storing versioned data. For reliability, we propose erasure encoding techniques that exploit the sparsity of information in the deltas while storing them reliably in a distributed back-end storage system, resulting in improved I/O read performance to retrieve the whole versioned archive. Along with the basic techniques, we propose a few optimization heuristics, and evaluate the techniques' efficacy analytically and with numerical simulations.**

I. Introduction

Using deltas is a well known technique to store a sequence of versions of a data object, where the differences between consecutive versions, rather than complete object instances themselves are maintained. It is used for a variety of applications, e.g.: (1) consider a user working on a local copy of his data, who explicitly saves/commits versions using a tool like the popular version management system, Subversion, a.k.a. SVN [8]. Then SVN is keeping the differences ('deltas') across consecutive versions, instead of all versions. (2) A very different kind of application is Wikipedia, which likewise keeps track of the differences between article contents, so that it is easy to track/revert changes, or identify vandalism. (3) Deltas are also exploited by cloud based back-up services, to reduce the network usage when uploading/downloading (synching) data, to give users old file versions from previous back-ups.

As illustrated by the previous examples, the notion of differences (deltas) is particularly suited to the storage of multiple versions of the same data objects.

In this paper, we are interested in looking at the back-end storage systems to store the versioned data reliably. This reliability in the back-end system is derived by applying two mechanisms, (i) distribution - i.e., deployment of multiple storage devices, so that even if some of the devices fail, there are other storage devices which can still serve the data, (ii) storing the data redundantly over this distributed storage network, so that despite the loss of individual storage devices, enough information is retained in the system such that the original data is preserved. The redundancy can be obtained by replicating the data, or by employing erasure coding techniques, which are known (see e.g. [2], [9]) to achieve better fault tolerance vis-a-vis storage overhead. This in turn has also led to a renewed interest in designing new erasure codes, aimed to address peculiarities of distributed storage systems [6]. Existing works are however predominantly geared towards storing immutable content, unlike the case of versioned data. The recent works which do focus on mutable content do so in the context of efficiently carrying out an update [7], [3], [5], [1], and thus focuses only on the storage of the latest version of the data.

In contrast to these existing works, we address the question of efficient storage of versioned data and design a novel erasure coding framework - Sparsity Exploiting Coding (SEC) - where the version differences are erasure encoded, instead of encoding each version individually, and the sparsity of information across versions is opportunistically exploited to optimize the system's (disk) I/O performance during retrieval of the versioned archive. We evaluate the efficacies of the presented framework using static resiliency analysis, and do so by studying both systematic and non-systematic maximum distance separable (MDS) codes (see Section II for a definition of MDS and systematic), and for different redundancy placement strategies. Our analysis demonstrates that the number of I/O accesses required is significantly reduced when retrieving the multiple versioned data archive. Due to a lack of consensus on proper workloads to study such systems (see e.g. [10]), we experimented with a few example scenarios, where I/O reduction of up to 20% were observed to retrieve a data object with 5 versions, while reductions between 4-13% were observed for even just two versions, in randomized experiments where probability distributions on the sparsity of differences were chosen to study scenarios ranging from unfavorable to favorable to the proposed framework.

II. System Model & Preliminaries

We start by providing a formal framework that describes erasure coding. Let $\mathbb{F}_q$ denote the finite field with q elements, where q is a prime power, typically a power of 2 here. We will denote by $\mathbf{x} \in \mathbb{F}_q^k$ a data object to be stored over a storage

network, that is, the data object is seen as a vector of k blocks taking value in the alphabet $\mathbb{F}_q$. We assume a fixed sized data object, and in particular that the modifications of this object do not change its length, which does not readily translate to application level objects such as files or directories. Thus, we implicitly assume that the application level objects are split and transformed into fixed sized objects (arguably with necessary zero padding), which is then used as the input $\mathbf{x} \in \mathbb{F}_q^k$ for the encoding process. The nuances of this transformation, as well as the subsequent reassembly of the whole files to be used by the applications is beyond the scope of this work, and all our subsequent discussions will instead be centered around the abstract data objects represented by $\mathbf{x} \in \mathbb{F}_q^k$.

We consider the scenario of an erasure coding based distributed storage system, where fault tolerance is achieved using linear erasure codes. Recall that to archive an object $\mathbf{x} \in \mathbb{F}_q^k$, we first encode it using an (n, k) linear code, that is $\mathbf{x}$ is mapped to the codeword

$$\mathbf{c} = \mathbf{G}\mathbf{x} \in \mathbb{F}_q^n,\ n > k, \tag{1}$$

for $\mathbf{G}$ an $n \times k$ matrix with coefficients in $\mathbb{F}_q$ called *generator matrix* or sometimes coding matrix. The ratio k/n is called the *rate* of the code. We use the term *systematic* to refer to a codeword $\mathbf{c}$ whose k first components are $\mathbf{x}$, that is $c_i = x_i$, $i = 1, \ldots, k$. Once a codeword of length n is obtained, all the n coefficients c_i, $i = 1, \ldots, n$ are stored across n distinct nodes of the network. We say that an (n, k) linear code is MDS (which stands for Maximum Distance Separable) when any patterns of $n - k$ failures can be tolerated.

Let $\mathbf{x}_1 \in \mathbb{F}_q^k$ be the first version of a data object to be stored. The data owner may at any time decide to modify it, giving rise to a new version of this data object, denoted by $\mathbf{x}_2 \in \mathbb{F}_q^k$. More generally, a new version $\mathbf{x}_{j+1}$ is obtained from $\mathbf{x}_j$, and over time, we obtain a sequence $\{\mathbf{x}_j \in \mathbb{F}_q^k,\ j = 1, 2, \ldots, L < \infty\}$ of different versions of a data object, to be stored in the network. The bit level-wise modifications between two successive versions are modelled by

$$\mathbf{x}_{j+1} = \mathbf{x}_j + \mathbf{z}_{j+1}, \tag{2}$$

where $\mathbf{z}_{j+1} \in \mathbb{F}_q^k$ keeps track of the changes in the j-th update.

From a user point of view, the difference between two consecutive versions is determined by the application semantics. From a back-end storage system view however, we are interested in sequences $\{\mathbf{x}_j \in \mathbb{F}_q^k,\ j = 1, 2, \ldots, L < \infty\}$ of data objects in their bit level representation, and exploit opportunistically the fact that often $\mathbf{x}_{j+1}$ and $\mathbf{x}_j$ may have little differences at the bit level or said differently $\mathbf{z}_{j+1}$ in (2) is sparse (formally defined in Definition 1). As motivated in the introduction, version management systems like SVN [8] store differences (deltas) across versions, and are natural candidates to benefit from the proposed coding strategy.

Definition 1: For some integer $1 \leq \gamma < k$, a vector $\mathbf{z} \in \mathbb{F}_q^k$ is said to be γ-sparse if it contains at most γ non-zero entries.

Once $\mathbf{z}_{j+1} \in \mathbb{F}_q^k$ is γ-sparse, it suggests that it should be possible to access it more efficiently (with less I/O reads) than a normal data object. Indeed, the ideal case would be if one could only use γ I/O reads, since the other $k - \gamma$ positions contain zeroes, without having to look for the non-zero positions. We will show in next section, by proposing an explicit coding strategy, that it is possible to reduce the number of I/O reads from k to 2γ I/O reads, which thus becomes beneficial when $\gamma < \frac{k}{2}$. The ideal case of γ I/O reads is not achieved, since in practice we do not know the positions of the zeroes.

1: **procedure** ENCODE($\mathcal{X}, \mathbf{G}$)
2: **FOR** $0 \leq j \leq L - 1$
3: **IF** $j = 0$
4: return $\mathbf{c}_1 = \mathbf{G}\mathbf{x}_1$;
5: **ELSE** (*This part summarizes Step* $j + 1$ *in text*)
6: Compute $\mathbf{z}_{j+1} = \mathbf{x}_{j+1} - \mathbf{x}_j$;
7: Compute and Store γ_{j+1};
8: return $\mathbf{c}_{j+1} = \mathbf{G}\mathbf{z}_{j+1}$;
9: **END IF**
10: **END FOR**
11: **end procedure**

Fig. 1. Encoding Procedure for SEC

III. SPARSITY EXPLOITING CODING (SEC)

Let $\{\mathbf{x}_j \in \mathbb{F}_q^k,\ 1 \leq j \leq L\}$ be the sequence of versions of a data object to be stored in the network, where $\mathbf{x}_j$ is the jth version (or version at the j-th instant of time). The number of components modified from $\mathbf{x}_j$ to $\mathbf{x}_{j+1}$ is reflected in the vector $\mathbf{z}_{j+1} = \mathbf{x}_{j+1} - \mathbf{x}_j$ in (2) which is then γ_{j+1}-sparse (see Definition 1) for some $1 \leq \gamma_{j+1} \leq k$. We propose an encoding strategy using an (n, k) linear erasure code (see (1)) which exploits the sparsity of the differences across updates, thus referred to as *sparsity exploiting coding (SEC)*. Note that the value γ_{j+1} may a priori vary across updates of the same object and across different objects, and that sparsity is exploitable only when $\gamma_{j+1} < \frac{k}{2}$.

A. Object Encoding

The **basic SEC method** to encode the jth version $\mathbf{x}_{j+1}$, $j \leq L - 1$, using deltas is formally given by:

Step $j + 1$**.** To encode the $(j + 1)$-th version, the difference vector

$$\mathbf{z}_{j+1} = \mathbf{x}_{j+1} - \mathbf{x}_j$$

and the corresponding sparsity level γ_{j+1} are computed. Then the object $\mathbf{z}_{j+1}$ is encoded as either

$$\mathbf{c}_{j+1} = \mathbf{G}_S \mathbf{z}_{j+1},$$

if the coding matrix $\mathbf{G}_S \in \mathbf{F}_2^{n \times k}$ is in systematic form, or

$$\mathbf{c}_{j+1} = \mathbf{G}_N \mathbf{z}_{j+1},$$

if $\mathbf{G}_N \in \mathbf{F}_2^{n \times k}$ is not in systematic form.

The sparsity exploiting coding (SEC) procedure is summarized algorithmically in Figure 1. The input and the output of the algorithm are $\mathcal{X} = \{\mathbf{x}_{j+1} \in \mathbb{F}_q^k,\ 0 \leq j \leq L - 1\}$ and $\{\mathbf{c}_{j+1},\ 0 \leq j \leq L - 1\}$, respectively.

The above description emphasizes the differential nature of the proposed SEC, where the first version is encoded in full while the subsequent versions are encoded via their subsequent differences. This leads to a recursive encoding of the object $\mathbf{x}_l$ for $l > 1$, whose overall storage pattern is $\{\mathbf{x}_1, \mathbf{z}_2, \ldots, \mathbf{z}_L\}$.

There are two main missing ingredients to complete the description of the proposed SEC: (1) explicit constructions for the coding matrices $\mathbf{G}_S$ and $\mathbf{G}_N$ that facilitate the recovery of $\mathbf{z}_{j+1}$ with fewer than k I/O reads when $\gamma_{j+1} < \frac{k}{2}$, and (2) how the data placement of the objects $\{\mathbf{x}_1, \mathbf{z}_2, \ldots, \mathbf{z}_L\}$ should be done across the sets of nodes $\{\mathcal{N}_1, \mathcal{N}_2, \ldots, \mathcal{N}_L\}$, where the set $\mathcal{N}_{j+1}$ of nodes is used to store the components of $\mathbf{c}_{j+1}$. Both issues are equally important, and thus deserve a (sub)section of their own (see Subsection III-B for the code design, and Section IV for the data allocation).

We conclude this subsection with some remarks, including two possible variations of the above SEC. These variations are not mutually exclusive and can be used in conjunction.

Optimized Step $j+1$**.** A first variant of **Step** $j+1$ is obtained by encoding a whole object if the sparsity level is too high, namely: Store $\mathbf{c}_{j+1} = \mathbf{G}_S \mathbf{z}_{j+1}$ (or $\mathbf{c}_{j+1} = \mathbf{G}_N \mathbf{z}_{j+1}$) only when $\gamma_{j+1} < \frac{k}{2}$, and store $\mathbf{c}_{j+1} = \mathbf{G}_S \mathbf{x}_{j+1}$ (or $\mathbf{c}_{j+1} = \mathbf{G}_N \mathbf{z}_{j+1}$), otherwise.

The I/O advantages of the **Optimized Step** $j+1$ will be discussed with an example in Subsection III-D.

Reversed SEC. For applications where the latest archived versions of the object are frequently accessed, a variant of the proposed SEC method could be employed where the order of storing the difference vectors is reversed as $\{\mathbf{z}_2, \mathbf{z}_3, \ldots, \mathbf{z}_L, \mathbf{x}_L\}$, so as to favor the latest version access.

Finally, note that the SEC stores only the deltas, yet there is an implicit assumption that $\mathbf{x}_j$ is known, in order to compute its difference with $\mathbf{x}_{j+1}$, $j = 1, \ldots, L-1$. A practical way to satisfy this requirement is to cache a full copy of the latest version $\mathbf{x}_j$, until a new version $\mathbf{x}_{j+1}$ arrives. Keeping a cache of the latest version also helps in improving the response time and overheads of data read operations in general. Alternatively, the second variation above, Reversed SEC, can be applied, where the latest version is encoded fully, along with differences of older versions.

B. Object Retrieval with Non-Systematic SEC

Suppose that the L versions of a data object have been archived, and the user needs to retrieve $\mathbf{x}_l$ for some $1 < l \leq L$. We discuss the procedure to retrieve $\mathbf{x}_1, \mathbf{z}_2, \ldots, \mathbf{z}_l$ from $\mathcal{N}_1, \mathcal{N}_2, \ldots, \mathcal{N}_l$. The recovery procedure depends on the structure of the SEC generator matrix, and hence, we explain the procedure considering two separate cases where the coding matrix is (i) $\mathbf{G}_N$ (non-systematic) and (ii) $\mathbf{G}_S$ (systematic). We start with the non-systematic case.

To retrieve $\mathbf{x}_1$, choose a subset of k nodes from $\mathcal{N}_1$ to obtain

$$\mathbf{y} = \mathbf{G}_{sub} \mathbf{x}_1,$$

where $\mathbf{G}_{sub} \in \mathbb{F}_q^{k \times k}$ is a submatrix of $\mathbf{G}_N$ which is invertible, then recover $\mathbf{x}_1$ as

$$\mathbf{x}_1 = \mathbf{G}_{sub}^{-1} \mathbf{y}.$$

We need to make sure that such a submatrix always exists, which gives us a first design criterion:

- **Criterion 1.** There is at least one $k \times k$ submatrix of $\mathbf{G}_N$ that is full rank (to retrieve $\{\mathbf{x}_1, \mathbf{z}_j \mid \gamma_j \geq \frac{k}{2}\}$).

The retrieval procedure for $\{\mathbf{z}_j, 2 \leq j \leq l\}$ depends on the corresponding sparsity levels $\{\gamma_j, 2 \leq j \leq l\}$. If $\gamma_j \geq \frac{k}{2}$, then $\mathbf{z}_j$ is recovered using the same procedure as that of $\mathbf{x}_1$. If $\gamma_j < \frac{k}{2}$, choose a subset of $2\gamma_j$ nodes from $\mathcal{N}_j$ to obtain

$$\mathbf{y} = \mathbf{G}_{\gamma_j} \mathbf{z}_j,$$

where $\mathbf{G}_{\gamma_j} \in \mathbb{F}^{2\gamma \times k}$ is a submatrix of $\mathbf{G}$. This gives us our second code design criterion, which follows from [11]:

Proposition 1: If any 2γ columns of the $2\gamma \times k$ matrix Φ are linearly independent, then it is possible to uniquely recover the γ-sparse vector $\mathbf{z}$ from $\Phi\mathbf{z}$.

Proof: Since $2\gamma < k$, we can view Φ as the parity check matrix of a $(k, k-2\gamma)$ linear code $\mathcal{C}$ in $\mathbb{F}_q^k$. For the matrix Φ, if any 2γ columns of Φ are linearly independent, then the minimum Hamming distance of $\mathcal{C}$ is at least $2\gamma+1$. Thus, from the properties of a linear code, $\mathcal{C}$ can correct all error patterns of weight less than or equal to γ, which in turn implies that it is possible to uniquely recover a γ-sparse vector $\mathbf{x}$. ■

- **Criterion 2.** For every $\gamma_j < \frac{k}{2}$, there is at least one $2\gamma_j \times k$ submatrix of $\mathbf{G}_N$ for which any $2\gamma_j$ columns are linearly independent (to retrieve $\{\mathbf{z}_j \mid \gamma_j < \frac{k}{2}\}$.)

It is clear that a minimum of k I/O reads are needed to retrieve $\mathbf{x}_1$. However, to recover $\mathbf{z}_j$ for $2 \leq j \leq l$, the number of I/O reads is $\min(2\gamma_j, k)$. Overall, the total number of I/O reads to retrieve $\mathbf{x}_l$ in the differential set up is

$$\eta(\mathbf{x}_l) = k + \sum_{j=l'+1}^{l} \min(2\gamma_j, k), \tag{3}$$

where $l' = 1$ for the basic encoding method (**Step** $j+1$, $j \leq L-1$). For the optimized method (**Optimized Step** $j+1$, $j \leq L-1$), $l' \leq l$ corresponds to the most recent version such that $\gamma_{l'} \geq \frac{k}{2}$. Finally, since the decoding method is differential, the procedure to read the first l versions is the same as that for reading $\mathbf{x}_l$ for both the basic and the optimized method. Hence, the total number of I/O reads to retrieve the first l versions is

$$\eta(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_l) = k + \sum_{j=2}^{l} \min(2\gamma_j, k). \tag{4}$$

When there are node failures, different contenders for $\mathbf{G}_{sub}$ and $\mathbf{G}_{\gamma_j}$ give us the option to retrieve the objects before the node repair process. Hence, it is beneficial for the overall system performance to relax the condition of *at least one submatrix* to *several submatrices* in the criteria 1 and 2.

Example 1: Consider an (n, k) maximum distance separable (MDS) code whose generator matrix $\mathbf{G}_N$ is given by the

Cauchy matrix

$$\mathbf{G}_N = \begin{bmatrix} g_{1,1} & g_{1,2} & \cdots & g_{1,k} \\ g_{2,1} & g_{2,2} & \cdots & g_{2,k} \\ \vdots & \vdots & \vdots & \vdots \\ g_{n,1} & g_{n,2} & \cdots & g_{n,k} \end{bmatrix}, \quad (5)$$

where $g_{i,j} = \frac{1}{h_i - f_j}$ for $\{h_i \in \mathbb{F}_q, 1 \leq i \leq n\}$ and $\{f_j \in \mathbb{F}_q, 1 \leq j \leq k\}$ such that $h_i - f_j \neq 0 \ \forall \ i, j$. Since any square submatrix of a Cauchy matrix is full rank over a finite field [4], any $2\gamma_j \times k$ submatrix of $\mathbf{G}_N$ satisfies Proposition 1. Thus, MDS codes from Cauchy matrices are readily applicable in the proposed differential set up.

C. Object Retrieval with Systematic SEC

We next consider the case where the L versions of a data object are archived using a systematic code, and the user needs to retrieve $\mathbf{x}_l$ for some $1 < l \leq L$. The generator matrix of the systematic code is of the form

$$\mathbf{G}_S = \begin{bmatrix} \mathbf{I}_k \\ \mathbf{B} \end{bmatrix},$$

where $\mathbf{I}_k$ is the $k \times k$ identity matrix and $\mathbf{B} \in \mathbb{F}_q^{n-k \times k}$ generates the $n-k$ parity symbols. Since the code is systematic, the objects $\mathbf{x}_1$ and $\{\mathbf{z}_j, \forall \gamma_j \geq \frac{k}{2}\}$ can be retrieved by downloading the contents from the k systematic nodes. If $\gamma_j < \frac{k}{2}$, then choose a subset of $2\gamma_j$ nodes from $\mathcal{N}_j$ to obtain

$$\mathbf{y} = \mathbf{G}_{\gamma_j} \mathbf{z}_j,$$

where $\mathbf{G}_{\gamma_j} \in \mathbb{F}^{2\gamma \times k}$ is a submatrix of $\mathbf{G}_S$. If $\mathbf{G}_{\gamma_j}$ satisfies Proposition 1, then $\mathbf{z}_j$ can be recovered. Note that the submatrix satisfying **Criterion 2** is most likely to come from $\mathbf{B}$. Indeed, suppose that any row of $\mathbf{I}_k$ is taken, then since its length k satisfies $k > 2\gamma$, any pattern of consecutive 2γ zeroes results in a $2\gamma \times 2\gamma$ submatrix which is not full rank, which is likely to happen whenever $2\gamma << k$. Restricting to the matrix $\mathbf{B}$ which has only $n-k$ rows leads to the constraint that γ_j-sparse updates can be recovered with $2\gamma_j$ I/O reads only for $2\gamma_j < n-k$, that is $\gamma_j < \frac{n-k}{2}$. The number of I/O reads to retrieve $\mathbf{z}_j$ with a systematic code whose matrix B satisfies **Criterion 2** is

$$\eta_j = \begin{cases} 2\gamma_j, \text{ if } \gamma_j \leq \frac{n-k}{2} \\ k, \text{otherwise.} \end{cases}$$

Since we are interested in $\gamma_j < \frac{k}{2}$, either we have $\frac{n-k}{2} < \frac{k}{2} \iff \frac{k}{n} > \frac{1}{2}$, which implies that systematic erasure coding can only recover less than $\lceil \frac{k}{2} \rceil - 1$ sparse levels with reduced I/O, or, if $\frac{n-k}{2} \geq \frac{k}{2} \iff \frac{k}{n} \leq \frac{1}{2}$, and it can recover up to $\lceil \frac{k}{2} \rceil - 1$ sparse levels (which is the same as that of non-systematic encoding). The total number of I/O reads to retrieve $\mathbf{x}_l$ in the differential set up is

$$\eta(\mathbf{x}_l) = k + \sum_{j=l'+1}^{l} \eta_j,$$

where $l' = 1$ for basic encoding. For the optimized method, $l' \leq l$ corresponds to the most recent version such that $\gamma_{l'} \geq \frac{k}{2}$. The total number of I/O reads to retrieve the first l versions is also

$$\eta(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_l) = k + \sum_{j=2}^{l} \eta_j.$$

Example 2: Similarly to Example 1, an $(n-k) \times k$ Cauchy matrix can be used to construct the matrix $\mathbf{B}$, resulting in an MDS code.

D. I/O Benefits: An Illustrative Example

Consider a differential storage system that stores $L = 5$ versions of an object of size $k = 10$ using a $(20, 10)$ erasure code that satisfies the desired design criteria. Let the sparsity levels of subsequent versions be $\{\gamma_j \mid 2 \leq j \leq L\} = \{3, 8, 3, 6\}$. We also assume that $\mathbf{x}_1$ itself is not sparse. We compute the number $\eta(\mathbf{x}_l)$ of I/O reads needed to retrieve the lth version $\mathbf{x}_l$. Since the employed code has rate $\frac{k}{n} = \frac{1}{2}$, the below given I/O read numbers are applicable for both systematic and non-systematic cases.

Basic encoding (Step $j+1$, $j \leq L-1$). In this technique, the stored objects are $\{\mathbf{x}_1, \mathbf{z}_2, \mathbf{z}_3, \mathbf{z}_4, \mathbf{z}_5\}$. The number of I/O reads to retrieve $\mathbf{x}_1, \mathbf{z}_2, \mathbf{z}_3, \mathbf{z}_4, \mathbf{z}_5$ are $10, 6, 10, 6, 10$, respectively. Thus, $\{\eta(\mathbf{x}_l), 1 \leq l \leq 5\}$ is $\{10, 16, 26, 32, 42\}$. The total I/O reads to recover all the 5 versions is 42 (instead of 50 for the non-differential method). Thus, there is a reduction in the number of I/O reads to retrieve all the versions.

Optimized encoding (Optimized Step $j+1$, $j \leq L-1$). In this method, the stored objects are $\{\mathbf{x}_1, \mathbf{z}_2, \mathbf{x}_3, \mathbf{z}_4, \mathbf{x}_5\}$. The number of I/O reads to retrieve $\mathbf{x}_1, \mathbf{z}_2, \mathbf{x}_3, \mathbf{z}_4, \mathbf{x}_5$ are $10, 6, 10, 6, 10$, respectively. Thus, $\{\eta(\mathbf{x}_l), 1 \leq l \leq 5\}$ is $\{10, 16, 10, 16, 10\}$. However, the number of I/O reads to recover all the 5 versions is same as the basic encoding method. Note that the number of I/O reads to retrieve individual versions is lower than the basic encoding method.

IV. Static Resilience Analysis

We next compute the static resilience (the amount of failures that the system tolerates based on the initial redundancy, if no further remedial actions are taken) of the proposed SEC coding strategies that exploit the sparsity across subsequent versions. We suppose that L versions of a data object are stored, namely the pattern of stored data is $\{\mathbf{x}_1, \mathbf{z}_2, \mathbf{z}_3, \mathbf{z}_{L-1}, \mathbf{z}_L\}$, and encoded pieces for any of these versions are stored in n nodes. The static resilience is computed for two practical redundancy placement choices: a (**dispersed placement**), where differences and first version are all stored in different nodes, involving a total of nL distinct nodes, and a (**colocated placement**), when all the versions' encoded pieces are stored in a common set of n nodes.

We use the non-differential strategy where each version is coded and stored individually as a baseline, and demonstrate that for both placements, both the systematic and non-systematic SEC schemes achieve the same resiliency as non-differential coding for a given overall storage overhead (i.e., there is no resiliency compromise), even though both new

coding technique results in a reduction in access I/O when retrieving all the previous versions of the data.

When comparing systematic and non-systematic strategies, a key difference is that the number of submatrices of $\mathbf{G}_S$ satisfying Criterion 2 is fewer compared to that of $\mathbf{G}_N$ (as explained in Section III-C). We will demonstrate that this reduction in the number of options results in a poorer resilience of individual version differences for the systematic SEC, compared to its non-systematic counterpart. However, the best resilience, taking into account all the versions of the data, is realized, for each of the two coding strategies, when the encoded pieces for each of the versions (or differences) are stored in the same set of nodes - which ultimately leads to the same net resilience for all the coding strategies.

In the following, we assume that individual nodes fail with a probability p and the failure events are independent.

A. SEC Analysis

We start by computing the probability of losing one individual version among $\{\mathbf{x}_1, \mathbf{z}_2, \mathbf{z}_3, \mathbf{z}_{L-1}, \mathbf{z}_L\}$. We assume SEC are MDS, since Cauchy matrices based SEC are MDS.

For a MDS SEC, whether it is systematic or not, any k nodes storing the encoded pieces of $\mathbf{x}_1$ are sufficient to retrieve it. Thus $\mathbf{x}_1$ is lost if the event

$$\mathcal{E}_1 \quad = \quad \{n-k+1 \text{ or more nodes fail}\}$$

occurs. The probability of losing $\mathbf{x}_1$ is then given by

$$\text{Prob}_N(\mathcal{E}_1) = \text{Prob}_S(\mathcal{E}_1) = \sum_{j=0}^{k-1} C_{n-j}^{n} p^{n-j}(1-p)^j. \qquad (6)$$

For any other arbitrary version l ($2 \leq l \leq L$) in the **non-systematic case**, where only the difference with its previous version $\mathbf{z}_l$ is stored, any $\upsilon_l = \min(2\gamma_l, k)$ nodes suffice to retrieve $\mathbf{z}_l$ since any $\upsilon_l \times k$ submatrix of $\mathbf{G}_N$ satisfies Criterion 2. As a result, the object $\mathbf{z}_l$ is lost if the event

$$\mathcal{E}_l \quad = \quad \{n-\upsilon_l+1 \text{ or more nodes fail}\}$$

occurs. The probability of losing $\mathbf{z}_l$ is given by

$$\text{Prob}_N(\mathcal{E}_l) = \sum_{j=0}^{\upsilon_l-1} C_{n-j}^{n} p^{n-j}(1-p)^j. \qquad (7)$$

Similarly, in the **systematic case**, for $\mathbf{z}_l$ with $\gamma_l \geq \frac{k}{2}$, we have

$$\text{Prob}_S(\mathcal{E}_l) \quad = \quad \text{Prob}_S(\mathcal{E}_1). \qquad (8)$$

However, for $\mathbf{z}_l$ with $l < \frac{k}{2}$, not all combinations of $2\gamma_l$ nodes can retrieve $\mathbf{z}_l$ since only few $2\gamma_l \times k$ submatrices of $\mathbf{G}_N$ satisfies Criterion 2. As a result, the probability of losing $\mathbf{z}_l$ is strictly lower bounded as

$$\text{Prob}_S(\mathcal{E}_l) \quad > \quad \sum_{j=0}^{2\gamma_l-1} C_{n-j}^{n} p^{n-j}(1-p)^j. \qquad (9)$$

Thus, we have

$$\text{Prob}_S(\mathcal{E}_l) \geq \text{Prob}_N(\mathcal{E}_l) \text{ for } 1 \leq l \leq L, \qquad (10)$$

where the inequality holds when $\gamma_l < \frac{k}{2}$, and equality otherwise.

Dispersed placement: In a dispersed placement, the probability of retaining all the L versions of the object are

$$P_d(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_L) = \prod_{l=1}^{L}(1 - \text{Prob}(\mathcal{E}_l)). \qquad (11)$$

Using the inequality of (10) in (11), it is clear that non-systematic SEC provides at least same level of resilience as that of systematic codes if dispersed placement is employed.

Colocated placement: In a colocated placement, the probability of retaining all the L versions of the object are

$$P_c(\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_L) \quad = \quad 1 - \text{Prob}(\cup_{l=1}^{L}\mathcal{E}_l), \qquad (12)$$

For colocated placement, all the L objects $\{\mathbf{x}_1, \mathbf{z}_2, \cdots, \mathbf{z}_L\}$ can be recovered with probability one if any k nodes are alive. Although some objects $\{\mathbf{z}_l \mid \gamma_l < \frac{k}{2}\}$ can be retrieved even with the loss of $n - 2\gamma_l > n - k$ nodes, such failure patterns nevertheless result in the loss of $\mathbf{x}_1$, thereby making the recovery of all the L versions impossible. Hence, the existence of any k live nodes guarantees the recovery of $\{\mathbf{x}_1, \mathbf{z}_2, \cdots, \mathbf{z}_L\}$ and this argument is applicable for both systematic and non-systematic SEC. With that, the probability of retaining all the L versions is same for both systematic and non-systematic SEC and is given by

$$P_c(\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_L) \quad = \quad 1 - \text{Prob}_S(\mathcal{E}_1). \qquad (13)$$

Note that if any $2\gamma_l$ nodes are sufficient to recover the sparse updates for the the non-systematic erasure codes, only specific patterns of $2\gamma_l$ nodes are applicable for the systematic codes. More discussion on the effects of different possible options to recover sparse updates are discussed in Section V.

Finally, comparing (13) and (11), we conclude that colocated placement yields higher resilience for both systematic and non-systematic erasure codes than the dispersed placement. Henceforth, the resilience values calculated in colocated placement are used as the resilience of the systematic and non-systematic SEC methods.

B. Non-differential Coding (Baseline)

Any k nodes of the n nodes where encoded pieces of a version $\mathbf{x}_l$ for $1 \leq l \leq L$ are stored, is adequate to retrieve that version, i.e., $\text{Prob}_{ND}(\mathcal{E}_1) = \text{Prob}_N(\mathcal{E}_1)$ in (6).

Thus, for dispersed placement, probability of retaining $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_L\}$ is

$$P_d(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_L) = (1 - \text{Prob}_{ND}(\mathcal{E}_1))^L, \qquad (14)$$

For $l \geq 2$, we have $\text{Prob}_{ND}(\mathcal{E}_1) \geq \text{Prob}_N(\mathcal{E}_l)$, where the inequality holds when $\gamma_l < \frac{k}{2}$, and equality otherwise. Using the above inequality in (14), it is clear that non-differential erasure codes provide at most as much resilience as that of differential non-systematic SEC in dispersed placement.

For the case of collocated placement while using non-differential coding,

$$P_c(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_L) \quad = \quad (1 - \text{Prob}_{ND}(\mathcal{E}_1)). \qquad (15)$$

$$\geq \quad P_d(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_L) \qquad (16)$$

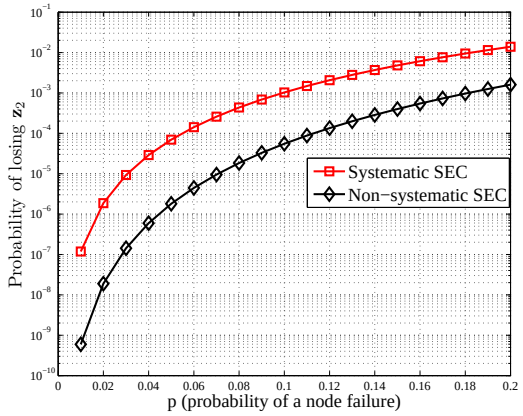

Fig. 2. Probability of losing the 1-sparse difference object $\mathbf{z}_2$ in example of Subsection IV-C: $\text{Prob}_S(\mathcal{E}_2)$ in (20) and $\text{Prob}_N(\mathcal{E}_2)$ in (18) for the systematic and non-systematic SEC respectively.

Therefore, non-differential erasure codes have same resilience as that of the differential SEC codes, when encoded information about all the versions are colocated. Similar to the SEC methods, colocated placement is also optimal for the non-differential encoding method.

C. Example

We revisit the calculations above with a concrete example with specific parameter choices. This helps us to obtain specific values and compare the static resilience of the different schemes explicitly.

a) Set-up: Consider a system that stores 2 versions of a data object. Let the original data object be a binary file of size 3KB. We represent this object as a 3-length vector $\mathbf{x}_1$ over the finite field of size 1KB, i.e., let $\mathbf{x}_1 \in \mathbb{F}_q^3$ where $q = 1024$. Further, let the second version of the object be such that only the first 1KB of the binary file has been modified. It is important to note that the quantum and location of changes are not known a priori, and the coding scheme, decided in advance, has to work irrespective of the specificities of the update. In the finite field level, the second version is represented as $\mathbf{x}_2 = \mathbf{x}_1 + \mathbf{z}_2$, where $\mathbf{z}_2$ is 1-sparse given by

$$\mathbf{z}_2 = \begin{bmatrix} X \\ 0 \\ 0 \end{bmatrix} \in \mathbb{F}_q^3,$$

where X denotes a non-zero element of $\mathbb{F}_q$.

b) The non-systematic case: For the system parameters of our set-up, we pick a $(6,3)$ non-systematic MDS code whose generator matrix $\mathbf{G}_N \in \mathbf{F}_q^{6\times 3}$ is carved from a Cauchy matrix. Subsequently, $\mathbf{z}_2$ is encoded using $\mathbf{G}_N$ to obtain the codeword $\mathbf{c}_2 = \mathbf{G}_N\mathbf{z}_2 \in \mathbb{F}_q^6$. It is clear that the number of I/O reads needed to retrieve the first 2 versions is 5. Since the code is MDS, the probability of losing $\mathbf{x}_1$ is given by

$$\text{Prob}_N(\mathcal{E}_1) = p^6 + C_5^6 p^5(1-p) + C_4^6 p^4(1-p)^2. \quad (17)$$

Similarly, the probability of losing $\mathbf{z}_2$ is given by

$$\begin{aligned} \text{Prob}_N(\mathcal{E}_2) &= p^6 + C_5^6 p^5(1-p), \quad (18)\\ &< \text{Prob}_N(\mathcal{E}_1). \end{aligned}$$

The proposed non-systematic SEC opportunistically exploits sparsity in $\mathbf{z}_2$ to provide higher resilience for the object $\mathbf{z}_2$ than the object $\mathbf{x}_1$. However, in colocated placement, the resilience for both the objects $\mathbf{x}_1$ and $\mathbf{x}_2$ is dominated by that for $\mathbf{x}_1$. Hence, we have

$$P_c(\mathbf{x}_1, \mathbf{x}_2) = 1 - \left(p^6 + C_5^6 p^5(1-p) + C_4^6 p^4(1-p)^2\right).$$

c) The systematic case: Now let us pick a $(6,3)$ systematic erasure code whose generator matrix is given by

$$\mathbf{G}_S = [\mathbf{I}_3 \ \mathbf{B}^T]^T \in \mathbf{F}_q^{6\times 3}$$

where $\mathbf{B} \in \mathbf{F}_q^{3\times 3}$ is a Cauchy matrix. We encode the first version $\mathbf{x}_1 \in \mathbb{F}_q^3$ as

$$\mathbf{c}_1 = \mathbf{G}_S\mathbf{x}_1 = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{B}\mathbf{x}_1 \end{bmatrix} \in \mathbb{F}_q^6.$$

The storage overhead for the first version is $\frac{n}{k} = 2$. Subsequently, $\mathbf{z}_2$ is encoded using $\mathbf{G}_S$ to obtain the codeword

$$\mathbf{c}_2 = \mathbf{G}_S\mathbf{z}_2 = \begin{bmatrix} \mathbf{z}_2 \\ \mathbf{B}\mathbf{z}_2 \end{bmatrix} \in \mathbb{F}_q^6.$$

For this scheme too, the number of I/O reads needed to retrieve the first 2 versions is 5.

Since the code is MDS, probability of losing $\mathbf{x}_1$ is given by

$$\text{Prob}_S(\mathcal{E}_1) = p^6 + C_5^6 p^5(1-p) + C_4^6 p^4(1-p)^2. \quad (19)$$

The difference object $\mathbf{z}_2$ is lost if 5 or more nodes fail. In addition, since not all 2×3 submatrices of $\mathbf{G}_s$ satisfy Criterion 2, there are some specific 4 node failure patterns that lead to the loss of the object. Considering all possibilities, we have

$$\mathcal{E}_2 = \{\text{5 or more nodes fail}\} \cup \{\text{specific 4 nodes failure}\}.$$

Hence, the probability of losing $\mathbf{z}_2$ is given by

$$\begin{aligned} \text{Prob}_S(\mathcal{E}_2) &= p^6 + C_5^6 p^5(1-p) + 12p^4(1-p)^2, \quad (20)\\ &< \text{Prob}_S(\mathcal{E}_1). \end{aligned}$$

In Fig. 2, we compare $\text{Prob}_S(\mathcal{E}_2)$ (for systematic SEC) and $\text{Prob}_N(\mathcal{E}_2)$ (for non-systematic SEC) for different values of p. The plots show that systematic SEC, while exploiting the sparsity to reduce the I/O reads, does not provide higher protection for the difference object $\mathbf{z}_2$ when compared to the non-systematic SEC. Nevertheless, in colocated placement, the resilience for both the objects $\mathbf{x}_1$ and $\mathbf{x}_2$ is dominated by that for $\mathbf{x}_1$. Hence, we have

$$P_c(\mathbf{x}_1, \mathbf{x}_2) = 1 - \left(p^6 + C_5^6 p^5(1-p) + C_4^6 p^4(1-p)^2\right).$$

From the point of view of failure events for $\mathbf{z}_2$, there are a total of 63 possible failure patterns of nodes. Among them, both non-systematic and systematic methods can recover from 41 patterns due to the inherent MDS property, wherein, sparsity is not exploited and $\mathbf{z}_2$ is retrieved with 3 I/O reads. In

TABLE I

DIFFERENTIAL VS. NON-DIFFERENTIAL ERASURE CODING (NUMBERS ARE BASED ON EXAMPLE OF SUBSECTION IV-C)

Version	Parameter	Differential Non-systematic	Differential Systematic	Non-differential Systematic
1st	Encoding	$\mathbf{c}_1 = \mathbf{G}_N\mathbf{x}_1$	$\mathbf{c}_1 = \mathbf{G}_S\mathbf{x}_1$	$\mathbf{c}_1 = \mathbf{G}_S\mathbf{x}_1$
	Encoding Complexity	matrix multiplication	matrix multiplication for parity only	matrix multiplication for parity only
	Nr. of nodes	6	6	6
	Decoding Complexity	inverse operation	low	low
	I/O reads	3	3	3
2nd	Encoding	$\mathbf{c}_2 = \mathbf{G}_N\mathbf{z}_2$	$\mathbf{c}_2 = \mathbf{G}_S\mathbf{z}_2$	$\mathbf{c}_2 = \mathbf{G}_S\mathbf{x}_2$
	Encoding Complexity	matrix multiplication	matrix multiplication for parity only	matrix multiplication for parity only
	Nr. of nodes	6	6	6
	Decoding Complexity	sparse reconstruction	sparse reconstruction	low
	I/O reads	2	2	3

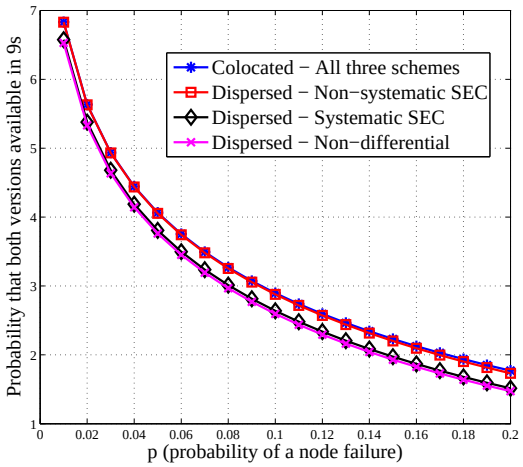

Fig. 3. Resilience of colocated and distributed placement strategies for the example of Subsection IV-C. On the y-axis, the probability of joint availability of $\mathbf{x}_1$ and $\mathbf{x}_2$ in 9s format is shown, defined as $-\log_{10}(1-p_d(\mathbf{x}_1,\mathbf{x}_2))$ resp. as $-\log_{10}(1-p_c(\mathbf{x}_1,\mathbf{x}_2))$ for dispersed, resp. colocated placement.

addition to these failures, the non-systematic code can resist 15 more failure patterns arising from failure of all possible 4 node combinations. However, for such a case, the systematic SEC can resist only in additional 3 cases. Therefore, non-systematic SEC can handle a total of 56 failure patterns, while systematic can handle only 44 patterns. Thus, non-systematic SEC can opportunistically improve the resilience for the difference object $\mathbf{z}_2$ compared to systematic SEC.

In Fig. 3 we show the probability of availability of all the versions (i.e., both $\mathbf{x}_1$ and $\mathbf{x}_2$ in Example of Subsection IV-C) of the data objects for both dispersed (in (11)) and colocated placements (in (12)) for a range of values for the probability of failure p of individual storage nodes. For colocated placement, all the three schemes have the same resilience to store $\mathbf{x}_1$ and $\mathbf{x}_2$. However, for dispersed placement non-systematic SEC provides higher resilience than the other two schemes.

D. Resilience analysis summary

A comprehensive summary of the resilience of the three schemes is provided in Table I. The highlights are as follows:

(1) For any given choice of coding scheme, colocated placement of encoded pieces of multiple versions of a data object results in higher resilience than the dispersed placement of these encoded pieces. However, going into more subtlety, there is one advantage of using non-differential coding with dispersed storage, namely, some random versions (but not the whole versioned archive) will survive with a greater probability than the probability of survival of the whole archive for the colocated case. In contrast, since the basic SEC stores differences, this is not the case. The optimized SEC however benefits from this serendipity as well, but less often than for the non-differential case.

(2) For colocated placement, the non-systematic SEC provides the same resilience to retrieve the whole archive as that of the systematic SEC. However, larger number of options to recover the sparse updates for the former method results in higher resilience for storing the individual difference objects than the latter method. In addition, whenever $\frac{k}{n} > \frac{1}{2}$, non-systematic SEC works for a larger range of sparseness levels with reduced I/O than the systematic SEC.

(3) For colocated placement, the non-systematic SEC provides the same resilience as that of the non-differential method, but with the advantage of requiring fewer I/O reads than the latter when retrieving the whole versioned archive.

Finally, we note that for the non-systematic SEC, the individual version deltas have higher static resilience (Fig 2), however, there is no advantage in this, given that the actual retrievability is bottlenecked by the facts that (i) colocation is best strategy, and (2) the availability of the first version (where the whole object is coded) thus dominates and determines retrievability of the whole archive. This suggests that the additional resilience of individual deltas in the non-systematic SEC is wasteful in terms of storage resources, and that there is potential room to reduce storage overhead while storing with non-systematic SEC. Study of storage optimization for

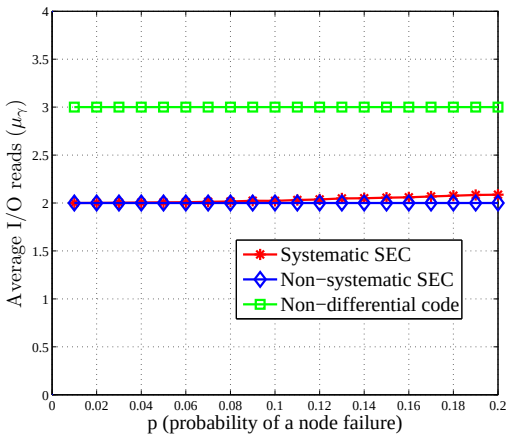

Fig. 4. Average I/O reads μ_γ given in (21) for $\gamma = 1$ to retrieve the 1-sparse object $\mathbf{z}_2$ for different erasure codes. These results are obtained for the parameters $n = 6$ and $k = 3$ in Section IV-C.

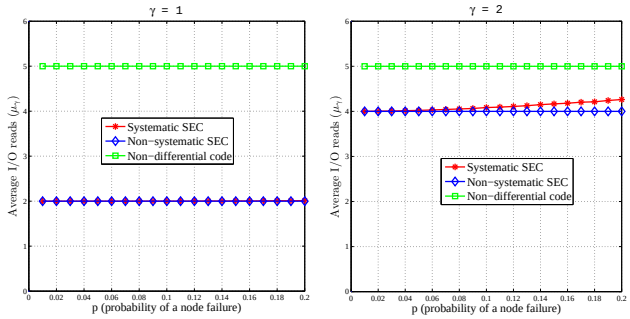

Fig. 5. Average I/O reads given in (21) to retrieve the sparse object $\mathbf{z}_2$ for different erasure codes. The results are obtained for the parameters $n = 10$ and $k = 5$. The plots on the left and right correspond to retrieving $\mathbf{z}_2$ which is 1-sparse and 2-sparse, respectively.

the non-systematic SEC will be part of our future work.

V. Versioned Archive Retrieval I/O

In the previous section, we studied the static resilience for the various schemes - and demonstrated that, for a given storage overhead, all the strategies achieve the same effective fault-tolerance for the colocated placement scenario, which is the best case, and hence practical. We next focus on the disk I/Os involved while retrieving a versioned archive. The actual savings when using SEC depend on the actual sparsity of the differences across versions, and hence we obtain these results numerically for different example workloads.

A. Non-systematic and Systematic SEC

We consider the object from Section IV-C where we chose a specific case of $\mathbf{z}_2$ being 1-sparse. However, in general, the sparsity level of $\mathbf{z}_2$ is a random variable over the support $\{1, 2, 3\}$. Since $k = 3$, the sparsity can be exploited only when $\gamma_2 = 1$. Henceforth, we denote γ_2 by γ. We now discuss the number of options for the systematic and non-systematic codes to retrieve the 1-sparse object with just 2 I/O reads. For the non-systematic code, since any 2×3 submatrix of $\mathbf{G}_N$ satisfies the Criterion 2, there are a total of 15 such matrices. However, for the systematic code, only 3 submatrices of $\mathbf{G}_S$ satisfy Criterion 2.

We next discuss the implications of having reduced number of submatrices satisfying Criterion 2 on the (average) I/O for retrieving the γ-sparse object $\mathbf{z}_2$. The following approach is needed only for systematic SEC, since for the non-systematic SEC, every pattern of k or more live nodes has a subset of size 2γ that can recover $\mathbf{z}_2$, and hence 2γ I/O reads are guaranteed. We randomly generate a large ensemble of failure patterns of nodes by assuming that each node fails independently with probability p. We traverse through every failure pattern to identify if at least k live nodes remain, in order to recover the object. If k or more nodes are alive, we find the possibility of retrieving $\mathbf{z}_2$ with just 2γ I/O reads by checking Criterion 2 on the submatrices corresponding to live nodes. Accordingly, we retrieve the entire object with just 2γ I/O reads, otherwise, it is recovered using k reads. Provided that a failure pattern leaves k or more live nodes, we compute the percentage of cases when only $2\gamma_2$ reads are sufficient (denoted by $p_{2\gamma}$) and that of when k I/O reads are needed (denoted by p_k). Then the average number of I/O reads is computed as

$$\mu_\gamma = p_{2\gamma} 2\gamma + p_k k. \tag{21}$$

For the parameters in Section IV-C, we have $\mu_1 = 2p_2 + 3p_3$. In Fig. 4, we plot μ_1 for different values of p from 0.01 to 0.2. The plot shows that when p is small, systematic SEC recovers $\mathbf{z}_2$ with just 2 I/O reads. However, as p increases, a non-negligible number of error patterns occurs for which no subset of live nodes with cardinality 2 can recover $\mathbf{z}_2$. The two other schemes are also shown: (i) the lower one (with constant reads of 2) corresponds to the non-systematic SEC as every pattern of k or more live nodes has a subset of size 2 that can recover $\mathbf{z}_2$, and (ii) the top one (with constant reads of 3) corresponds to non-differential encoding where sparsity cannot be exploited.

A similar experiment is repeated with parameters $n = 10, k = 5$ and $L = 2$. The first version $\mathbf{x}_1$ is fully encoded and the second version is encoded with the SEC schemes. Since $k = 5$, we apply the average I/O reads study to $\gamma = 1$ and 2. Provided that failure patterns are such that k or more nodes are alive, the average number of I/O reads μ_γ (given in (21)) to retrieve $\mathbf{z}_2$ are provided in Fig. 5 for (i) $\gamma = 1$ and (ii) $\gamma = 2$. The plots show that for $\gamma = 1$, systematic SEC retrieves $\mathbf{z}_2$ using 2 I/O reads almost always for values of p till $p = 0.2$. However, for $\gamma = 2$, there is marginal increase in the values of μ_γ for higher values of p till $p = 0.2$.

In conclusion (i) both variants of SEC outperform the naive solution of encoding individual versions, and (ii) while non-systematic SEC consistently performs better than systematic SEC, the differences are marginal for practical settings (where p) is not very high.

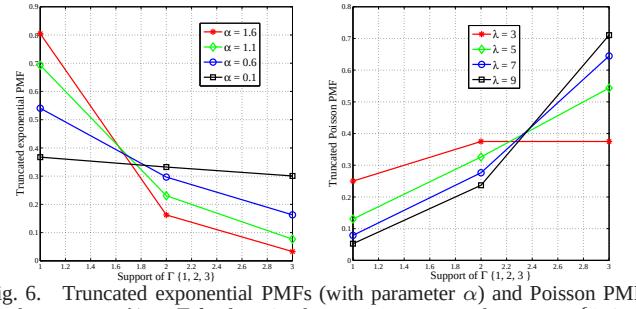

Fig. 6. Truncated exponential PMFs (with parameter α) and Poisson PMFs (with parameter λ) on Γ for $k = 3$. The x-axis represents the support $\{1, 2, 3\}$ of Γ.

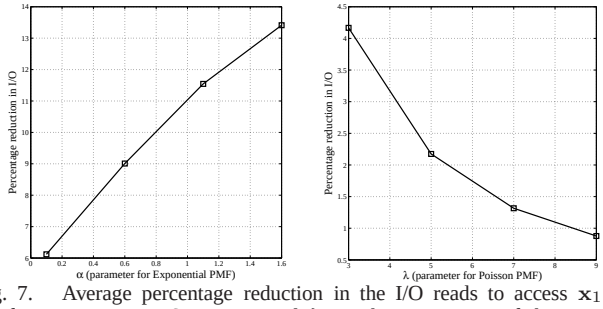

Fig. 7. Average percentage reduction in the I/O reads to access $\mathbf{x}_1$ and $\mathbf{x}_2$ for PMFs in Fig. 6. Here α and λ are the parameters of the truncated Exponential PMF and truncated Poisson PMFs in (22) and (23), respectively. The results are for $n = 6$ and $k = 3$.

B. Expected I/O savings with two versions

Previously, we saw in Section III that for the example from Section IV-C, SEC reduces the I/O reads for joint retrieval of $\mathbf{x}_1$ and $\mathbf{x}_2$ from 6 to 5 when $\mathbf{z}_2$ is 1-sparse. We further studied settings with fixed γ values in Section V-A above. However, in general, the sparsity level of $\mathbf{z}_2$ can take any value over $\{1, 2, 3\}$. We now present numerical results on the expected number of I/O reads when γ_2 is random (denoted by the random variable Γ_2).

For the SEC schemes, the number of I/O reads to access both the versions are $5, 6$, and 6 when $\mathbf{z}_2$ is 1-, 2- and 3-sparse, respectively. Since the average number of I/O reads depends on the underlying probability mass function (PMF) on the sparsity level, we study the advantages of the proposed method by testing different PMFs that reflect different difference sparsity behaviors, in the absence of standard workloads (see e.g. [10]) . Henceforth, we use Γ to denote the random variable Γ_2 and γ to denote its realization γ_2. The PMF on Γ is denoted by $\mathrm{P}_\Gamma(\gamma)$ for $\gamma \in \{1, 2, 3\}$.

PMFs on sparsity. We apply the finite support versions of the exponential distribution in parameter $\alpha > 0$ given by

$$\mathrm{P}_\Gamma(\gamma) = ce^{-\alpha\gamma}, \text{ for } \gamma = 1, 2, 3, \tag{22}$$

where the constant c is chosen such that $\sum_{\gamma=1}^{k} \mathrm{P}_\Gamma(\gamma) = 1$, and referred to as *truncated exponential* PMF. Likewise, a *truncated Poisson* PMF with parameter λ, given by

$$\mathrm{P}_\Gamma(\gamma) = c\frac{\lambda^\gamma e^{-\lambda}}{\gamma!}, \text{ for } \gamma = 1, 2, 3, \tag{23}$$

are also considered, where c is such that $\sum_{\gamma=1}^{k} \mathrm{P}_\Gamma(\gamma) = 1$. These PMFs are specifically picked to study the reduction in I/O reads for two extreme scenarios: (i) the family of exponential PMFs provides thick concentration towards smaller value of Γ, whereas (ii) the family of Poisson PMFs provides thick concentration towards larger value of Γ. Thus they facilitate the study of both the best-case and worst-case scenarios for SEC. In Figures 6, we plot the PMFs in (22) and (23), respectively for different parameters.

Average I/O reads to retrieve $\mathbf{x}_1$ and $\mathbf{x}_2$. For a given $\mathrm{P}_\Gamma(\gamma)$, the average number of I/O reads for accessing the first two versions are given by

$$\mathbb{E}[\eta] = k + \sum_{\gamma=1}^{k} \mathrm{P}_\Gamma(\gamma)\min(2\gamma, k).$$

In Figure 7, we plot the average percentage reduction in the I/O reads when compared to the non-differential setup as $\frac{2k-\mathbb{E}[\eta]}{2k} \times 100$ where $2k$ is the total number of I/O reads for the non-differential scheme. The plots show a significant reduction in the I/O reads when the distribution is skewed towards smaller γ. However, as expected, the reduction is marginal otherwise.

Average I/O reads to retrieve $\mathbf{x}_2$ alone. The average number of I/O reads to retrieve the 2nd version alone using the basic SEC is $\mathbb{E}[\eta(\mathbf{x}_2)] = \mathbb{E}[\eta(\mathbf{x}_1, \mathbf{x}_2)]$ since the delta has to applied over the first version. However, for the optimized method, the average I/O reads is $\mathbb{E}[\eta(\mathbf{x}_2)] = \sum_{\gamma=1}^{k} \mathrm{P}_\Gamma(\gamma)t(\gamma)$ where $t(\gamma) = k$ when $\gamma \geq \frac{k}{2}$, and $t(\gamma) = k + 2\gamma$, otherwise. Compared to non-differential coding, the average percentage increase in the I/O reads for fetching the 2nd version for both the basic and the optimized methods is computed as $\frac{\mathbb{E}[\eta(\mathbf{x}_2)]-k}{k} \times 100$. In Fig. 8 we present the results corresponding to the PMFs in Fig. 6. It shows that the optimized SEC reduces the excess number of I/O reads for the 2nd version. Though the optimized SEC reduces the excess I/O, this additional I/O reads for $\mathbf{x}_2$ is due to differential encoding that reduces the I/O for accessing both $\mathbf{x}_1$ and $\mathbf{x}_2$. One possible direction to reduce the I/O for the latest version is to employ reverse SEC (as pointed in Section III-A).

C. An example system with $L > 2$ versions

To study the trade-offs of using SEC when multiple versions are involved, we revisit the example in Section III-D. $L = 5$ versions with sparsity levels of subsequent versions $\{\gamma_j \mid 2 \leq j \leq L\} = \{3, 8, 3, 6\}$ of an object of size $k = 10$ is stored using a $(20, 10)$ SEC. We plot the I/O numbers for the basic and the optimized SEC in Fig. 9. The numbers are presented to retrieve both the individual versions (l-th version for $1 \leq l \leq 5$) as well as all the first l versions. The plot shows 20% saving in total I/O reads with respect to the non-differential scheme, for only slightly higher I/O for the optimized DEC.

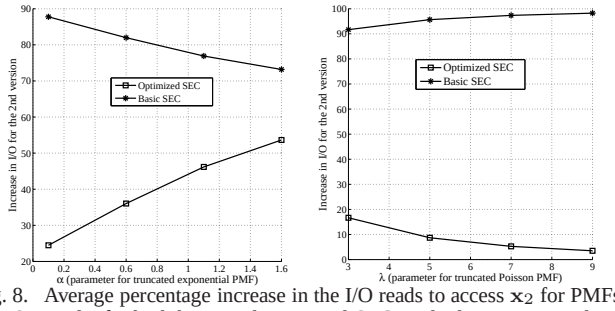

Fig. 8. Average percentage increase in the I/O reads to access $\mathbf{x}_2$ for PMFs in Fig. 6. Results for both basic and optimized SEC methods are presented. Here α and λ are the parameters of the truncated Exponential PMF and truncated Poisson PMFs in (22) and (23), respectively.

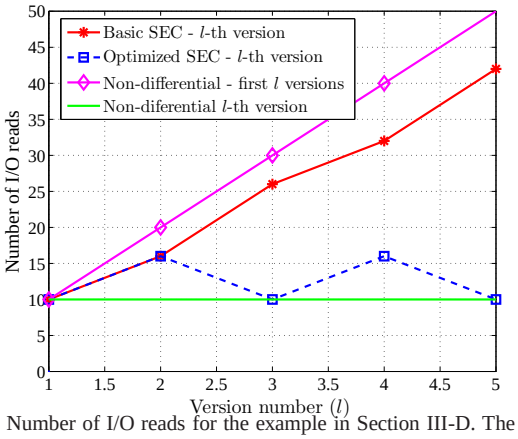

Fig. 9. Number of I/O reads for the example in Section III-D. The I/O reads to retrieve the l-th version and the first l-versions, $1 \leq l \leq 5$ are presented for different methods.

VI. Conclusions

In this paper we propose a framework - Sparsity Exploiting Coding (SEC) - for archiving versioned data using storage efficient erasure coding, where the individual versions are not coded in isolation, but instead the differences across subsequent versions are coded. The sparsity in the delta information is exploited for better I/O performance when the archive of versioned data is read back. We identify Cauchy matrix based MDS codes as one candidate which satisfies the requirements laid out in our framework to be able to opportunistically exploit the sparsity, and discuss two variants, a systematic and a non-systematic one. We demonstrate that, in doing so, for a given choice of storage overhead, and for a practical redundancy placement strategy (where encoded pieces pertaining to all the versions are colocated) there is no compromise in the system's resilience. Analysis and numerical/simulation experiments confirm the effectiveness of SEC.

Our study shows that the non-systematic SEC provides better resilience for deltas corresponding to individual intermediate versions, even though the whole archive's resilience is constricted by the resilience of the first (or last) version of the data, which is coded as it is. This suggests room for reducing the storage overhead of the intermediate versions in the non-systematic SEC, possibly using puncturing techniques, which we will pursue in immediate future.

References

[1] K. S. Esmaili, A. Chiniah, and A. Datta. Efficient updates in cross-object erasure-coded storage systems. In *IEEE International Conference on Big Data*, 2013.

[2] D. Ford, F. Labelle, F. I. Popovici, M. Stokely, V.-A. Truong, L. Barroso, C. Grimes, and S. Quinlan. Availability in globally distributed storage systems. In *The 9th USENIX conference on Operating Systems Designand Implementation (OSDI)*, 2010.

[3] S. Han, H.-T. Pai, R. Zheng, and P. K. Varshney. Update-efficient regenerating codes with minimum per-node storage. In *Proceedings of the Int. Symp. Inf. Theory*, 2013.

[4] J. Lacan and J. Fimes. A construction of matrices with no singular square submatrices. In *International Conference on Finite Fields and Applications*, 2003.

[5] A. Mazumdar, G. W. Wornell, and V. Chandar. Update efficient codes for error correction. In *Proceedings of the Int. Symp. Inf. Theory*, 2012.

[6] F. Oggier and A. Datta. *Coding Techniques for Repairability in Networked Distributed Storage Systems*. Foundations and Trends in Communications and Information Theory, Now Publishers, 2013.

[7] A. Rawat, S. Vishwanath, A. Bhowmick, and E. Soljanin. Update efficient codes for distributed storage. In *Proceedings of the Int. Symp. Inf. Theory*, 2011.

[8] "SVN". `http://subversion.apache.org/`.

[9] A. Thusoo, Z. Shao, S. Anthony, D. Borthakur, N. Jain, J. S. Sarma, R. Murthy, and H. Liu. Data warehousing and analytics infrastructure at facebook. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data, ser. SIGMOD 10*, 2010.

[10] V.Tarasov, A. Mudrankit, W. Buik, P. Shilane, G. Kuenning, and E. Zadok. Generating realistic datasets for deduplication analysis. In *Proceedings of the 2012 USENIX conference on Annual Technical Conference*, 2012.

[11] F. Zhang and H. D. Pfister. Compressed sensing and linear codes over real numbers. In *Information Theory and Applications Workshop (ITA)*, 2008.