Synchronizing Edits in Distributed Storage Networks

Salim El Rouayheb*, Sreechakra Goparaju†, Han Mao Kiah‡, Olgica Milenkovic§
*Department of Electrical and Computer Engineering, Illinois Institute of Technology, USA
†Calit2, University of California, San Diego, USA
‡School of Physical and Mathematical Sciences, Nanyang Technological University, Singapore
§Coordinated Science Lab, University of Illinois at Urbana-Champaign, USA

***Abstract*—We consider the problem of synchronizing data in distributed storage networks under edits that include deletions and insertions. We present modifications of codes on distributed storage systems that allow updates in the parity-check values to be performed with one round of communication at low bit rates and a small storage overhead. Our main contributions are novel protocols for synchronizing both frequently updated and semi-static data, and protocols for data deduplication applications, based on intermediary coding using permutation and Vandermonde matrices.**

***Index Terms*—Distributed storage, Synchronization.**

I. Introduction

Coding for distributed storage systems (DSSs) has garnered significant attention in the past few years [1]–[5], due to the rapid development of information technologies and the emergence of Big Data formats that need to be saved or disseminated in a distributed fashion across a network. Two key functionalities of codes for DSSs are (i) *reconstruction* of the stored files via access to a subset of the nodes to achieve reliability, and (ii) content *repair* of failed nodes. Both functionalities need to be retained when the files undergo *edits*, such as symbol insertions, deletions, or substitutions. Edits frequently arise in storage and networked systems involving non-archival data that is frequently edited or data deduplication[1] features [6], or in shared file systems such as Dropbox and Sugarsync [7]. Current solutions for synchronization protocols operate exclusively on uncoded data (examples include rsync [8], dsync [9], and a number of related file synchronization methods put forward in the information theory literature [10]–[14]); furthermore, they do not fully exploit the distributed nature of information. On the other hand, deduplication methods mostly apply to read-only architectures and are in an early stage of development for distributed systems [15].

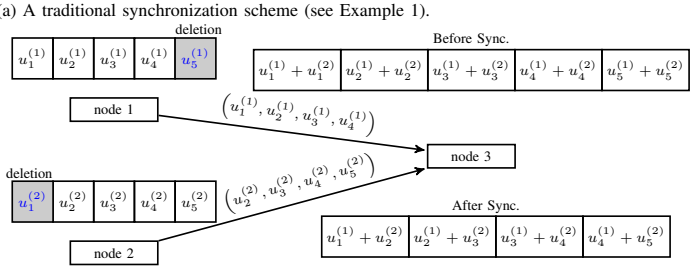
It is in general not an easy task to synchronize or deduplicate a file and its corresponding DSS nodes while simultaneously optimizing the communication/updating costs and retaining the reconstruction and repair functionalities. In the *uncoded* domain, the synchronization problem involves a single user and a single node storing a replica of her/his file. Once the user edits the file, assuming no knowledge of the edits, the user and the storage node communicate interactively until their files are matched (or until the node matches the master copy of the user).

In the *encoded* DSS scenario that we propose to analyze, we make the natural assumption that the users have *full knowledge* of the edits that they make. Furthermore, our work is primarily concerned with efficiently updating encoded copies in the DSS nodes with minimal communication rates under edits that are insertions or deletions. This fundamentally differs from update efficient codes studied in [16], [17], where the edits can be viewed as substitutions.

This work was supported in part by NSF Grants CIF 1218764, CIF 1117980, and STC Class 2010, CCF 0939370, and the Strategic Research Program of University of Illinois, Urbana-Champaign. This work was completed when the third author was at University of Illinois at Urbana-Champaign.

[1]Deduplication is the process of identifying duplicate files or data chunks in the system and deleting the duplicates and replacing them with a reference to the original data.

(a) A traditional synchronization scheme (see Example 1).



(b) Our synchronization scheme (see Example 3).

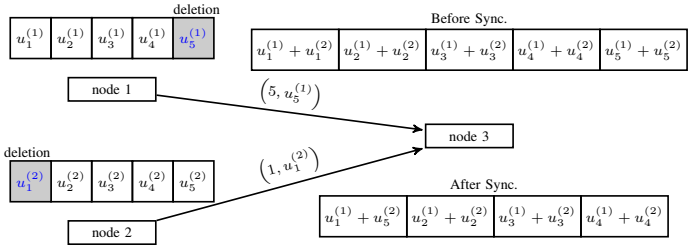


Fig. 1. Synchronization schemes for coded data. Communication cost is reduced from $8\log_2 q$ in (a) to $2\log_2 q + 2\log_2 5$ bits in (b). Here q is the alphabet size.

Contributions and Outline: We describe coding protocols for efficient data synchronization in a distributed storage environment that dynamically maintain regenerative properties. The synchronization protocols are based on a simple new scheme termed *intermediary coding*, which changes the structure of the code during each update. The intermediary preprocessing scheme also offers flexibility in terms of accommodating a very broad family of coding schemes used in DSSs (such as erasure codes, regenerating codes, and locally repairable codes). In addition, we show that traditional synchronization schemes require significantly higher communication costs and describe an application of our method for an important class of deduplication algorithm called post-processing deduplication.

The paper is organized as follows. Section II motivates the problem using the example in Fig. 1 and provides the precise problem statement. Section III examines the underlying communication costs when traditional DSS encoding methods are used. Section IV contains our main result, a collection of encoding algorithms and protocols for data synchronization that have communication cost a constant factor away from the fundamental limits. The full version of the paper [18] contains proofs to propositions, extensions of the proposed schemes, and derivations of fundamental performance limits.

II. Motivation and Problem Statement

We start with a straightforward example to motivate the difficulties encountered in synchronizing coded data.

Example 1. Consider two users with data blocks $\mathbf{u}^{(1)}$ and $\mathbf{u}^{(2)}$, each consisting of five symbols of length 5 over $\mathbb{F}_q$. Suppose these blocks are encoded in a DSS consisting of three nodes, that store $\mathbf{u}^{(1)}$, $\mathbf{u}^{(2)}$, and $\mathbf{u}^{(1)}$ + $\mathbf{u}^{(2)}$, respectively. This DSS coding scheme, illustrated in Fig. 1, satisfies the *reconstruction property* for the data blocks $\{\mathbf{u}^{(1)},\mathbf{u}^{(2)}\}$, and it may be used to repair any one failed node by accessing the remaining two nodes. Suppose next that the last symbol in $\mathbf{u}^{(1)}$ and the first symbol in $\mathbf{u}^{(2)}$ are deleted, resulting in the smaller data blocks $\tilde{\mathbf{u}}^{(1)} = (u_1^{(1)}, u_2^{(1)}, u_3^{(1)}, u_4^{(1)})$ and $\tilde{\mathbf{u}}^{(2)} = (u_2^{(2)}, u_3^{(2)}, u_4^{(2)}, u_5^{(2)})$. The question of interest may be stated as follows: What communication protocol should the users and the DSS nodes employ to minimize the data transmission cost whilst retaining the reconstruction and repair functionalities?

One way would be for the three DSS nodes to update their contents to $\tilde{\mathbf{u}}^{(1)}$, $\tilde{\mathbf{u}}^{(2)}$, and $\tilde{\mathbf{u}}^{(1)} + \tilde{\mathbf{u}}^{(2)}$, respectively. For the uncoded parts of the encoded DSS nodes, it is both necessary and sufficient for the user with data block $\mathbf{u}^{(i)}$ to communicate her/his deletion position (that is, $\log_2 5$ bits in Example 1) to the corresponding node i. The next proposition states that at least four symbols need to be transmitted to the parity check node 3.

Proposition 1. *Let $\mathbf{u}^{(1)}$ and $\mathbf{u}^{(2)}$ be the user data blocks of length ℓ. Assume that exactly one deletion has occurred in $\mathbf{u}^{(1)}$ and $\mathbf{u}^{(2)}$. Let $\tilde{\mathbf{u}}^{(1)}$ and $\tilde{\mathbf{u}}^{(2)}$ denote the respective edited blocks, and suppose that the information to be updated at the three nodes of the corresponding encoded DSS is given by $\tilde{\mathbf{u}}^{(1)}$, $\tilde{\mathbf{u}}^{(2)}$, and $\tilde{\mathbf{u}}^{(1)} + \tilde{\mathbf{u}}^{(2)}$, respectively. Then, in the worst case over all possible edit locations, the total communication cost is at least $\ell - 1$ symbols, or $(\ell-1)\log_2 q$ bits, independent of the network topology between the users and the nodes.*

It therefore appears that in Example 1, both users have to necessarily transmit almost their whole contents to node 3. We show however that, akin to the notion of functional repair in DSS [2], one may significantly save in communication complexity by updating the contents of node 3 via a flexible change in the code structure. The key idea is to use the fact that the users have full knowledge of the edits and "transfer" the burden of updating the data from the DSS nodes to the users. We first formally describe the general problem setup and then proceed to describe our underlying solution.

A. Coding for Distributed Storage Systems

Notation: We denote by $[n]$ the set of integers $\{1, 2, \ldots, n\}$ and by $\binom{X}{k}$ the collection of k-subsets of X for a set X and $k \leq |X|$. Given a finite field $\mathbb{F}_q$ of order q, the vector space of all vectors of length ℓ, the vector space of all matrices of dimensions $m \times n$, and the vector space of all tensors of dimensions $m \times n \times \ell$ are denoted by $\mathbb{F}_q^{\ell}$, $\mathbb{F}_q^{m\times n}$ and $\mathbb{F}_q^{m\times n\times \ell}$, respectively. For $i \in [\ell]$, x_i represents the i-th coordinate of a vector $\mathbf{x} \in \mathbb{F}_q^{\ell}$, and $\mathbf{e}_i$ the i-th standard basis vector. Given a matrix $\mathbf{M} \in \mathbb{F}_q^{m\times n}$, a subset of the rows $R \subseteq [m]$, and a subset of the columns $C \subseteq [n]$, $\mathbf{M}|_{R\times C}$ represents the $|R| \times |C|$ matrix obtained by restricting $\mathbf{M}$ to rows in R and columns in C. We use analogous definitions for tensors.

Let $\mathbf{x} = \left(x^{(1)}, x^{(2)}, \ldots, x^{(B)}\right) \in \mathbb{F}_q^B$ be an information vector to be stored in a distributed storage system (DSS). We call each $x^{(s)}$, for $s = 1, \ldots, B$, a *data unit*. A DSS is equipped with an *encoding* function, as well as a set of reconstruction and a set of repair algorithms specifying a coding scheme. The encoding algorithm converts the vector $\mathbf{x}$ into n vectors of length α and stores them in n *storage nodes*, while the reconstruction algorithm recovers $\mathbf{x}$ from the contents of any $k \leq n$ out of n nodes. In addition, when a node fails, one uses the contents of a subset of $d \leq n$ nodes to repair the contents of the failed node. More formally, we have the following definitions.

1) An *encoding* function is a map $\textsc{Ec} : \mathbb{F}_q^B \to \mathbb{F}_q^{n\times\alpha}$, where α is chosen such that $\textsc{Ec}$ can be constructed.
2) For any $T \in \binom{[n]}{k}$, a map $\textsc{Rc}_T : \mathbb{F}_q^{k\times\alpha} \to \mathbb{F}_q^B$ is termed a *reconstruction* function if for $\mathbf{x} \in \mathbb{F}_q^B$,
$$\textsc{Rc}_T\left(\textsc{Ec}\,(\mathbf{x})|_{T\times[\alpha]}\right) = \mathbf{x}.$$
3) Given a $t \in [n]$ and $T \in \binom{[n]\setminus\{t\}}{d}$, a map $\textsc{Rp}_{t,T} : \mathbb{F}_q^{d\times\alpha} \to \mathbb{F}_q^{\alpha}$ is termed an *exact repair* function if
$$\textsc{Rp}_{t,T}(\mathbf{C}|_{T\times[\alpha]}) = \mathbf{C}|_{\{t\}\times[\alpha]},$$
where $\mathbf{C}$ is the information stored over n nodes.

We refer to an encoding for a DSS as an (n, k, d, α, B) DSS code, and focus on codes where the above mappings are linear.

B. Problem Description

Our edit model assumes that there is a user $s \in [B]$ corresponding to each data unit. In general, each user is associated with a *data block* consisting of ℓ data units. The overall $B\ell$ data units are stored by extending the (n, k, d, α, B) DSS code to an $(n, k, d, \ell\alpha, \ell B)$ DSS code. This extension is akin to data-striping in RAID systems, amounting to a simple conversion of a symbol to a vector of symbols. One can also view this construction as a means of dividing the $B\ell$ data units into ℓ groups of B symbols each, and then applying to each group an (n, k, d, α, B) DSS encoding. We say that a set of nodes $N(s)$ is *connected to* user s if the nodes need to be updated when data unit s is edited. That is, $N(s) \triangleq \{t \in [n] : \textsc{Ec}(\mathbf{e}_s)|_{\{t\}\times[\alpha]} \neq \mathbf{0}\}$.

We assume that the data blocks are subjected to deletions performed in an independent fashion by B *different* users[2]. Furthermore, we focus on a *uniform edits* model, in which each data block has the same number of deletions and thus the resulting data blocks all have the same length.

Example 2 (Example 1 Continued). Consider a $[3, 2]$ MDS code, or a $(3, 2, 2, 5, 10)$ DSS code. Note that the DSS stores data blocks $\mathbf{u}^{(1)}$ and $\mathbf{u}^{(2)}$ of length $\ell = 5$, that are assumed to be edited by users 1 and 2. Nodes 1 and 3 are connected to user 1, and nodes 2 and 3 are connected to user 2. □

The problem of interest may be stated as follows:

Find the smallest communication cost protocol for the B users to communicate their edits to the connected DSS nodes, so that the nodes can update their information while maintaining the reconstruction and repair functionalities.

For simplicity and concreteness, we define cost as the number of bits transmitted from a user to a DSS node *connected* to it, *averaged* over all users. We are typically concerned with the worst case cost, that is cost *maximized* over all edit scenarios (say, all single deletion scenarios). For example, in the worst case,

[2]It may be possible that a user edits a number of different data blocks. However, for simplicity, our model assumes that each data block is edited by one user.

$\ell \log_2 q \geq \mathsf{cost} \geq \log_2 \ell$ for a deletion, and $\ell \log_2 q \geq \mathsf{cost} \geq \log_2 \ell + \log_2 q$ for an insertion. A scheme that achieves these bounds is for each user to send the entire edited data file to each DSS node. The lower bound follows by noting that at least one user needs to communicate the edit position (and inserted symbol in the case of an insertion) to at least one DSS node.

In what follows, we introduce what we call the traditional synchronization scheme and describe its shortcomings. We then propose two schemes that achieve a communication cost of $O(\log \ell + \log q)$ bits, i.e., of the order of the intuitive lower bound. To facilitate this low communication cost, we introduce additional storage overhead needed to describe the intermediary encoding function. The gist of the encoding method is to transform the information, and hence the codes applied to data blocks, via permutation and Vandermonde matrix multiplication. The resulting schemes are subsequently referred to as Schemes P and V, respectively. The key property of the transforms is that they reduce the update and synchronization communication cost by changing the code structure. Detailed descriptions and analysis of this storage overhead may be found in our companion paper [18].

III. Synchronization with Traditional Encoding

We call a synchronization scheme *traditional* if for a given length ℓ of the data blocks, the corresponding $(n, k, d, \ell\alpha, \ell B)$ DSS code is a tensorized version of an (n, k, d, α, B) DSS code as described in Section II-B. In other words, for a fixed ℓ, we encode B data blocks of ℓ units each by encoding the i-th unit of each block using the same (n, k, d, α, B) DSS code. Formally, we have the *traditional* encoding function[3] $\mathrm{Ec}^{\ell} : \mathbb{F}_q^{B\times\ell} \to \mathbb{F}_q^{n\times\alpha\times\ell}$, defined such that for $i \in [\ell]$,

$$\mathrm{Ec}^{\ell}\left(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(B)}\right)\Big|_{[n]\times[\alpha]\times\{i\}} = \mathrm{Ec}\left(x_i^{(1)}, \ldots, x_i^{(B)}\right). \quad (1)$$

Hence, we regard the information stored at the n nodes as an $n \times \alpha \times \ell$ tensor. The corresponding reconstruction and repair functions can similarly be given by the functions $\mathrm{Rc}_T^{\ell} : \mathbb{F}_q^{k\times\alpha\times\ell} \to \mathbb{F}_q^{B\times\ell}$ for $T \in \binom{[n]}{k}$; and $\mathrm{Rp}_{t,T}^{\ell} : \mathbb{F}_q^{d\times\alpha\times\ell} \to \mathbb{F}_q^{\alpha\times\ell}$ for $t \in [n]$ and $T \in \binom{[n]\setminus\{t\}}{d}$, respectively, defined such that for $i \in [\ell]$,

$$\mathrm{Rc}_T^{\ell}\left(\mathbf{C}|_{T\times[\alpha]\times[\ell]}\right)\Big|_{[B]\times\{i\}} \triangleq \mathrm{Rc}_T\left(\mathbf{C}|_{T\times[\alpha]\times\{i\}}\right), \quad (2)$$

$$\mathrm{Rp}_{t,T}^{\ell}\left(\mathbf{C}|_{T\times[\alpha]\times[\ell]}\right)\Big|_{[\alpha]\times\{i\}} \triangleq \mathrm{Rp}_{t,T}\left(\mathbf{C}|_{T\times[\alpha]\times\{i\}}\right), \quad (3)$$

where $\mathbf{C}$, for brevity, is the encoded data $\mathrm{Ec}^{\ell}\left(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(B)}\right)$ stored in the DSS.

Suppose that following one edit per block, the data blocks are updated to $\tilde{\mathbf{x}}^{(1)}, \ldots, \tilde{\mathbf{x}}^{(B)}$, each of length $\ell - 1$. Then, the traditional synchronization scheme (henceforth, referred to as *Scheme T*) requires that the information stored at the n nodes be given by $\mathrm{Ec}^{\ell'}\left(\tilde{\mathbf{x}}^{(1)}, \ldots, \tilde{\mathbf{x}}^{(B)}\right)$. We can characterize the worst-case cost of scheme T according to the following proposition.

Proposition 2 (Scheme T)**.** *Consider an $(n, k, d, \ell\alpha, \ell B)$ DSS code and assume single deletions in the user data blocks. Scheme T updates the content of the storage nodes to $(n, k, d, (\ell - 1)\alpha, (\ell - 1)B)$ DSS codes, with each user sending out $\mathsf{cost} = |I| \log_2 q$ bits to a connected storage node. Here, $I = \{i \in [\ell] : i_{\min} \leq i \leq i_{\max} - 1\}$, where i_s is the deletion position for data block $s \in [B]$, $i_{\max} = \max_{s\in[B]} i_s$ and $i_{\min} = \min_{s\in[B]} i_s$. In the worst case, $\mathsf{cost} = (\ell-1)\log_2 q$ bits. Furthermore, no scheme can do better (up to a constant factor).*

Remark: The previous results easily extended to uniform deletion as well as insertion models.

IV. Synchronization with Order-Optimal Communication Cost

We present next two synchronization schemes that achieve a cost of $O(\log_2 \ell + \log_2 q)$ bits as opposed to $O(\ell \log_2 q)$ bits. Intuitively, the idea is illustrated in the continuation of Example 1 below.

Example 3 (Example 1 Continued). Suppose each user sends the following information: (deletion position, deleted symbol) to node 3, i.e., user 1 transmits $(5, u_5^{(1)})$ and user 2 transmits $(1, u_1^{(2)})$. See Fig. 1. Now, node 3 which has $\mathbf{u}^{(1)} + \mathbf{u}^{(2)}$ and receives $u_1^{(2)}, u_5^{(1)}$ can store $(u_1^{(1)}+u_5^{(2)}, u_2^{(1)}+u_2^{(2)}, u_3^{(1)}+u_3^{(2)}, u_4^{(1)}+u_4^{(2)})$. The new code continues to be a $[3, 2]$ MDS code with $\ell = 4$, albeit involving a different parity-check. The communication cost however is reduced from 8 bits (assuming a binary alphabet) to $2\log_2(5) + 2 = 6.6$ bits.

The idea is formalized by introducing the notion of intermediary encoding.

Let $\left(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(B)}\right) \in \mathbb{F}_q^{B\times\ell}$ and let $\mathbf{A}^{(1)}$, $\mathbf{A}^{(2)}$, …, $\mathbf{A}^{(B)}$ be invertible $\ell \times \ell$ matrices over $\mathbb{F}_q$. We define a new encoding function $\mathrm{Ec}^{*\ell} : \mathbb{F}_q^{B\times\ell} \to \mathbb{F}_q^{n\times\alpha\times\ell}$, which uses the *traditional* encoding function Ec^{ℓ} as a building block, according to

$$\mathrm{Ec}^{*\ell}\left(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(B)}\right) \triangleq \mathrm{Ec}^{\ell}\left(\mathbf{x}^{(1)}\mathbf{A}^{(1)}, \ldots, \mathbf{x}^{(B)}\mathbf{A}^{(B)}\right). \quad (1*)$$

The repair functions remain the same as in (3), where $\mathbf{C}$ now equals $\mathrm{Ec}^{*\ell}\left(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(B)}\right)$. The reconstruction functions in (2) give back the linearly transformed data blocks and since the corresponding matrices are invertible, the original data blocks are reconstructible. Observe that when $\mathbf{A}^{(s)} = \mathbf{I}$ for all $s \in [B]$, we recover scheme T.

We describe next how to choose the matrices $\mathbf{A}^{(s)}$ and the accompanying update protocols to ensure small values of the communication cost. Suppose once again that the data blocks are edited to length $\ell' = \ell - 1$. The idea is to request the users to modify their respective matrices $\mathbf{A}^{(s)}$ to invertible $\ell' \times \ell'$ matrices $\tilde{\mathbf{A}}^{(s)}$ according to the edits made. Then, users may transmit the *locations* and *values* of their edits rather than a whole span of values, with the storage nodes being able to update their respective information so that $(1*)$ holds. The matrices $\tilde{\mathbf{A}}^{(s)}$ are designed to be invertible, so the resulting system remains an $(n, k, d, \alpha\ell', B\ell')$ DSS code. We propose two different update schemes for the matrices $\tilde{\mathbf{A}}^{(s)}$ based on the frequency and extent to which edits are made. For simplicity, we describe the schemes for edits of the form of a single deletion or insertion; extensions to multiple edits are straightforward.

A. Semi-Static Data: Synchronization with Permutation Matrices

For *semi-static data*, we assume that only a *constant fraction* of the data blocks is edited by users so that most data blocks retain their original length ℓ. In this case, the matrices $\tilde{\mathbf{A}}^{(s)}$ – albeit modified – remain of dimension $\ell\times\ell$. The most appropriate

[3]For compactness, we write $\left(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(B)}\right)$ instead of its transpose.

TABLE I
EXAMPLE ILLUSTRATING SCHEME P WITH A $[3,2]$ MDS CODE OVER $\mathbb{F}_5$.

$\mathbf{u}^{(1)}$	$\mathbf{u}^{(2)}$	edit	$\mathbf{D}$	$\mathrm{Ec}^5\left(\mathbf{u}^{(1)}\mathbf{A}^{(1)}, \mathbf{u}^{(2)}\mathbf{A}^{(2)}\right)$	$\mathbf{A}^{(1)}$	$\mathbf{A}^{(2)}$
$(1,2,3,4,4)$	$(1,1,1,1,1)$	—	—	$\begin{pmatrix}1&2&3&4&4\\1&1&1&1&1\\2&3&4&0&0\end{pmatrix}$	$\begin{pmatrix}1&0&0&0&0\\0&1&0&0&0\\0&0&1&0&0\\0&0&0&1&0\\0&0&0&0&1\end{pmatrix}$	$\begin{pmatrix}1&0&0&0&0\\0&1&0&0&0\\0&0&1&0&0\\0&0&0&1&0\\0&0&0&0&1\end{pmatrix}$
$(1,3,4,4,0)$	$(1,1,1,1,1)$	deletion at position 2 of $\mathbf{u}^{(1)}$	$\begin{pmatrix}0&2&0&0&0\\0&0&0&0&0\\0&2&0&0&0\end{pmatrix}$	$\begin{pmatrix}1&0&3&4&4\\1&1&1&1&1\\2&1&4&0&0\end{pmatrix}$	$\begin{pmatrix}1&0&0&0&0\\0&0&1&0&0\\0&0&0&1&0\\0&0&0&0&1\\0&1&0&0&0\end{pmatrix}$	$\begin{pmatrix}1&0&0&0&0\\0&1&0&0&0\\0&0&1&0&0\\0&0&0&1&0\\0&0&0&0&1\end{pmatrix}$
$(1,3,4,1,4)$	$(1,1,1,1,1)$	insertion of 1 at position 4 of $\mathbf{u}^{(1)}$	$\begin{pmatrix}0&1&0&0&0\\0&0&0&0&0\\0&1&0&0&0\end{pmatrix}$	$\begin{pmatrix}1&1&3&4&4\\1&1&1&1&1\\2&2&4&0&0\end{pmatrix}$	$\begin{pmatrix}1&0&0&0&0\\0&0&1&0&0\\0&0&0&1&0\\0&1&0&0&0\\0&0&0&0&1\end{pmatrix}$	$\begin{pmatrix}1&0&0&0&0\\0&1&0&0&0\\0&0&1&0&0\\0&0&0&1&0\\0&0&0&0&1\end{pmatrix}$

TABLE II
EXAMPLE ILLUSTRATING SCHEME V WITH A $[3,2]$ MDS CODE OVER $\mathbb{F}_5$.

$\mathbf{u}^{(1)}$	i_1	$\mathbf{u}^{(2)}$	i_2	$\mathbf{D}$	$\mathrm{Ec}^\ell\left(\mathbf{u}^{(1)}\mathbf{A}^{(1)}, \mathbf{u}^{(2)}\mathbf{A}^{(2)}\right)$	$\mathbf{A}^{(1)}$	$\mathbf{A}^{(2)}$
$(0,1,0,1)$	—	$(1,0,1,0)$	—	—	$\begin{pmatrix}2&1&0&2\\2&4&0&3\\4&0&0&0\end{pmatrix}$	$\begin{pmatrix}1&1&1&1\\1&2&4&3\\1&3&4&2\\1&4&1&4\end{pmatrix}$	$\begin{pmatrix}1&1&1&1\\1&2&4&3\\1&3&4&2\\1&4&1&4\end{pmatrix}$
$(0,1,0)$	4	$(0,1,0)$	1	$\begin{pmatrix}1&4&1&4\\1&1&1&1\\2&0&2&0\end{pmatrix}$	$\begin{pmatrix}1&2&4&\not{4}\\1&3&4&\not{2}\\2&0&3&\not{0}\end{pmatrix}$	$\begin{pmatrix}1&1&1\\1&2&4\\1&3&4\end{pmatrix}$	$\begin{pmatrix}1&2&4\\1&3&4\\1&4&1\end{pmatrix}$

choice for the matrices are permutation matrices, i.e., 0-1 matrices with exactly one 1 per row and per column. We observe that each node stores $\alpha\ell$ symbols. For the nonuniform setting of semi-static data, we pad the shorter data blocks by an appropriate number of zeroes: as an example, a deletion in $\mathbf{x}$ at position 2 results in $\tilde{\mathbf{x}} = (x_1, x_3, \ldots, x_{\ell-1}, 0)$.

Scheme P: Suppose $\mathbf{x}^{(s)}$ is edited at coordinate i_s.

1 **if** edit is a deletion **then**
2 $\quad j_s \leftarrow$ coordinate where the i_sth row $\mathbf{A}^{(s)}$ is one (note: $\mathbf{A}^{(s)}$ is a permutation matrix)
3 $\quad$ shift i_sth row of $\mathbf{A}^{(s)}$ to the last row
4 **else**
5 $\quad j_s \leftarrow$ coordinate where the last row $\mathbf{A}^{(s)}$ is one
6 $\quad$ shift last row of $\mathbf{A}^{(s)}$ to the i_sth row
7 **end**
8 User s sends to the connected storage nodes $N(s)$: the value affected, using x ($\log q$ bits), the type of edit – insertion or deletion (one bit), and the coordinate j_s ($\log \ell$ bits)
9 **for** $t \in N(s)$ **do**
10 $\quad$ Compute $\mathbf{d} = \mathrm{Ec}^\ell\left(\mathbf{0}, \ldots, \mathbf{0}, x\mathbf{e}_{j_s}, \mathbf{0}, \ldots, \mathbf{0}\right)|_{\{t\}\times[\alpha]\times[\ell]}$
11 $\quad$ **if** edit is a deletion **then**
12 $\quad\quad$ subtract $\mathbf{d}$ from coordinate j_s at each storage node
13 $\quad$ **else**
14 $\quad\quad$ add $\mathbf{d}$ to coordinate j_s at each storage node
15 $\quad$ **end**
16 **end**

Let data block $\mathbf{x}^{(s)}$ be edited at coordinate i_s. Recall that we associate with $\mathbf{x}^{(s)}$ an $\ell \times \ell$ matrix $\mathbf{A}^{(s)}$. The matrix $\mathbf{A}^{(s)}$ is initialized to the identity matrix $\mathbf{I}$ and it remains a permutation matrix after each update. Roughly speaking, the storage nodes maintain the coded information in the original order. Since with each edit this order changes, the permutation matrix $\mathbf{A}^{(s)}$ is used to keep track of the order in the data blocks relative to that in the storage nodes. Hence, $\mathbf{A}^{(s)}$ indicates that instead of editing "position i_s" of the check nodes, one has to edit a position "position j_s" in the original order. These assertions are stated in Proposition 3. We call the permutation matrix intermediary encoding scheme *Scheme P*.

Proposition 3 (Scheme P)**.** *Consider an $(n,k,d,\alpha\ell,B\ell)$ DSS code and assume a single edit for a single user. The updates in accordance to Scheme P result in an $(n,k,d,\alpha\ell,B\ell)$ DSS code and the user needs to communicate $\log_2 \ell + \log_2 q$ bits to a connected storage node to update her/his information.*

Example 4. Consider two data blocks $\mathbf{u}^{(1)}$ and $\mathbf{u}^{(2)}$, shown in Table I, of length $\ell = 5$, as part of a $[3,2]$ MDS code over $\mathbb{F}_5$. Let $\mathbf{D}$ denote the tensor given by $\mathrm{Ec}^\ell\left(\mathbf{0}, \ldots, \mathbf{0}, x\mathbf{e}_{j_s}, \mathbf{0}, \ldots, \mathbf{0}\right)$ in line 10. We illustrate the steps of the protocol in Table I. We can reconstruct the user data by contacting any two nodes, obtaining the data stored in nodes 1 and 2, and multiplying it with the corresponding matrix inverses to obtain $\hat{\mathbf{u}}^{(1)} = (1,1,3,4,4)\left(\mathbf{A}^{(1)}\right)^{-1} = (1,3,4,1,4)$, and $\hat{\mathbf{u}}^{(2)} = (1,1,1,1,1)\left(\mathbf{A}^{(2)}\right)^{-1} = (1,1,1,1,1)$.

Notice that Scheme P requires the storage of the most updated "data order" and the deletion positions. To mitigate this undesirable feature, we propose the following scheme which changes the order automatically, and once data is deleted, all information about it is removed from the system. This also brings us to the second editing paradigm, *frequent editing*.

B. Frequently-Updated Data: Synchronization with Vandermonde Matrices

In contrast to the semi-static case, one may also assume that a significant proportion of the data blocks are edited or accessed by users[4]. In this case, suppose that the resulting data blocks are of length $\ell' < \ell$; then, the requirements are for each node to store

[4]In many application, both hot data (i.e. frequently accessed data) and frequently updated data is left uncoded in order to facilitate quick access to information and eliminate the need for re-encoding. Our scheme may be seen as a means to mitigate both the issues of access and re-encoding, while allowing higher level of data integrity through distributed coding. This is achieved at the price of intermediary code parameter storage and simple arithmetic procedures.

$\alpha\ell' < \alpha\ell$ symbols. The appropriate choices for the matrices $\tilde{\mathbf{A}}^{(s)}$ are Vandermonde or Cauchy matrices of dimension $\ell' \times \ell'$.

Let there be a single deletion at coordinate i_s in data block $\mathbf{x}^{(s)}$, for $s \in [B]$. The updated data block length equals $\ell' = \ell - 1$. Recall that we associate with $\mathbf{x}^{(s)}$ an $\ell \times \ell$ matrix $\mathbf{A}^{(s)}$. After synchronization, we want the updated matrix $\tilde{\mathbf{A}}^{(s)}$ to be of dimension $(\ell-1)\times(\ell-1)$ and invertible, and the information in the n storage nodes reduced to $\alpha \times (\ell - 1)$ arrays. The deleted values $x_{i_1}^{(1)}, x_{i_2}^{(2)}, \ldots, x_{i_B}^{(B)}$ are stored in the storage nodes as

$$\mathbf{D} \triangleq \mathrm{EC}^{\ell}\left(x_{i_1}^{(1)}\mathbf{A}|_{\{i_1\}\times[\ell]}, \ldots, x_{i_B}^{(B)}\mathbf{A}|_{\{i_B\}\times[\ell]}\right). \tag{4}$$

Hence, when given the values $x_{i_s}^{(s)}$ and positions i_s, each node t may subtract the vector $\mathbf{D}|_{\{t\}\times[\alpha]\times[\ell]}$ from its content. To reduce the size of the storage node arrays, we simply remove the coordinates in the set $[n] \times [\alpha] \times \{\ell\}$. Suppose $\tilde{\mathbf{A}}^{(s)}$ is the $(\ell-1)\times(\ell-1)$ matrix obtained from $\mathbf{A}^{(s)}$ by removing the i_sth row and last column. It is easy to check that *a posteriori* the edit, the storage nodes contain the tensor

$$\mathrm{EC}^{\ell-1}\left(\tilde{\mathbf{x}}^{(1)}\tilde{\mathbf{A}}^{(1)}, \tilde{\mathbf{x}}^{(2)}\tilde{\mathbf{A}}^{(2)}, \ldots, \tilde{\mathbf{x}}^{(B)}\tilde{\mathbf{A}}^{(B)}\right). \tag{5}$$

For the system to be an $(n, k, d, \alpha(\ell-1), B(\ell-1))$ DSS code, we require $\tilde{\mathbf{A}}^{(s)}$ to remain invertible. This is clearly true if the matrix $\mathbf{A}^{(s)}$ is a Vandermonde or a Cauchy matrix. We refer to the former method as *Scheme V*, the details of which are given below.

Scheme V: Symbol $\mathbf{x}^{(s)}$ is deleted at i_s, $s \in [B]$.

for $s \in [B]$ **do**
  User s sends to all connected storage nodes its deleted value deleted – $x_{i_s}^{(s)}$ – ($\log q$ bits) as well as the coordinate i_s of the deletion ($\log \ell$ bits)
  $\mathbf{A}^{(s)} \leftarrow \mathbf{A}^{(s)}$ via removal of the i_sth row and last column
end
for $t \in [n]$ **do**
  Using (4), compute and subtract $\mathbf{D}|_{\{t\}\times[\alpha]\times[\ell]}$
  Remove the (t, j, ℓ)th coordinate, for all $j \in [\alpha]$
end

Proposition 4 (Scheme V)**.** *Consider an $(n, k, d, \alpha\ell, B\ell)$ DSS code and assume single deletions in each user data block. The updates in accordance to Scheme V result in an $(n, k, d, \alpha(\ell-1), B(\ell-1))$ DSS code and each user needs to communicate $\log_2 \ell + \log_2 q$ bits to connected storage nodes.*

Example 5. Assume that $\ell = 4$ and that the DSS code is a $[3, 2]$ MDS code over $\mathbb{F}_5$. As before, we choose two data blocks $\mathbf{u}^{(1)}$ and $\mathbf{u}^{(2)}$ and illustrate the scheme in Table II. As in Example 4, we can reconstruct the user data blocks from the data stored in nodes 1 and 2: $\hat{\mathbf{u}}^{(1)} = (1, 2, 4)(\mathbf{A}^{(1)})^{-1} = (0, 1, 0)$, and $\hat{\mathbf{u}}^{(2)} = (1, 3, 4)(\mathbf{A}^{(2)})^{-1} = (0, 1, 0)$.

Application to data deduplication. Deduplication broadly refers to the process of removing duplicate copies of data with the objective of saving storage [6]. Scheme V may easily be integrated into a data deduplication process for a DSS so as to remove duplicates not only amongst the users, but also their redundantly encoded information at the storage nodes.

We describe how to accomplish this task for *post-process* deduplication, i.e., deduplication after users have already written on their disks certain data blocks, say $(f_1, \ldots, f_e) \in \mathbb{F}_q^e$. Deduplication proceeds as follows:

(I) A central node broadcasts to all users and nodes the data $(f_1, f_2, \ldots, f_e)$ to be removed.
(II) For $s \in [B]$, user s scans the string $\mathbf{x}^{(s)}$ for the data string $(f_1, f_2, \ldots, f_e)$ and identifies positions $i_{s,1}, i_{s,2}, \ldots, i_{s,e}$ where the blocks are stored.
(III) User s transmits positions $i_{s,1}, i_{s,2}, \ldots, i_{s,e}$ to all connected storage nodes.
(IV) Each storage node and user updates information as requested by Scheme V in e iterations.

REFERENCES

[1] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop distributed file system," in *IEEE Symp. Mass Storage Syst. Technologies*, 2010, pp. 1–10.
[2] A. G. Dimakis, P. B. Godfrey, Y. Wu, M. J. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," *IEEE Trans. Inform. Theory*, vol. 56, no. 9, pp. 4539–4551, 2010.
[3] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, and S. Yekhanin, "Erasure coding in Windows Azure storage," in *USENIX Annu. Technical Conference*, 2012, pp. 15–26.
[4] K. Rashmi, N. B. Shah, P. V. Kumar, and K. Ramchandran, "Explicit construction of optimal exact regenerating codes for distributed storage," in *Proc. 47th Annu. Alllerton Conf. Commun., Control, and Computing*, 2009, pp. 1243–1249.
[5] P. Gopalan, C. Huang, H. Simitci, and S. Yekhanin, "On the locality of codeword symbols," *IEEE Trans. Inform. Theory*, vol. 58, no. 11, pp. 6925–6934, 2012.
[6] D. Meister and A. Brinkmann, "Multi-level comparison of data deduplication in a backup scenario," in *Proc. SYSTOR 2009: The Israeli Experimental Systems Conference*. ACM, 2009, p. 8.
[7] I. Drago, M. Mellia, M. M Munafo, A. Sperotto, R. Sadre, and A. Pras, "Inside dropbox: understanding personal cloud storage services," in *Proc. the 2012 ACM conference on Internet measurement conference*. ACM, 2012, pp. 481–494.
[8] A. Tridgell, P. Mackerras *et al.*, "The rsync algorithm," 1996.
[9] T. Knauth and C. Fetzer, "dsync: Efficient block-wise synchronization of multi-gigabyte binary data." in *LISA*, 2013, pp. 45–58.
[10] A. Orlitsky and K. Viswanathan, "One-way communication and error-correcting codes," *IEEE Trans. Inform. Theory*, vol. 49, no. 7, pp. 1781–1788, 2003.
[11] Y. Minsky, A. Trachtenberg, and R. Zippel, "Set reconciliation with nearly optimal communication complexity," *IEEE Trans. Inform. Theory*, vol. 49, no. 9, pp. 2213–2218, 2003.
[12] R. Venkataramanan, H. Zhang, and K. Ramchandran, "Interactive low-complexity codes for synchronization from deletions and insertions," in *Proc. 48th Annu. Alllerton Conf. Commun., Control, and Computing*, 2010, pp. 1412–1419.
[13] N. Ma, K. Ramchandran, and D. Tse, "Efficient file synchronization: A distributed source coding approach," in *Proc. IEEE Intl. Symp. Inform. Theory*, 2011, pp. 583–587.
[14] S. Tabatabaei Yazdi and L. Dolecek, "A deterministic, polynomial-time protocol for synchronizing from deletions," *IEEE Trans. Inform. Theory*, vol. 60, no. 1, pp. 397–409, 2014.
[15] Y. Zhang, Y. Wu, and G. Yang, "Droplet: a distributed solution of data deduplication," in *Proc. the 2012 ACM/IEEE 13th International Conference on Grid Computing*. IEEE Computer Society, 2012, pp. 114–121.
[16] A. S. Rawat, S. Vishwanath, A. Bhowmick, and E. Soljanin, "Update efficient codes for distributed storage," in *Proc. IEEE Intl. Symp. Inform. Theory*, 2011, pp. 1457–1461.
[17] A. Mazumdar, V. Chandar, and G. W. Wornell, "Update-efficiency and local repairability limits for capacity approaching codes," *Selected Areas in Commun., IEEE Journal on*, vol. 32, no. 5, pp. 976–988, 2014.
[18] S. El Rouayheb, S. Goparaju, H. M. Kiah, and O. Milenkovic, "Synchronizing edits in distributed storage networks," *preprint arXiv:1409.1551*, 2014.