Synchronization and Deduplication in Coded Distributed Storage Networks

Salim El Rouayheb, *Member, IEEE*, Sreechakra Goparaju, Han Mao Kiah, and Olgica Milenkovic, *Senior Member, IEEE*

***Abstract*—We consider the problem of synchronizing coded data in distributed storage networks undergoing insertion and deletion edits. We present modifications of distributed storage codes that allow updates in the parity-check values to be performed with one round of communication at low bit rates and with small storage overhead. Our main contributions are novel protocols for synchronizing frequently updated and semi-static data based on functional intermediary coding involving permutation and Vandermonde matrices.**

***Index Terms*—Deduplication, deletions, distributed storage codes, file synchronization, insertions.**

I. Introduction

CODING for distributed storage systems (DSSs) has garnered significant attention in the past few years [2], [3] due to the rapid development of information technologies and the emergence of Big Data formats that need to be stored and disseminated across large-scale networks. As typical DSSs such as Google's Bigtable, Microsoft Azure and the Hadoop Distributed File System [4], [5] need to ensure low-latency data access and store a large number of files over a set of nodes connected through a communication network, it is imperative to protect the systems from undesired component failures. This is accomplished by implementing[1] Reed-Solomon, regenerating and local repair solutions [3], [6]–[9] that extend coding paradigms used in CDs, DVDs, flash memories and RAID systems [10].

Two key functionalities of codes used in DSSs are (i) *reconstruction* of files via access to a subset of the nodes; and (ii) content *repair* of failed nodes that does not compromise data reconstruction capabilities. Both these functionalities need to be retained when the files are accessed and processed by the users or undergo edits, such as symbol/block insertions, deletions, or substitutions. Deletions frequently arise due to system-level *data deduplication*, referring to a family of protocols designed to reduce storage costs by removing all except one or a few copies of the same file within the system. When parts of files are deduplicated or edited, the changes in the information content need to be communicated to the redundant parity-check nodes so that the DSS retains its reconstruction and repair capabilities, and such that the communication and update costs are minimized. Current solutions for synchronizing data that underwent edits assume that data is uncoded and they do not fully exploit the distributed nature of information. They mostly ignore the presence of deduplication protocols as these are believed to be in an early stage of development as far as distributed systems are concerned. Nevertheless, there are strong indications that deduplication is used in Dropbox and Sugarsync as well as in many other storage systems [11]–[13].

In Dropbox and related systems, deletion and insertion synchronization problems are resolved in the *uncoded* domain via the use of the rsync [14], dsync [15], or zsync [16] algorithms, related to a number of file synchronization methods put forward in the information theory literature [17]–[22]. There, uncoded stored copies of a file are synchronized from an edited user's copy. More specifically, the protocols involve a single user and a single node storing a replica of the user's file. After the user edits his/her file, assuming no knowledge of the edits, the user and the storage node communicate interactively until their files are matched or until one node matches the master copy in another node.

Synchronization in the coded domain is a significantly more complicated task as deletions and insertions cause issues that appear hard to resolve by standard substring pivoting and rolling checksum and hashing methods [14]. This family of protocols also requires fundamentally different techniques compared to those implemented for update efficient coding [23], [24]. Update efficient coding aims to minimize the number of nodes that need to be updated when a user's file is changed by substitutions. In this case, minimizing the communication cost reduces to simply minimizing the number of nodes a user needs to recruit during update. On the other hand, the core problem in coded synchronization is to efficiently update *encoded data* in storage nodes while maintaining small internode *communication rates*. In addition, instead of minimizing the number of storage nodes a

Manuscript received September 09, 2014; revised May 04, 2015 and September 17, 2015; accepted November 03, 2015; approved by IEEE/ACM Transactions on Networking Editor D. Goeckel. This work was supported in part by the NSF under Grants CCF 15-26875, CIF 12-18764, CIF 11-17980, and STC Class 2010, CCF 09-39370, and the Strategic Research Program of University of Illinois, Urbana-Champaign. Part of this work was completed when H. M. Kiah was with the University of Illinois at Urbana-Champaign. Initial results on the topic were presented at the 2015 IEEE International Symposium on Information Theory.

S. El Rouayheb is with the Department of Electrical and Computer Engineering, Illinois Institute of Technology, Chicago, IL 60616 USA.

S. Goparaju is with Calit2, University of California, San Diego, San Diego, CA 92093 USA.

H. M. Kiah is with the School of Physical and Mathematical Sciences, Nanyang Technological University, Singapore 637371, Singapore (e-mail: hmkiah@ntu.edu.sg).

O. Milenkovic is with the Coordinated Science Lab, University of Illinois at Urbana-Champaign, Urbana, IL 61801 USA.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNET.2015.2502274

[1] While locally repairable codes [3] have been implemented in industrial storage systems, regenerating codes are still in a developmental stage.

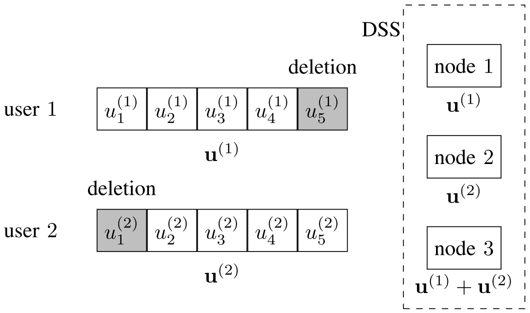

DSS
deletion
user 1
$u_1^{(1)}$ $u_2^{(1)}$ $u_3^{(1)}$ $u_4^{(1)}$ $u_5^{(1)}$
$\mathbf{u}^{(1)}$
deletion
user 2
$u_1^{(2)}$ $u_2^{(2)}$ $u_3^{(2)}$ $u_4^{(2)}$ $u_5^{(2)}$
$\mathbf{u}^{(2)}$
node 1
$\mathbf{u}^{(1)}$
node 2
$\mathbf{u}^{(2)}$
node 3
$\mathbf{u}^{(1)}+\mathbf{u}^{(2)}$

Fig. 1. Users in the DSS described in Example 1.

user needs to communicate with, the objective of synchronizing from deletions and insertions is to minimize the communication cost between a user and a storage node even at the cost of introducing a small, controlled amount of storage overhead.

We describe the *first* known coding and accompanying distributed synchronization protocols that maintain repair properties under a number of edit and deduplication models. The protocols rely on a simple new scheme termed *intermediary coding*, which changes the structure of the code during each edit or deduplication event and thereby reduces the communication complexity between nodes needed for content update. The intermediary coding 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 study extensions of the edit models arising in deduplication applications that cater to different data types, such as frequently rewritten and semi-static data. Our results also include worst and average case communication cost analyses for different edit models and synchronization protocols as used in typical HYDRAstor architectures [25]. The analysis reveals that traditional schemes require a significantly higher communication cost than schemes based on intermediary coding, both in the worst case and average case scenario and for all practical ranges of parameter values. This may be attributed to the fact that traditional encoding requires each node to communicate symbols in the *span* of all deletion positions in different nodes, while intermediary coding allows for reducing the communication cost to the number of bits needed to encode the particular edits.

The paper is organized as follows. Section II motivates the synchronization question and provides the precise problem formulation. Section III contains the description of simple update protocols and their underlying communication costs when traditional DSS encoding methods are used. Section IV contains our main results, a collection of encoding algorithms and protocols for data synchronization that have order-optimal communication cost with respect to the fundamental limits. Section V examines both storage and communication overheads and explains how to trade between these two system parameters, while Section VI provides a short discussion on how to handle unknown edit positions. Section VII presents numerical results indicating the communication savings of the proposed schemes.

II. Motivation and Problem Statement

We start with a straightforward example that motivates the work and describes the difficulties encountered when synchronizing coded data.

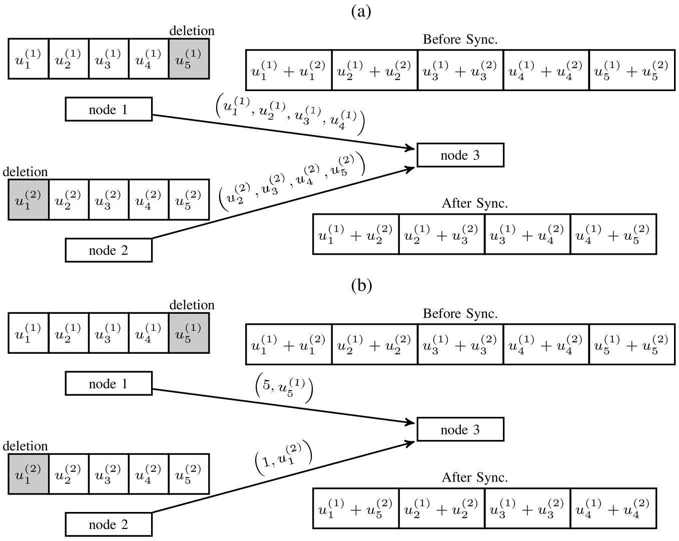

(a)
deletion
$u_1^{(1)}$ $u_2^{(1)}$ $u_3^{(1)}$ $u_4^{(1)}$ $u_5^{(1)}$
Before Sync.
$u_1^{(1)}+u_1^{(2)}$ $u_2^{(1)}+u_2^{(2)}$ $u_3^{(1)}+u_3^{(2)}$ $u_4^{(1)}+u_4^{(2)}$ $u_5^{(1)}+u_5^{(2)}$
node 1
$(u_1^{(1)},u_2^{(1)},u_3^{(1)},u_4^{(1)})$
node 3
deletion
$u_1^{(2)}$ $u_2^{(2)}$ $u_3^{(2)}$ $u_4^{(2)}$ $u_5^{(2)}$
$(u_2^{(2)},u_3^{(2)},u_4^{(2)},u_5^{(2)})$
After Sync.
$u_1^{(1)}+u_2^{(2)}$ $u_2^{(1)}+u_3^{(2)}$ $u_3^{(1)}+u_4^{(2)}$ $u_4^{(1)}+u_5^{(2)}$
node 2
(b)
deletion
$u_1^{(1)}$ $u_2^{(1)}$ $u_3^{(1)}$ $u_4^{(1)}$ $u_5^{(1)}$
Before Sync.
$u_1^{(1)}+u_1^{(2)}$ $u_2^{(1)}+u_2^{(2)}$ $u_3^{(1)}+u_3^{(2)}$ $u_4^{(1)}+u_4^{(2)}$ $u_5^{(1)}+u_5^{(2)}$
node 1
$(5,u_5^{(1)})$
node 3
deletion
$u_1^{(2)}$ $u_2^{(2)}$ $u_3^{(2)}$ $u_4^{(2)}$ $u_5^{(2)}$
$(1,u_1^{(2)})$
After Sync.
$u_1^{(1)}+u_5^{(2)}$ $u_2^{(1)}+u_2^{(2)}$ $u_3^{(1)}+u_3^{(2)}$ $u_4^{(1)}+u_4^{(2)}$
node 2

Fig. 2. Two synchronization schemes for coded data. The average communication cost is reduced from 4 bits with the traditional scheme (a) to $1+\log 5$ bits with our scheme (b). (a) A traditional synchronization scheme (see Example 1). (b) Our synchronization scheme (see Example 4).

Example 1: Consider two users with data blocks $\mathbf{u}^{(1)}=\left(u_1^{(1)},u_2^{(1)},u_3^{(1)},u_4^{(1)},u_5^{(1)}\right)$ and $\mathbf{u}^{(2)}=\left(u_1^{(2)},u_2^{(2)},u_3^{(2)},u_4^{(2)},u_5^{(2)}\right)$, each consisting of $\ell=5$ bits. Suppose that these blocks are encoded in a DSS comprising three nodes that store $\mathbf{u}^{(1)}$, $\mathbf{u}^{(2)}$, and $\mathbf{u}^{(1)}+\mathbf{u}^{(2)}$, respectively. The DSS coding scheme, illustrated in Fig. 1, satisfies the *reconstruction property* for the data blocks $\{\mathbf{u}^{(1)},\mathbf{u}^{(2)}\}$: one may 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 smaller data blocks $\tilde{\mathbf{u}}^{(1)}=\left(u_1^{(1)},u_2^{(1)},u_3^{(1)},u_4^{(1)}\right)$ and $\tilde{\mathbf{u}}^{(2)}=\left(u_2^{(2)},u_3^{(2)},u_4^{(2)},u_5^{(2)}\right)$. 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 on the edited data?

One way to retain both functionalities 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 DSS nodes, it is both necessary and sufficient for the user to communicate his/her deletion position in data block $\mathbf{u}^{(i)}$, $i=1,2$, to the corresponding node i. In Example 1, this amounts to communicating $\log 5$ bits. Regarding the parity-check node 3, a simple scenario involves both users transmitting their data blocks $\tilde{\mathbf{u}}^{(1)}$, $\tilde{\mathbf{u}}^{(2)}$ to node 3, which consequently updates its content to $\tilde{\mathbf{u}}^{(1)}+\tilde{\mathbf{u}}^{(2)}$. This solution requires an average communication cost of four symbols. The next proposition states that in general 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 two user data blocks of length ℓ over $\mathbb{F}_q$. Assume that exactly one deletion has occurred in $\mathbf{u}^{(1)}$ and one in $\mathbf{u}^{(2)}$. 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 scenario 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.*

The proof relies on concepts from communication complexity (see [26]) and is deferred to Appendix A.

It appears that in Example 1, both users have to necessarily transmit almost their whole information content to node 3. However, we show that by using methods akin to functional repair in DSSs [6], one may significantly save in communication complexity by updating the content 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. For clarity of exposition, we proceed to formally describe the problem setup and provide our solution for maximum distance separable (MDS) array codes only. The solution easily extends to a general setup including erasure, regenerating and locally repairable codes. The formal proofs of all assertions are deferred to Appendix E.

A. Coding for Distributed Storage Systems

Our model assumes that there are k users and that each user is associated with a data block of ℓ symbols.[2] We represent this information as an array $\mathbf{X} = \left(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(k)}\right) \in \mathbb{F}_q^{k\times\ell}$ to be stored in a distributed storage system. The system is equipped with an *encoding* function, as well as a set of reconstruction and repair algorithms. We focus on *maximum distance separable (MDS) array codes* [27], where the encoding algorithm converts the array $\mathbf{X}$ into n vectors of length ℓ and stores them in n *storage nodes*. The reconstruction algorithm recovers $\mathbf{X}$ from the contents of any k out of n nodes. Formally, we have the following definitions:

1) An *encoding* function is a map $\mathrm{Ec} : \mathbb{F}_q^{k\times\ell} \rightarrow \mathbb{F}_q^{n\times\ell}$.
2) Let $[n] \triangleq \{1, 2, \ldots, n\}$. For any k-subset T of $[n]$, a map $\mathrm{Rc}_T : \mathbb{F}_q^{k\times\ell} \rightarrow \mathbb{F}_q^{k\times\ell}$ is termed a *reconstruction* function if for $\left(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(k)}\right) \in \mathbb{F}_q^{k\times\ell}$,

$$\mathrm{Rc}_T\left(\mathrm{Ec}\left(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(k)}\right)\Big|_{T\times[\ell]}\right) = \left(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(k)}\right).$$

Here, $\mathbf{C}|_{R\times C}$ refers to the entries in the array $\mathbf{C}$ with coordinates in $R \times C$, while T refers to the nodes the users have to contact in order to retrieve their information.

We restrict our attention to linear code maps and refer to the above coding scheme as an $[n, k; \ell]$ MDS array code. When $\ell = 1$, we recover the usual notion of an $[n, k]$ MDS code [28]. On the other hand, given an $[n, k]$ MDS code and any integer ℓ, we may extend the code to obtain an $[n, k; \ell]$ MDS array code. This extension is akin to data-striping in RAID systems, amounting to a simple conversion of a symbol to a string of symbols. One may also view this construction as a means of dividing $k\ell$ symbols into ℓ groups of k symbols each, and then encoding each group of symbols by an $[n, k]$ MDS code.

In addition, we denote the set of nodes that need to be updated when user s edits his/her data block by $N(s)$, and say that the nodes in $N(s)$ are *connected to* user s.

Example 2 (Example 1 Continued): In Example 1, we implicitly considered a single parity $[3, 2]$ MDS code, with codewords of the form $\left(u^{(1)}, u^{(2)}, u^{(1)} + u^{(2)}\right)$, and $u^{(1)}, u^{(2)}$ belonging to a finite field. Assume that there are two users with data blocks $\mathbf{u}^{(1)}$ and $\mathbf{u}^{(2)}$ of length ℓ. Then by having nodes 1, 2 and 3 store $\mathbf{u}^{(1)}, \mathbf{u}^{(2)}, \mathbf{u}^{(1)} + \mathbf{u}^{(2)}$, we obtain a $[3, 2; \ell]$ MDS array code. Nodes 1 and 3 are connected to user 1, and nodes 2 and 3 are connected to user 2.

B. Problem Formulation

Consider a string $(x_1, x_2, \ldots, x_\ell)$. A *deletion* at position i results in $(x_1, x_2, \ldots, x_{i-1}, x_{i+1}, \ldots, x_\ell)$, while an *insertion* of symbol a at position i results in $(x_1, x_2, \ldots, x_{i-1}, a, x_i, \ldots, x_\ell)$. Suppose next that the data blocks in a DSS are updated via deletions or insertions. Furthermore, assume that the data blocks are subjected to edits performed in an independent fashion by k different users.[3] More precisely, the edit models studied include:

1) *The 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. Here the number of deletions is assumed to be $o(\ell/\log\ell)$. This model is used to describe the main ideas behind the work in a succinct and notationally simple manner, but may not be of practical importance.
2) *The nonuniform edits model,* in which each data block has a possibly different number of edits. We analyze this model in a probabilistic setting by assuming that one is given the probability p of deleting any particular symbol in a data block, resulting in an average number of $p\,\ell$ edits per data block. Note that p may depend on ℓ.

The problem of interest may be stated as follows:

Find the smallest "communication cost" protocol for the k users to send information about their edits to connected storage nodes so that the nodes can update their information while maintaining reconstruction and repair functionalities.

For concreteness, the "cost" is simply defined as the number of bits transmitted from a user to a storage node *connected* to it, *averaged over all users*. We are typically concerned with the worst case average cost, that is the cost maximized over all edit scenarios (say, all single deletion scenarios). For example, in the worst case, one may need to communicate $\ell\log q \geq \mathsf{cost} \geq \log\ell$ bits for a deletion, and $\ell\log q \geq \mathsf{cost} \geq \log\ell + \log q$ bits for an insertion. A scheme that achieves these bounds is for each user to send the entire edited data file to each node. The lower bound follows by noting that at least one user needs to communicate the edit position, and the inserted symbol in the case of an insertion, to at least one node.

In what follows, we introduce what we term a *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 such a low communication cost, we introduce a storage overhead needed to describe an intermediary encoding function. The gist of the encoding method is

[2] The choice of ℓ is dictated by the storage system at hand, and in particular, by the size of the block (which is a system parameter) and the expected size of an edit. Consequently, we assume for our algorithmic approaches that ℓ is a fixed system parameter. Nevertheless, for the asymptotic analysis performed in later sections, we use a standard modeling assumption in which ℓ is allowed to grow arbitrarily large.

[3] In practice, it may be possible for a single user to edit a number of different data blocks. However, for simplicity, we assume that each data block is edited by one user.

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. A key property of the transforms is that they reduce the update and synchronization communication cost by changing the code structure. The storage overhead induced by the change in code structure is carefully controlled by choosing the parameters of the corresponding matrices as described in Section V-B. We also demonstrate that our schemes are optimal in terms of storage allocation when the number of edits is $o(\ell/\log\ell)$. Under the same condition of $o(\ell/\log\ell)$ edits, we demonstrate in Appendix D that on average our schemes outperform schemes that do not utilize intermediary encoding functions. Furthermore, we point out that our methods do not rely on specific assumptions regarding the network topology: a user is allowed to communicate with any other user or storage node and vice versa, i.e., the storage network is a complete graph. Nevertheless, users are naturally assumed to communicate only with a "minimal" set of storage nodes which contain encodings or systematic repetitions of their data blocks.

III. Communication Cost With Traditional Encoding

We call a scheme *traditional* (or *Scheme T*) if for a given length ℓ of the data blocks, the corresponding $[n,k;\ell]$ MDS array code is a vectorized version of an $[n,k]$ MDS code as described in Section II-B. In other words, for a fixed ℓ, we encode k data blocks of ℓ units each by encoding the ith unit of each block using *the same* $[n,k]$ MDS code. Formally, define an $[n,k]$ MDS code via its corresponding Ec and Rc functions. The *traditional* encoding function[4] $\mathrm{Ec}^\ell:\mathbb{F}_q^{k\times\ell}\to\mathbb{F}_q^{n\times\ell}$, satisfies the following condition for any $i\in[\ell]$

$$\mathrm{Ec}^\ell\left(\mathbf{x}^{(1)},\dots,\mathbf{x}^{(k)}\right)\Big|_{[n]\times\{i\}}=\mathrm{Ec}\left(x_i^{(1)},\dots,x_i^{(k)}\right). \tag{1}$$

The corresponding reconstruction function $\mathrm{Rc}_T^\ell:\mathbb{F}_q^{k\times\ell}\to\mathbb{F}_q^{B\times\ell}$ satisfies the following condition for any k-subset T of $[n]$ and for any $i\in[\ell]$

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

where $\mathbf{C}$ is used to succinctly denote the encoded data $\mathrm{Ec}^\ell\left(\mathbf{x}^{(1)},\dots,\mathbf{x}^{(k)}\right)$ stored at the nodes. In words, given access to k nodes, we regard their information as a $k\times\ell$ array and apply the classical reconstruction algorithm to the ith column for all $i\in[\ell]$. Since the ith column is in fact $\mathrm{Ec}\left(x_i^{(1)},\dots,x_i^{(k)}\right)$, we retrieve the data units $\left(x_i^{(1)},\dots,x_i^{(k)}\right)$, which correspond to the ith coordinates of the data blocks $\mathbf{x}^{(1)},\dots,\mathbf{x}^{(k)}$. Hence, the function in (2) allows for retrieving the data blocks of all the k users.

Example 3 (Example 1 Continued): Consider the systematic single parity $[3,2]$ MDS code of Example 1 and let the data blocks of users 1 and 2 be $\mathbf{u}^{(1)}$, $\mathbf{u}^{(2)}\in\mathbb{F}_2^\ell$, respectively. For this code, we may use (1) and rewrite the information at the storage nodes as

$$\mathrm{Ec}^\ell\left(\mathbf{u}^{(1)},\mathbf{u}^{(2)}\right)=\begin{pmatrix}\mathbf{u}^{(1)}\\ \mathbf{u}^{(2)}\\ \mathbf{u}^{(1)}+\mathbf{u}^{(2)}\end{pmatrix}. \tag{3}$$

Suppose that following one edit per block, the data blocks are updated to $\tilde{\mathbf{x}}^{(1)},\dots,\tilde{\mathbf{x}}^{(k)}$, each of length $\ell-1$. Then, synchronization scheme T requires that the information stored at the n nodes be given by $\mathrm{Ec}^{\ell-1}\left(\tilde{\mathbf{x}}^{(1)},\dots,\tilde{\mathbf{x}}^{(k)}\right)$. The next proposition characterizes the worst-case cost of scheme T.

Proposition 2 (Scheme T): Consider an $[n,k;\ell]$ MDS array code and assume single deletions in the user data blocks. Scheme T updates the content of the storage nodes to an $[n,k;\ell-1]$ MDS array code, with each user sending out $\mathsf{cost}=|I|\log q$ bits to a connected storage node. Here, $I=\{i\in[\ell]:i_{\min}\leq i\leq i_{\max}-1\}$, where i_s denotes the deletion position for data block $s\in[k]$, $i_{\max}=\max_{s\in[k]}i_s$ and $i_{\min}=\min_{s\in[k]}i_s$. In the worst case, the protocol has a communication cost of $\mathsf{cost}=(\ell-1)\log q$ bits. Furthermore, no scheme performs better than this protocol up to a constant multiplicative factor.

Proof: Proposition 1 proves the converse for the case of an $n=3$, $k=2$ code. For the general case involving more than two users and more than three storage nodes, one can focus on the worst case scenario in which two users each have a single deletion and need to update a parity-check value in a common, connected node. The proof of Proposition 1 can be easily modified to show that in this case, the worst case cost remains $(\ell-1)$ symbols, or $(\ell-1)\log_2 q$ bits. The achievability scheme (Scheme T) is just an extension of the intuitive protocol one would apply in the worst case scenario (such as in Example 1), and is detailed below. □

Suppose that all users send their deleted coordinates to a designated central storage node, and that the designated central node computes $i_{\max}\triangleq\max_{s\in[k]}i_s$ and $i_{\min}\triangleq\min_{s\in[k]}i_s$. Define $I=\{i\in[\ell]:i_{\min}\leq i\leq i_{\max}-1\}$. Since we have $\tilde{x}_i^{(s)}=x_i^{(s)}$ for $i<i_{\min}$ and $s\in[k]$ and $\tilde{x}_i^{(s)}=x_{i+1}^{(s)}$ for $i\geq i_{\max}$ and $s\in[k]$, we have

$$\mathrm{Ec}^{\ell-1}\left(\tilde{\mathbf{x}}^{(1)},\dots,\tilde{\mathbf{x}}^{(k)}\right)\Big|_{[n]\times\{i\}}=\begin{cases}\mathrm{Ec}^\ell\left(\mathbf{x}^{(1)},\dots,\mathbf{x}^{(k)}\right)\big|_{[n]\times\{i\}}, & \text{if } i<i_{\min},\\ \mathrm{Ec}^\ell\left(\mathbf{x}^{(1)},\dots,\mathbf{x}^{(k)}\right)\big|_{[n]\times\{i+1\}}, & \text{if } i\geq i_{\max}.\end{cases} \tag{4}$$

In other words, the information stored at coordinates in $[n]\times([\ell]\setminus I)$ need not be updated. Therefore, it suffices to compute $\mathrm{Ec}^{\ell-1}\left(\tilde{\mathbf{x}}^{(1)},\dots,\tilde{\mathbf{x}}^{(k)}\right)\big|_{[n]\times I}$ as given by

$$\mathrm{Ec}^{\ell-1}\left(\tilde{\mathbf{x}}^{(1)},\dots,\tilde{\mathbf{x}}^{(k)}\right)\Big|_{[n]\times I}=\mathrm{Ec}^{|I|}\left(\tilde{\mathbf{x}}_I^{(1)},\dots,\tilde{\mathbf{x}}_I^{(k)}\right), \tag{5}$$

where $\tilde{\mathbf{x}}_I^{(s)}$ is the updated string restricted to the coordinates in I. In other words,

$$\tilde{\mathbf{x}}_I^{(s)}\triangleq\left(x_{i_{\min}}^{(s)},x_{i_{\min}+1}^{(s)},\dots,x_{i_s-1}^{(s)},x_{i_s+1}^{(s)},\dots,x_{i_{\max}}^{(s)}\right).$$

Consequently, equipped with $\tilde{\mathbf{x}}_I^{(s)}$ for $s\in[k]$ and (4) and (5), the nodes are able to compute $\mathrm{Ec}^{\ell-1}\left(\tilde{\mathbf{x}}^{(1)},\dots,\tilde{\mathbf{x}}^{(k)}\right)$.

IV. Synchronization Schemes With Order-Optimal Communication Cost

To avoid repeated transmissions of all data blocks of users, one needs to develop encoding methods that work around the problems associated with Scheme T. Synchronization schemes based on intermediary coding, which are the focus of this section, allow for reducing the communication cost from $O(\ell\log q)$ (Scheme T) to $O(\log\ell+\log q)$ bits.

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

Example 4 (Example 1 Continued): Suppose that each user sends the following information: (deletion position, deleted symbol) to node 3; i.e., user 1 transmits $\left(5, u_5^{(1)}\right)$ and user 2 transmits $\left(1, u_1^{(2)}\right)$. Now, node 3 which has $\mathbf{u}^{(1)} + \mathbf{u}^{(2)}$ and receives $u_1^{(2)}, u_5^{(1)}$, can update its content to $\left(u_1^{(1)} + u_5^{(2)}, u_2^{(1)} + u_2^{(2)}, u_3^{(1)} + u_3^{(2)}, u_4^{(1)} + u_4^{(2)}\right)$. The new storage code over the three nodes continues to be a $[3, 2; 4]$ MDS array code. We emphasize that in this array code, the third node does *not* store the sum of the content of the first two nodes. Nevertheless, the reconstruction capability is maintained. The communication cost is reduced from 8 bits, assuming Scheme T, to $2 \log 5 + 2 = 6.6$ bits (see Fig. 2). The savings are more dramatic as block lengths increase: for $\ell = 1024$, the communication cost may be reduced from 2048 bits to only $2 \log 1024 + 2 = 22$ bits.

In what follows, we formally introduce the notion of *intermediary encoding*, illustrated in Fig. 3.

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

$$\textsc{Ec}^{*\ell}\left(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(k)}\right) \triangleq \textsc{Ec}^{\ell}\left(\mathbf{x}^{(1)}\mathbf{A}^{(1)}, \ldots, \mathbf{x}^{(k)}\mathbf{A}^{(k)}\right). \tag{1a}$$

We prove that this encoding function preserves reconstruction capabilities by exhibiting a reconstruction algorithm. To this end, for any k-subset T of $[n]$, we define $\textsc{Rc}_T^{*\ell} : \mathbb{F}_q^{k\times\ell} \to \mathbb{F}_q^{k\times\ell}$, so that for all $s \in [k]$,

$$\textsc{Rc}_T^{*\ell}(\mathbf{C}|_{T\times[\ell]})|_{\{s\}\times[\ell]} \triangleq \textsc{Rc}_T^{\ell}(\mathbf{C}|_{T\times[\ell]})|_{\{s\}\times[\ell]}, \left(\mathbf{A}^{(s)}\right)^{-1}. \tag{2a}$$

Proposition 3: Consider an $[n, k]$ MDS code with functions $\textsc{Ec}$ *and* $\textsc{Rc}$. *For any integer ℓ and k invertible $\ell \times \ell$ matrices $\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(k)}$, the functions* $\textsc{Ec}^*$ *and* $\textsc{Rc}^*$ *given by* (1a) *and* (2a) *define an $[n, k; \ell]$ MDS array code.*

Note that $\mathbf{A}^{(s)} = \mathbf{I}$, $\forall\ s \in [k]$, corresponds to Scheme T.

Example 5 (Example 1 Continued): Suppose node 3 stores the value $\left(u_1^{(1)} + u_5^{(2)}, u_2^{(1)} + u_2^{(2)}, u_3^{(1)} + u_3^{(2)}, u_4^{(1)} + u_4^{(2)}\right)$. Then, we may rewrite the information stored at all three nodes as

$$\textsc{Ec}^{*2}\left(\tilde{\mathbf{u}}^{(1)}, \tilde{\mathbf{u}}^{(2)}\right) = \textsc{Ec}^{2}\left(\tilde{\mathbf{u}}^{(1)}\mathbf{A}^{(1)}, \tilde{\mathbf{u}}^{(2)}\mathbf{A}^{(2)}\right),$$

where $\mathbf{A}^{(1)} = \mathbf{I}$ and $\mathbf{A}^{(2)} = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$. Since both $\mathbf{A}^{(1)}$ and $\mathbf{A}^{(2)}$ are invertible, the resulting code is indeed an $[3, 2; 4]$ MDS array code.

In the next subsection, we describe how to choose the matrices $\mathbf{A}^{(s)}$ and accompanying update protocols for a general setting, so as to ensure significantly lower communication cost between a user and connected storage node as compared to Scheme T. Furthermore, In Appendix B, we describe simple modifications of (1a) that enable the use of systematic MDS codes and that apply to variable data block lengths.

A. Synchronization Schemes Using Intermediary Coding

Suppose that the data blocks are edited to length $\ell' < \ell$. The idea behind intermediary coding 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 only 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 (1a) holds. Since the matrices $\tilde{\mathbf{A}}^{(s)}$ are designed to be invertible, the resulting system remains an $[n, k; \ell']$ MDS array code. We propose two different update schemes and choices for the matrices $\tilde{\mathbf{A}}^{(s)}$ based on the frequency and extent to which edits are made. These matrices cater to the need of:

(a) *Semi-static Data*. For this scenario, 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 choice for the matrices are permutation matrices, i.e., binary matrices with exactly one 1 per row and per column.

(b) *Frequently Updated Data*. In contrast to the semi-static case, one may also assume that a significant proportion of the data blocks are edited by users.[5] In this case, the resulting stored data blocks are of length $\ell' < \ell$; the matrices $\tilde{\mathbf{A}}^{(s)}$ have dimension $\ell' \times \ell'$ and appropriate choices for them are Vandermonde and Cauchy matrices (see [28]).

Scheme Based on Permutation Matrices: Although this scheme applies to general nonuniform edit models, and consequently to data blocks of variables lengths, for simplicity we assume edits of the form of a single deletion or insertion. In the nonuniform setting we pad the shorter data blocks by an appropriate number of zeros. As an illustration, after a deletion in $\mathbf{x}$ at position 2, we store $\tilde{\mathbf{x}} = (x_1, x_3, \ldots, x_{\ell-1}, 0)$.

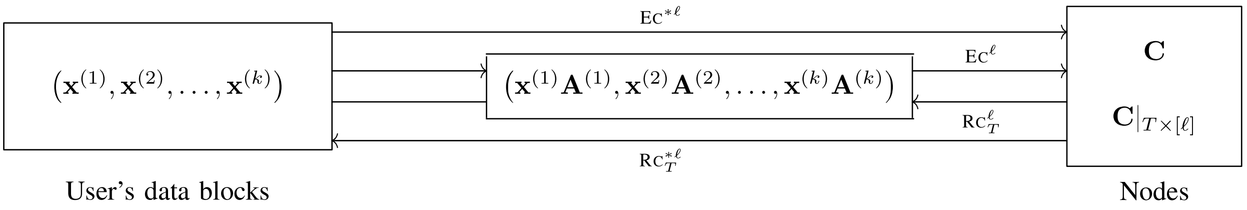
Let the data block $\mathbf{x}^{(s)}$ be edited at coordinate i_s. Recall that we associate with $\mathbf{x}^{(s)}$ an $\ell \times \ell$ matrix $\mathbf{A}^{(s)}$. The matrix $\mathbf{A}^{(s)}$ is initialized to the identity matrix $\mathbf{I}$ and it remains a permutation matrix after each update. 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 "position j_s" in the original order. These assertions are stated and formally proved in Proposition 4.

Since permutation matrices are invertible, the approach results in an $[n, k; \ell]$ MDS code by Proposition 3. We refer to the intermediary encoding scheme that uses permutation matrices as *Scheme P*.

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

```
if edit is a deletion then
    $j_s$ ← coordinate where the $i_s$th row $\mathbf{A}^{(s)}$ is one
    (note: $\mathbf{A}^{(s)}$ is a permutation matrix)
```

[5] 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 reencoding. Our scheme may be seen as a means to mitigate both the issues of access and reencoding, while allowing higher level of data integrity through distributed coding. This is achieved at the price of storing an intermediary code and additional computations.



$\textsc{Ec}^{*\ell}$
$\left(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(k)}\right)$
$\left(\mathbf{x}^{(1)}\mathbf{A}^{(1)}, \mathbf{x}^{(2)}\mathbf{A}^{(2)}, \ldots, \mathbf{x}^{(k)}\mathbf{A}^{(k)}\right)$
$\textsc{Ec}^{\ell}$
$\textsc{Rc}_T^{\ell}$
$\mathbf{C}$
$\mathbf{C}|_{T\times[\ell]}$
$\textsc{Rc}_T^{*\ell}$
User's data blocks
Nodes

Fig. 3. Intermediary encoding that retains both reconstruction and repair functionalities.

```
3       $i_s$th row of $\mathbf{A}^{(s)}$ is moved to the position of the last row
4   else
5       $j_s \leftarrow$ coordinate where the last row $\mathbf{A}^{(s)}$ is one
6       last row of $\mathbf{A}^{(s)}$ is moved to the position of the $i_s$th 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      One computes
        $\mathbf{D} = \textsc{Ec}^{\ell}\left(\mathbf{0}, \ldots, \mathbf{0}, x\mathbf{e}_{j_s}, \mathbf{0}, \ldots, \mathbf{0}\right)|_{\{t\}\times[\ell]}$
11      where for $j \in [\ell]$, $\mathbf{e}_j$ denotes the $j$th standard basis
        vector.
12      if edit is a deletion then
13          subtract $\mathbf{D}|_{\{t\}\times[\ell]}$
14      else
15          add $\mathbf{D}|_{\{t\}\times[\ell]}$
16      end
17  end
```

Proposition 4 (Scheme P): Consider an $[n, k; \ell]$ MDS array code and assume a single edit for a single user. The updates in accordance to Scheme P result in an $[n, k; \ell]$ MDS array code and the user needs to communicate $\log \ell + \log q$ *bits to a connected storage node to update his/her information.*

Proof of Proposition 4: We prove the correctness of Scheme P for the case of a single deletion. Insertions are handled similarly.

We first show that the updates performed as part of Scheme P result in an $[n, k; \ell]$ MDS code. It suffices to show that the nodes store

$$\textsc{Ec}^{\ell}\left(\mathbf{x}^{(1)}\mathbf{A}^{(1)}, \mathbf{x}^{(2)}\mathbf{A}^{(2)}, \ldots, \tilde{\mathbf{x}}^{(s)}\tilde{\mathbf{A}}^{(s)}, \ldots, \mathbf{x}^{(k)}\mathbf{A}^{(k)}\right),$$

where $\tilde{\mathbf{A}}^{(s)}$ is the matrix resulting from the instruction performed from line 1 to line 7 in the protocol. Note that prior to the edit, the nodes stored

$$\textsc{Ec}^{\ell}\left(\mathbf{x}^{(1)}\mathbf{A}^{(1)}, \ldots, \mathbf{x}^{(s)}\mathbf{A}^{(s)}, \ldots, \mathbf{x}^{(k)}\mathbf{A}^{(k)}\right).$$

Define

$$\begin{aligned}\mathbf{D} &\triangleq \textsc{Ec}^{\ell}\left(\mathbf{x}^{(1)}\mathbf{A}^{(1)}, \ldots, \mathbf{x}^{(s)}\mathbf{A}^{(s)}, \ldots, \mathbf{x}^{(k)}\mathbf{A}^{(k)}\right)\\ &\quad - \textsc{Ec}^{\ell}\left(\mathbf{x}^{(1)}\mathbf{A}^{(1)}, \ldots, \tilde{\mathbf{x}}^{(s)}\tilde{\mathbf{A}}^{(s)}, \ldots, \mathbf{x}^{(k)}\mathbf{A}^{(k)}\right)\\ &= \textsc{Ec}^{\ell}\left(\mathbf{0}, \ldots, \mathbf{0}, \mathbf{x}^{(s)}\mathbf{A}^{(s)} - \tilde{\mathbf{x}}^{(s)}\tilde{\mathbf{A}}^{(s)}, \mathbf{0}, \ldots, \mathbf{0}\right),\end{aligned}$$

and suppose that the following claim were true

$$\mathbf{x}^{(s)}\mathbf{A}^{(s)} - \tilde{\mathbf{x}}^{(s)}\tilde{\mathbf{A}}^{(s)} = x\mathbf{e}_{j_s}. \tag{6}$$

Then, the updates in lines 9 to 16 have the net effect of subtracting $\mathbf{D}$ from the information stored at the nodes, which proves the claim. Hence, it only remains to show that (6) holds.

Without loss of generality, let $\mathbf{x}^{(s)} = (x_1, x_2, \ldots, x_\ell)$. Then $\tilde{\mathbf{x}}^{(s)} = (x_1, x_2, \ldots, x_{i_s-1}, x_{i_s+1}, \ldots, x_\ell, 0)$ and $x = x_{i_s}$.

Since $\mathbf{A}^{(s)}|_{\{i\}\times[\ell]} = \tilde{\mathbf{A}}^{(s)}|_{\{i\}\times[\ell]}$ for $i \le i_s - 1$, we have $x_i\mathbf{A}^{(s)}|_{\{i\}\times[\ell]} - x_i\tilde{\mathbf{A}}^{(s)}|_{\{i\}\times[\ell]} = \mathbf{0}$. Similarly, $x_i\mathbf{A}^{(s)}|_{\{i\}\times[\ell]} - x_i\tilde{\mathbf{A}}^{(s)}|_{\{i-1\}\times[\ell]} = \mathbf{0}$ for $i_s + 1 \le i \le \ell$. Hence, the left hand side yields $x_{i_s}\tilde{\mathbf{A}}^{(s)}|_{\{i_s\}\times[\ell]} - 0 \cdot \tilde{\mathbf{A}}^{(s)}|_{\{\ell\}\times[\ell]} = x\mathbf{e}_{j_s}$. □

Example 6: Consider a simple example involving two data blocks $\mathbf{u}^{(1)}$ and $\mathbf{u}^{(2)}$ with $\ell = 5$, shown in Table I, that represent part of an $[3, 2; 5]$ MDS array code over $\mathbb{F}_5$. Let $\mathbf{D}$ denote the matrix given by $\textsc{Ec}^{\ell}\left(\mathbf{0}, \ldots, \mathbf{0}, x\mathbf{e}_{j_s}, \mathbf{0}, \ldots, \mathbf{0}\right)$ in line 10 of the protocol. One can reconstruct the user data by contacting any two nodes. Upon obtaining the data stored in nodes 1 and 2, and multiplying it with the corresponding matrix inverses, one arrives at $\hat{\mathbf{u}}^{(1)} = (1, 1, 3, 4, 4)\left(\mathbf{A}^{(1)}\right)^{-1} = (1, 3, 4, 1, 4)$, and $\hat{\mathbf{u}}^{(2)} = (1, 1, 1, 1, 1)\left(\mathbf{A}^{(2)}\right)^{-1} = (1, 1, 1, 1, 1)$.

One shortcoming of Scheme P is that it requires storing 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.

Scheme Based on Vandermonde Matrices: As for the case of Scheme P, the scheme using Vandermonde matrices applies to nonuniform edit models as well,[6] but for simplicity we assume a single deletion at coordinate i_s in data block $\mathbf{x}^{(s)}$, $\forall s \in [k]$. Note that when one deletion is present, the updated data block length equals $\ell' = \ell - 1$.

As before, 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 appropriate vectors of length $(\ell - 1)$. The deleted values $x_{i_1}^{(1)}, x_{i_2}^{(2)}, \ldots, x_{i_k}^{(k)}$ are stored in the storage nodes as

$$\mathbf{D} \triangleq \textsc{Ec}^{\ell}\left(x_{i_1}^{(1)}\mathbf{A}|_{\{i_1\}\times[\ell]}, \ldots, x_{i_k}^{(k)}\mathbf{A}|_{\{i_k\}\times[\ell]}\right). \tag{7}$$

Hence, when given the values $x_{i_s}^{(s)}$ and positions i_s, each node t may subtract the vector $\mathbf{D}|_{\{t\}\times[\ell]}$ from its content. To reduce the size of the storage node arrays, we simply remove the coordinates in the set $[n] \times \{\ell\}$.

Suppose that $\tilde{\mathbf{A}}^{(s)}$ is the $(\ell - 1) \times (\ell - 1)$ matrix obtained from $\mathbf{A}^{(s)}$ by removing the i_sth row and the last column. It is easy to check that after the edit, the storage nodes contain

$$\textsc{Ec}^{\ell-1}\left(\tilde{\mathbf{x}}^{(1)}\tilde{\mathbf{A}}^{(1)}, \tilde{\mathbf{x}}^{(2)}\tilde{\mathbf{A}}^{(2)}, \ldots, \tilde{\mathbf{x}}^{(k)}\tilde{\mathbf{A}}^{(k)}\right). \tag{8}$$

For the system to be an $[n, k; \ell]$ MDS array code, we require $\tilde{\mathbf{A}}^{(s)}$ to remain invertible. This is clearly true if the matrix $\mathbf{A}^{(s)}$ is Vandermonde. We refer to this method as *Scheme V.*

[6] Minor algorithmic changes that need to be performed in order to accommodate the nonuniform deletion model are outlined at the end of the section.

TABLE I
EXAMPLE ILLUSTRATING SCHEME P WITH AN $[3, 2]$ MDS CODE OVER $\mathbb{F}_5$ (EXAMPLE 6).

The datablocks $\mathbf{u}^{(1)}$ and $\mathbf{u}^{(2)}$ are initialized to $(1, 2, 3, 4, 4)$ and $(1, 1, 1, 1, 1)$, respectively. The information stored at the nodes is $\mathbf{u}^{(1)}, \mathbf{u}^{(2)}, \mathbf{u}^{(1)} + \mathbf{u}^{(2)}$, as shown in the first row of the table. When the first block undergoes a deletion at the second coordinate, $\mathbf{A}^{(1)}$ is adjusted according to Scheme P, lines 2 to 3, and user 1 sends the deleted value and its position to nodes 1 and 3. The nodes then compute the relevant rows in $\mathbf{D}$ according to line 10 of the algorithm and subsequently update their information. The net change $\mathbf{D}$, the updated data stored at the nodes, and the matrices $\mathbf{A}^{(1)}$ and $\mathbf{A}^{(2)}$ are given in the second row. A similar procedure is executed when the block experiences an insertion.

$\mathbf{u}^{(1)}$	$\mathbf{u}^{(2)}$	edit	$\mathbf{D}$	$\text{Ec}^5\left(\mathbf{u}^{(1)}\mathbf{A}^{(1)}, \mathbf{u}^{(2)}\mathbf{A}^{(2)}\right)$	$\mathbf{A}^{(1)}$	$\mathbf{A}^{(2)}$
$(1, 2, 3, 4, 4)$	$(1, 1, 1, 1, 1)$	—	—	$\begin{pmatrix} 1&2&3&4&4 \\ 1&1&1&1&1 \\ 2&3&4&0&0 \end{pmatrix}$	$\begin{pmatrix} 1&0&0&0&0 \\ 0&1&0&0&0 \\ 0&0&1&0&0 \\ 0&0&0&1&0 \\ 0&0&0&0&1 \end{pmatrix}$	$\begin{pmatrix} 1&0&0&0&0 \\ 0&1&0&0&0 \\ 0&0&1&0&0 \\ 0&0&0&1&0 \\ 0&0&0&0&1 \end{pmatrix}$
$(1, 3, 4, 4, 0)$	$(1, 1, 1, 1, 1)$	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 AN $[3, 2]$ MDS CODE OVER $\mathbb{F}_5$ (EXAMPLE 7).

We initialize the data blocks $\mathbf{u}^{(1)}$ and $\mathbf{u}^{(2)}$ and the matrices $\mathbf{A}^{(1)}$ and $\mathbf{A}^{(2)}$ as shown in the first row. When each of the data blocks undergoes a single deletion, $\mathbf{A}^{(1)}$ and $\mathbf{A}^{(2)}$ are adjusted according to the protocol of Scheme V, lines 2 to 3, and both users sends their deleted values and positions to the parity nodes. The nodes then compute the relevant rows in $\mathbf{D}$ according to (7).The net change $\mathbf{D}$,the updated stored data, matrices $\mathbf{A}^{(1)}$ and $\mathbf{A}^{(2)}$ are listed in the second row.

$\mathbf{u}^{(1)}$	i_1	$\mathbf{u}^{(2)}$	i_2	$\mathbf{D}$	$\text{Ec}(\ell)\left(\mathbf{u}^{(1)}\mathbf{A}^{(1)}, \mathbf{u}^{(2)}\mathbf{A}^{(2)}\right)$	$\mathbf{A}^{(1)}$	$\mathbf{A}^{(2)}$
$(0, 1, 0, 1)$	—	$(1, 0, 1, 0)$	—	—	$\begin{pmatrix} 2&1&0&2 \\ 2&4&0&3 \\ 4&0&0&0 \end{pmatrix}$	$\begin{pmatrix} 1&1&1&1 \\ 1&2&4&3 \\ 1&3&4&2 \\ 1&4&1&4 \end{pmatrix}$	$\begin{pmatrix} 1&1&1&1 \\ 1&2&4&3 \\ 1&3&4&2 \\ 1&4&1&4 \end{pmatrix}$
$(0, 1, 0)$	4	$(0, 1, 0)$	1	$\begin{pmatrix} 1&4&1&4 \\ 1&1&1&1 \\ 2&0&2&0 \end{pmatrix}$	$\begin{pmatrix} 1&2&4&\cancel{3} \\ 1&3&4&\cancel{2} \\ 2&0&3&\cancel{0} \end{pmatrix}$	$\begin{pmatrix} 1&1&1 \\ 1&2&4 \\ 1&3&4 \end{pmatrix}$	$\begin{pmatrix} 1&2&4 \\ 1&3&4 \\ 1&4&1 \end{pmatrix}$

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

```
1 for $s \in [k]$ do
2     User $s$ sends to all connected storage nodes its deleted
      value – $x^{(s)}_{i_s}$ – using $\log q$ bits, as well as the position $i_s$
      of the deletion, using $\log \ell$ bits
3     $\mathbf{A}^{(s)} \leftarrow \mathbf{A}^{(s)}$ via removal of the $i_s$th row and last
      column
4 end
5 for $t \in [n]$ do
6     Using (7), one computes and subtracts $\mathbf{D}|_{\{t\}\times[\ell]}$
7     and removes the $\ell$th coordinate.
8 end
```

Proposition 5 (Scheme V): Consider an $[n, k; \ell]$ MDS array code and assume a single deletion in each user data block. The updates in accordance to Scheme V result in an $[n, k; \ell-1]$ MDS array code and each user needs to communicate $\log \ell + \log q$ bits to connected nodes.

Proof of Proposition 5: As before, it suffices to show that the information stored at the nodes is given by (8). Proceeding in a similar fashion as was done for the proof of Proposition 4, and by referring to the linearity of the encoding maps, the proof reduces to showing that for $s \in [k]$,

$$\left(\mathbf{x}^{(s)}\mathbf{A}^{(s)}\right)\Big|_{[\ell-1]} - \tilde{\mathbf{x}}^{(s)}\tilde{\mathbf{A}}^{(s)} = x^{(s)}_{i_s}\mathbf{A}\Big|_{\{i_s\}\times[\ell-1]}. \qquad (9)$$

We check that $x^{(s)}_i\mathbf{A}^{(s)}|_{\{i\}\times[\ell-1]} - x^{(s)}_i\tilde{\mathbf{A}}^{(s)}|_{\{i\}\times[\ell-1]} = \mathbf{0}$ for $i \leq i_s - 1$ and $x^{(s)}_i\mathbf{A}^{(s)}|_{\{i\}\times[\ell-1]} - x^{(s)}_i\tilde{\mathbf{A}}^{(s)}|_{\{i-1\}\times[\ell-1]} = \mathbf{0}$ for $i_s + 1 \leq i \leq \ell$. Hence, the remaining term on the left hand side of the previous expression equals $x^{(s)}_{i_s}\mathbf{A}|_{\{i_s\}\times[\ell-1]}$, which establishes (9). □

Example 7: Assume that $\ell = 4$ and consider an $[3, 2; 4]$ MDS array code over $\mathbb{F}_5$. Choose two data blocks $\mathbf{u}^{(1)}$ and $\mathbf{u}^{(2)}$ and edits as shown in Table II. As in Example 6, one can reconstruct the user data blocks from the data stored in nodes 1 and 2 according to: $\hat{\mathbf{u}}^{(1)} = (1, 2, 4)(\mathbf{A}^{(1)})^{-1} = (0, 1, 0)$, and $\hat{\mathbf{u}}^{(2)} = (1, 3, 4)(\mathbf{A}^{(2)})^{-1} = (0, 1, 0)$.

Remark 8:

(i) *The size of ℓ.* Recall that Vandermonde matrices exist whenever $\ell \leq q$, hence this inequality has to hold true for the scheme to be implementable.

(ii) *Intermediary coding based on Cauchy matrices as a means of reducing communication complexity.* In Scheme V, all users were required to transmit $(\log \ell + \log q)$ bits to describe their edits. We may save $\log q$ bits in the communication protocol by having one of the users, say user 1, transmit only its location, without transmitting the deleted value.

We achieve this by fixing $\mathbf{A}^{(1)} = \mathbf{I}$. Suppose that $\mathbf{x}^{(1)}$ had a deletion at position i_1. To ensure that $\tilde{\mathbf{A}}^{(1)}$ remains invertible, we remove the i_1th row and i_1th column (line 3). This in turn forces us to delete the i_1th column in all matrices $\mathbf{A}^{(s)}$. Hence, one needs to ensure that all square submatrices of $\mathbf{A}^{(s)}$ are invertible for $s \geq 2$. It is known that $\ell \times \ell$ Cauchy matrices, taking the form

$$\begin{bmatrix} (a_1 - b_1)^{-1} & (a_1 - b_2)^{-1} & \dots & (a_1 - b_\ell)^{-1} \\ (a_2 - b_1)^{-1} & (a_2 - b_2)^{-1} & \dots & (a_2 - b_\ell)^{-1} \\ \vdots & \vdots & \ddots & \vdots \\ (a_\ell - b_1)^{-1} & (a_\ell - b_2)^{-1} & \dots & (a_\ell - b_\ell)^{-1} \end{bmatrix},$$

for distinct values $a_1, \dots, a_\ell, b_1, \dots, b_\ell$, satisfy this requirement. Cauchy matrices of the form needed exist whenever $2\ell \leq q$.

(iii) *Application to data deduplication.* As already pointed out, deduplication is inherently a deletion process which requires synchronizing all affected user information. Scheme V may easily be integrated into a data deduplication process for a DSS so as to remove duplicate blocks 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 recorded on their disks certain data blocks, say $(f_1, \dots, f_e) \in \mathbb{F}_q^e$. Deduplication proceeds as follows:

(I) A central node identifies all users and their corresponding storage nodes for which the data $(f_1, f_2, \dots, f_e)$ is to be removed.

(II) For $s \in [k]$, the central node scans the string $\mathbf{x}^{(s)}$ of user node s for $(f_1, f_2, \dots, f_e)$ and identifies positions $i_{s,1}, i_{s,2}, \dots, i_{s,e}$ where the blocks are stored.

(III) The central node transmits $i_{s,1}, i_{s,2}, \dots, i_{s,e}$ to all storage nodes connected to s.

(IV) Each storage node and user node update their information as dictated by Scheme V in e iterations.

(iv) *RESET synchronization.* For many deduplication scenarios, one is faced with a potentially large number of deletions of symbols. In this setting, it may be more efficient to communicate information about symbols that remained unedited, rather than information about deleted symbol. A simple counterpart of Scheme T for the large deletion regime is a synchronization scheme during which all the users send their *undeleted data blocks* to the parity nodes to recompute their values. We term this scheme RESET, and in the high deletion regime, this scheme may be expected to outperform its counterparts that communicate information about the deleted symbols.

Nonuniform Deletion Model: We briefly comment on how to modify Scheme V so as to adapt it to the general scenario where user $\mathbf{x}^{(s)}$ has d_s deletions. Define ℓ' as $\max_{s\in[k]} \ell - d_s$, or equivalently, $\ell - \min_{s\in[k]} d_s$. Our goal is to update the matrix $\tilde{\mathbf{A}}^{(s)}$ so that it has dimension $(\ell - d_s) \times \ell'$, and reduce the content of the n storage nodes to $\alpha \times \ell'$ arrays.

Suppose that $\mathbf{x}^{(s)}$ has deleted values $x_{i_{s1}}^{(s)}, x_{i_{s2}}^{(s)}, \dots, x_{i_{sd_s}}^{(s)}$. Define $\mathbf{d}^{(s)} \triangleq \sum_{t=1}^{d_s} x_{i_{st}}^{(s)} \mathbf{A}|_{\{i_{st}\}\times[\ell]}$ and modify (7) as

$$\mathbf{D} \triangleq \mathrm{Ec}^{\ell}\left(\mathbf{d}^{(1)}, \mathbf{d}^{(2)}, \dots, \mathbf{d}^{(k)}\right). \tag{8a}$$

As before, when given the deleted values and their coordinates, each node t subtracts the vector $\mathbf{D}|_{\{t\}\times[\ell]}$. To reduce the size of the storage nodes, we simply remove the coordinates belonging to $[n]\times\{i\}$ for $\ell'+1 \leq i \leq \ell$. Suppose that $\tilde{\mathbf{A}}^{(s)}$ is the $(\ell - d_s) \times \ell'$ matrix with rows corresponding to the d_s deletions and last $\ell - \ell'$ columns removed from $\mathbf{A}^{(s)}$. Then it is not difficult to check that the matrix $\tilde{\mathbf{A}}^{(s)}$ is Vandermonde if $\mathbf{A}^{(s)}$ was Vandermonde in the first place.

V. Fundamental Limits and a Tradeoff Between Communication Cost and Storage Overhead

Suppose that a data block of a user is subjected to a *single* edit. Then, the communication cost of both Schemes P and V is approximately $\log \ell + \log q$, for each pair of user and his/her connected storage node. Hence, for a single edit, the communication cost of both schemes is near-optimal.

However, when a data block has an arbitrary number of edits, say d, then the schemes require $d(\log \ell + \log q)$ bits to be communicated and it is unclear if this number is optimal, order-optimal (i.e., of the same order as the optimal solution) or order-suboptimal. In the next Section V-A, we establish a simple lower bound on the communication cost using results of Levenshtein [29] and show that Schemes P and V are within a constant factor away from the lower bound when $d = o(\ell^{1-\epsilon})$, for a constant $0 < \epsilon < 1$.

We point out that Schemes P and V outperform Scheme T for any number of edits. To achieve the communication cost of $O(\log \ell + \log q)$ bits, Schemes P and V have to store certain structural information regarding the matrices $\mathbf{A}^{(s)}$. As Scheme T does not require this storage overhead, we also analyze the tradeoff between communication cost and storage overhead in Section V-B. Our findings suggest that the use of Schemes P and V is preferred to the use of scheme T whenever the number of edits satisfies $d = o(\ell / \log \ell)$.

A. Fundamental Limits

We derive a lower bound on the communication cost between a user and a connected node and show that Schemes P and V are within a constant factor away from this lower bound under certain constraints on the number of edits. For this purpose, recall that a subsequence of a sequence is itself a sequence that can be obtained from the original sequence by deleting some elements without changing the order of the nonedited elements. Similarly, a supersequence of a sequence is itself a sequence that can be obtained from the original sequence by inserting some elements without changing the order of the nonedited elements.

Consider the following quantity that counts the number of possible subsequences or supersequences resulting from d edits on a data block of length ℓ, respectively, defined via

$$V(\ell, d) \triangleq \begin{cases} \max_{\mathbf{x}\in\mathbb{F}_q^\ell} |\{\tilde{\mathbf{x}}{:}\tilde{\mathbf{x}} \text{ results from at most } d \text{ deletions in } \mathbf{x}\}|, \\ \max_{\mathbf{x}\in\mathbb{F}_q^\ell} |\{\tilde{\mathbf{x}}{:}\tilde{\mathbf{x}} \text{ results from at most } d \text{ insertions/deletions in } \mathbf{x}\}|. \end{cases}$$

We require $\log V(\ell, d)$ bits to describe d deletions (or d insertions/deletions) to a node that contains the original sequence.

TABLE III
TRADE-OFF BETWEEN STORAGE OVERHEAD AND COMMUNICATION COMPLEXITY OF VARIOUS SYNCHRONIZATION PROTOCOLS.

	Scheme T	Scheme P	Scheme V
cost per edit	$(\ell-1)\log q$ (worst case)	$\log \ell + \log q + 1$	$\log \ell + \log q$
Storage Overhead per Edit	0	$\log \ell$	$\log \ell$
Applicability	—	—	$\ell \leq q$ and for deletions only
Average Size of Storage Nodes for Data Blocks of Length ℓ'	$\alpha\ell'$	$\alpha\ell$, where ℓ is the length of the original data blocks	$\alpha\ell'$

This shows that each user needs to communicate to a connected storage node at least $\log V(\ell, d)$ bits, given that the storage nodes contain only coded information about the original sequence and that the user knows the positions of the edits. A similar argument for establishing a lower bound for the two-way communication protocol was used in [30].

The fundamental combinatorial entity $V(\ell, d)$ was introduced by Levenshtein [31] and it is known (see [29, Eq. (11) and (24)]) that

$$\log V(\ell, d) \geq \begin{cases} \log \binom{\ell-d}{d}, & \text{for deletions only,} \\ \log (q-1)^d \binom{\ell+d}{d}, & \text{for deletions/insertions.} \end{cases}$$

When $d = o(\ell^{1-\epsilon})$ for a constant $0 < \epsilon < 1$, in the asymptotic parameter regime we have $\log \binom{\ell-d}{d} = \Omega(\log \ell)$ and $\log(q-1)^d \binom{\ell+d}{d} = \Omega(\log \ell + \log q)$. Therefore, Schemes P and V are within a constant factor away from this lower bound.

B. Communication Cost Versus Storage Overhead

Suppose that during each round of synchronization, a data block has a single edit. In this case, the communication cost between each user and each connected storage node for both Schemes P and V is approximately $\log \ell + \log q$. However, to achieve this cost, a user needs to store the matrix $\mathbf{A}^{(s)}$, which requires $\ell^2 \log q$ bits. However, this storage requirement may be easily reduced. For any synchronization scheme, we can first store the description of the initial matrices.[7] Subsequently, to generate the matrices $\mathbf{A}^{(s)}$, it suffices for the users to store the modifications to the initial matrices and we term this information the *storage overhead per edit*. For Schemes P and V, this overhead amounts to the space allocated for storing the locations of edits; hence, the storage overhead per edit is $\log \ell$ bits. In contrast, for Scheme T, the storage overhead per edit is zero as the matrices $\mathbf{A}^{(s)}$ are equal to the identity matrix.

We summarize the communication cost and storage overhead features of the various schemes in Table III.

Storage Overhead Versus Information Storage: Consider a single data block that has undergone d edits. Both Schemes P and V incur a total storage overhead of $d \log \ell$ bits for the user, with the data block itself being of size $\ell \log q$ bits. Therefore, for desirable storage allocation properties, one would require $d \log \ell = o(\ell)$ or, equivalently, the number of edits to be $o(\ell / \log \ell)$. This calculation implies that Schemes P and V should only be used in the small/moderate edit regime. It is also preferable to use Schemes P and V for semi-static and frequently updated data, respectively, since the former effectively maintains the length of the original data file, while the latter scales storage requirements according to the number of unedited symbols.

Hybrid Schemes: Suppose that one is given a constraint Γ on the storage overhead per edit, say $0 < \Gamma < \log \ell$. By combining Scheme V and Scheme T, we demonstrate a scheme that achieves a lower (worst case) communication cost while satisfying the aforementioned storage constraint.

Pick γ so that $\log(\gamma \ell) \leq \Gamma$. Next, divide each data block into two parts of lengths approximately $\gamma \ell$ and $(1-\gamma)\ell$. If the edit belongs to the first part, use intermediary coding via Scheme V and send out $\log \ell + \log q$ bits. The storage overhead of this protocol is $\log(\gamma \ell)$ bits. On the other hand, if the edit belongs to the second part, then send all $(1-\gamma)\ell \log q$ bits and the deleted position of $\log \ell$ bits. This protocol does not require any storage overhead. Hence, in the worst case, the hybrid scheme incurs a storage overhead of $\log(\gamma \ell)$ bits and a communication cost of $\max\{\log(\gamma \ell) + \log q, (1-\gamma)\ell \log q\}$ bits. The choice of γ that minimizes the communication cost is given by the next lemma.

Lemma 6: The hybrid scheme has smallest communication cost for

$$\gamma = \min \left\{ \frac{W(q^{\ell-1} \log q)}{\ell \log q}, \frac{2^{\Gamma}}{\ell} \right\}, \tag{10}$$

where $W(x)$ denotes the Lambert function, defined via $x = W(x) \exp(W(x))$.

Proof: The result easily follows by observing that $\max\{\log(\gamma \ell) + \log q, (1-\gamma)\ell \log q\}$ is minimized when $\log(\gamma \ell) + \log q = (1-\gamma)\ell \log q$. By rearranging terms, we arrive at the expression $z = y \exp y$, where $y = \gamma \, \ell \log q$ and $z = q^{\ell+1} \log q$. This yields a constraint satisfied by the Lambert function. The proof follows by observing that γ is also required to satisfy $\log(\gamma \, \ell) \leq \Gamma$. □

The hybrid scheme is described in more detail in Appendix C; Fig. 4 illustrates the communication cost and storage overhead trade-offs achievable by the hybrid scheme. Observe that both Schemes P and V approach the lower bound when one is allowed a storage overhead of $\log \ell$ bits, assuming single edits per data block. Note that the lower bound on the communication cost is independent on the storage overhead. It would hence be of interest to derive a lower bound that actually captures the dependency on the storage overhead.

We conclude this section by considering a system cost that takes into account *both* the communication complexity and storage overhead. Similar to the hybrid method, we request certain users to communicate via Scheme P/V and others via Scheme T. Specifically, we consider a probabilistic edit model, where one edit occurs in each round of communication and where the probability of data block s being edited equals p_s. Let U denote the set of users that synchronize via Scheme P/V, and define $p \triangleq \sum_{s \in U} p_s$. Then, the expected storage overhead is $p \log \ell$, while the expected communication cost equals $\log \ell + p \log q + (1-p)\ell \log q$. For some predefined $\theta \geq 0$, the aggregate cost is defined as

$$C_A(U) \triangleq \log \ell + p \log q + (1-p)\ell \log q + \theta p \log \ell. \tag{11}$$

[7] The initial matrix is the identity matrix for Scheme P and Scheme T, and a single Vandermonde matrix for Scheme V.

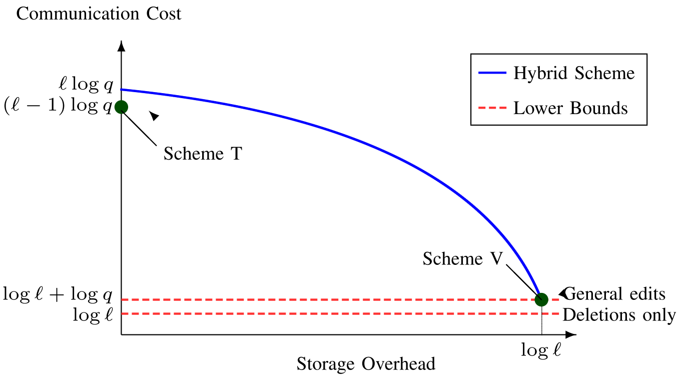


Fig. 4. Communication cost and storage overhead tradeoff.

Proposition 7: Let C_A be given by (11). *If $\theta \leq (\ell-1)\log q/\log \ell$, then C_A is minimized for $U=[k]$. Otherwise, if $\theta > (\ell-1)\log q/\log\ell$, then C_A is minimized for $U=\varnothing$.*

Proof: Consider $\theta \leq (\ell-1)\log q/\log\ell$. Suppose that $U \neq [k]$ and pick $s \notin U$. Let $U' = U \cup \{s\}$. Then

$$\begin{aligned} C_A(U') - C_A(U) &= p_s \log q - p_s \ell \log q + \theta p_s \log \ell \\ &= p_s(\theta \log \ell - (\ell-1)\log q) \leq 0. \end{aligned}$$

Hence, augmenting U with s lowers the aggregate cost and so C_A is minimized for $U=[k]$. The argument for the other case proceeds along similar lines. □

VI. Unknown Deletion Locations

In the previous sections, we focused on synchronization protocols for user-induced edits. In this case, the positions of deletions and the values of the corresponding symbols are known and available to the users of the DSS. Many applications, including Dropbox-like file synchronization, call for file updates where changes are made by secondary users, in which case the edits are unknown to primary or other file users. As will be shown next, it is straightforward to accommodate the update scenarios to this model, provided that an additional small storage overhead is allowed. Again, we focus on the single edit or low update frequency scenario, for which we show that one only needs to store the Varshamov-Tenengolts (VT) Syndrome [32] of the data, in addition to a properly encoded original file. When frequent checks or updates are performed, one may apply multiple-deletion correcting codes akin to those described in [33].

Consider the data string $\mathbf{x}^{(s)}$. Its VT syndrome is given by $\nu_1^{(s)} \triangleq \sum_{i=1}^{\ell} x_i^{(s)}$ and $\nu_2^{(s)} \triangleq \sum_{i=1}^{\ell-1} i\chi_i \mod \ell$, where χ_i is the indicator for the event $x_i^{(s)} \leq x_{i+1}^{(s)}$, $i \in [\ell-1]$.

Case I: All Data Blocks Experience a Single Deletion: In addition to the matrices $\mathbf{A}^{(s)}$, each user stores its VT syndrome $(\nu_1^{(s)}, \nu_2^{(s)})$ of size $\log q + \log \ell$ bits. Prior to an update, each user retrieves its VT syndrome to compute the deleted position and its value. With this information, the user proceeds with any of the previously outlined update schemes.

Case II: A Subset of Data Blocks Experience a Single Deletion: Observe that in the previous scheme, an additional overhead of $k(\log q + \log \ell)$ bits was required to store all VT syndromes of all k users. We show next that by using the structure of the MDS code, we can achieve storage savings by recording the VT syndromes of the $(n-k)$ check nodes only. For this purpose, assume that at most $n-k$ data blocks have a deletion and that $n-k<k$. Also, for simplicity, let $\ell = q$; q is a prime, and the syndrome is computed modulo q.

For the syndromes $\nu_1^{(1)}, \nu_1^{(2)}, \ldots, \nu_1^{(k)}$, each check node stores an additional check value that represents a linear combination of these syndromes, so that from any collection of k nodes all syndromes can be recovered. Similarly, each check node stores another check value from the syndromes $\nu_2^{(1)}, \nu_2^{(2)}, \ldots, \nu_2^{(k)}$. Therefore, with this scheme, we store an additional $2(n-k)\log q < k(\log q + \log \ell)$ bits. Suppose now that $n-k$ affected data blocks have a deletion. Each affected user requests the VT syndromes from the unaffected $k-(n-k)$ users and also the coded VT syndromes from the $n-k$ check nodes. With the k values, the affected user computes its own VT syndrome and consequently uses it to determine the position of its deletion and its value.

VII. Performance Analysis and Simulations

The analysis of Schemes P, V and RESET is performed for two already mentioned edit models that capture the combinatorial and probabilistic nature of edits:

1) PND: The probabilistic nonuniform deletion model. Here, we assume that each of the ℓ symbols of the k users is *independently* edited with probability p. This probability takes different values based on the application at hand. When deletions arise from deduplication, p typically ranges between 0.4 and 0.9. When deletions arise from user requests, they typically result in significantly smaller values of p, ranging from 0.01 to 0.1 [25], [34].
2) UD: The uniform deletions model. The underlying assumption is that each data block has $d = p\ell$ deletions, with the set of d coordinates chosen uniformly at random. This combinatorial model may be seen as an average case simplification of the PND model.

Suppose that C is the communication cost between a user and a storage node, with edits occurring in accordance with the PND and UD models. The following results, proved in Appendix D, establish the average communication costs of various synchronization protocols for a distributed system involving k users and n MDS array encoded storage nodes.

Theorem 8 (Scheme T): Consider Scheme T. Let C be the communication cost between a user and a storage node, with edits occurring in accordance with the PND and UD models. Then $E[C] = \eta(\ell)\,\ell \log q$, where

1) *for model UD, we have*

$$\eta(\ell)\,\ell = (\ell+1) - 2\sum_{i=1}^{\ell}\left(\frac{\binom{i}{p\ell}}{\binom{\ell}{p\ell}}\right)^k. \tag{12}$$

2) *for model PND, we have*

$$\eta(\ell)\,\ell = (\ell+1) - \frac{1-(1-p)^{k\ell}}{1-(1-p)^k}. \tag{13}$$

Proposition 9 (Scheme P/V): Consider either Scheme P or Scheme V. Define C to be the communication cost between one selected user and a connected storage node, with edits occurring according to the above models. Then under both models UD and PND,

$$E[C] = p\,\ell(\log \ell + \log q). \tag{14}$$

It is not difficult to see that for the RESET scheme, one has

$$E[C] = (1-p)\ell \log q, \tag{15}$$

for both the UD and PND model.

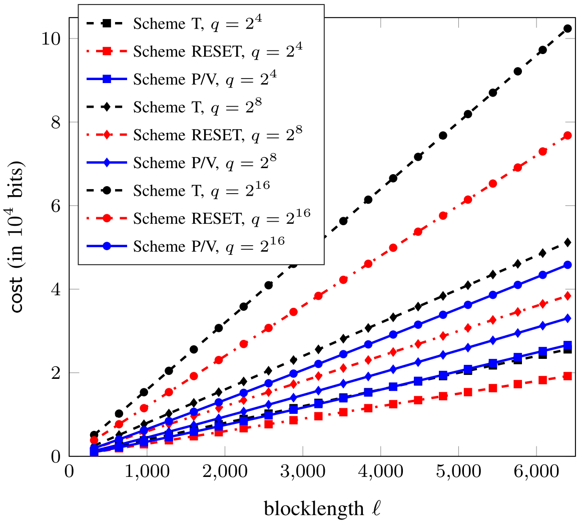


Fig. 5. Comparison of communication costs of schemes T (black), P/V (blue) and RESET (red) for a range of blocklengths ℓ and edit probability $p = 0.25$. Observe that under Scheme T, the performance of the UD and PND model almost coincide. The square, diamond, and circle symbols correspond respectively to field sizes $q = 2^4$, 2^8, and 2^{16}. The blocklengths are in bits.

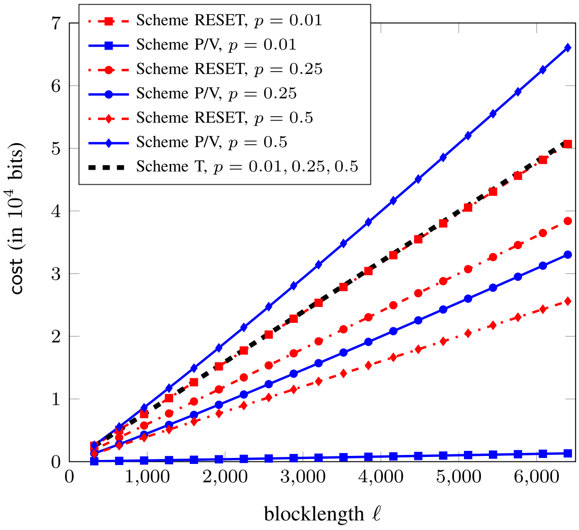


Fig. 6. Comparison of the communication cost cost of Schemes T (black), P/V (blue), and RESET (red) for a range of blocklengths ℓ and for different deletion probabilities $p = 0.01$, 0.25 and 0.5. The field size is fixed to $q = 2^8 = 256$. The performance of Scheme T changes negligibly as p varies, and is represented by a single line. The blocklengths are in bits.

We proceed next to numerically compare the communication cost savings of the proposed schemes for a practical system model. For this task, we adopt a setup similar to [25] and focus our attention on the well studied HYDRAstor DSS. Each storage node in a typical third generation HYDRAstor DSS is equipped with 12 1TB SATA disks, resulting in an overall storage capacity of 12TB. We assume that the atomic storage units are the disks themselves; furthermore, we restrict our attention to $[12, 9, \ell]$ MDS array codes over the finite fields $\mathbb{F}_q$, $q \in \{16, 256, 65536\}$ [35] that are used by typical HYDRAstor systems to protect information against disk failures. We assume block sizes up to 7.1 KB, as data sections of size 64 KB are divided into 9 fragments. Each disk stores roughly 155 million data blocks.

In Fig. 5, we plot the average communication costs of schemes T, P and RESET for a fixed value of the deduplication rate $p = 0.25$ and for different block and field sizes. The results indicate that in the high deletion scenario, RESET outperforms all other schemes only for small field sizes, but the P/V Schemes offer more than twofold improvements over other methods when the field size is large.

Fig. 6 shows the average communication costs of Schemes T, P and RESET for a fixed field size $q = 2^8$ and for different edit and deduplication probabilities. The chosen deletion probabilities cover both the operational point where one expects the communication cost savings to outweigh the storage overhead (i.e., $p = 0.01$) and where the deduplication rate is very high (i.e., $p = 0.25, 0.5$) [25]. As expected, as p increases, the communication cost increases for Scheme P/V, while it decreases for Scheme RESET.

VIII. Conclusion

We introduced the problem of synchronizing of deduplicated and coded information and proposed a collection of efficient protocols for reconciling distributed coded data. The gist of the proposed solution is intermediary encoding, which allows the edited encoded information to retain repair and reconstruction properties, although with a different encoding functionality. Hence, the synchronization protocols may be seen as a form of functional reconciliation methods – methods that preserve functional properties but not a particular code structure. For the presented protocols, we derived simple fundamental lower and average case performance bounds and simulated the protocols on a HYDRAstor-based model.

Appendix A
Proof of Proposition 1

To prove the claimed result, we adapt the *fooling set method* from communication complexity (see [26, § 13.2.1]), standardly used to lower bound the deterministic communication complexity of a function.

Assume that $u_\ell^{(1)}$ and $u_1^{(2)}$ are the deleted coordinates, i.e., the first user deleted the last coordinate, while the second user deleted the first coordinate. Define $f : \mathbb{F}_q^\ell \times \mathbb{F}_q^\ell \to \mathbb{F}_q^{\ell-1}$ to be a function such that $f(\mathbf{u}^{(1)}, \mathbf{u}^{(2)}) = \left(u_1^{(1)} + u_2^{(2)}, u_2^{(1)} + u_3^{(2)}, \ldots, u_{\ell-1}^{(1)} + u_\ell^{(2)}\right)$. Then the task of node 3 is to update its value to $f\left(\mathbf{u}^{(1)}, \mathbf{u}^{(2)}\right)$.

A *fooling set* for f of size M is a subset $S \subseteq \mathbb{F}_q^\ell \times \mathbb{F}_q^\ell$ and and a value $\mathbf{c} \in \mathbb{F}_q^{\ell-1}$ such that (a) for every $\left(\mathbf{u}^{(1)}, \mathbf{u}^{(2)}\right) \in S$, $f\left(\mathbf{u}^{(1)}, \mathbf{u}^{(2)}\right) = \mathbf{c}$ and (b) for every distinct $\left(\mathbf{u}^{(1)}, \mathbf{u}^{(2)}\right), \left(\mathbf{v}^{(1)}, \mathbf{v}^{(2)}\right) \in S$, either $f(\mathbf{u}^{(1)}, \mathbf{v}^{(2)}) \neq \mathbf{c}$ or $f(\mathbf{v}^{(1)}, \mathbf{u}^{(2)}) \neq \mathbf{c}$. One can show that if f has a fooling set of size M, then the total deterministic communication cost for any protocol computing f is at least $\log M$ [26, Lemma 13.5].

To prove the claim, we exhibit next a fooling set of size $q^{\ell-1}$ for the function of interest. Set $\mathbf{c} = \mathbf{0}$ and consider the subset of $\mathbb{F}_q^\ell \times \mathbb{F}_q^\ell$ of size $q^{\ell-1}$ defined as

$$S = \Big\{ \left(\mathbf{u}^{(1)}, \mathbf{u}^{(2)}\right) : \mathbf{u}^{(1)} = \left(0, u_2^{(1)}, u_3^{(1)}, \ldots, u_\ell^{(1)}\right), \quad \mathbf{u}^{(2)} = \left(-u_2^{(1)}, -u_3^{(1)}, \ldots, -u_\ell^{(1)}, 0\right) \Big\}.$$

Then $f\left(\mathbf{u}^{(1)}, \mathbf{u}^{(2)}\right) = \mathbf{0}$ for all $\left(\mathbf{u}^{(1)}, \mathbf{u}^{(2)}\right) \in S$. Furthermore, if $\mathbf{u}^{(1)} \neq \mathbf{v}^{(1)}$, or equivalently, if $\left(0, u_2^{(1)}, u_3^{(1)}, \ldots, u_\ell^{(1)}\right) \neq \left(0, v_2^{(1)}, v_3^{(1)}, \ldots, v_\ell^{(1)}\right)$, we can check that $f\left(\mathbf{u}^{(1)}, \mathbf{v}^{(2)}\right) = \left(u_2^{(1)} - v_2^{(1)}, u_3^{(1)} - v_3^{(1)}, \ldots, u_\ell^{(1)} - v_\ell^{(1)}\right) \neq \mathbf{0}$. Therefore, S is a fooling set of size $q^{\ell-1}$.

Appendix B
Modifications of the Intermediary Encoding Function

We describe next simple modifications of (1a) that allow for systematic MDS codes and for variable data block lengths.

(a) *Systematic MDS*. A *systematic* MDS array code is a MDS array code with the property that the k data blocks are explicitly stored amongst a set of k nodes, termed the systematic nodes. In other words, the content of the systematic nodes are the blocks $\mathbf{x}^{(s)}$, $s \in [k]$.

Assume a systematic MDS code with functionalities $\textsc{Ec}$ and $\textsc{Rc}$. As before, let $\mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \ldots, \mathbf{A}^{(k)}$ be invertible $\ell \times \ell$ matrices and define a systematic encoding function $\textsc{Ec}^{*\ell}\left(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(k)}\right)$ according to

$$\textsc{Ec}^{*\ell}\left(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(k)}\right)\Big|_{\{t\}\times[\ell]} = \begin{cases} \textsc{Ec}^{\ell}\left(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(k)}\right)\big|_{\{t\}\times[\ell]}, & \text{if } t \text{ is systematic,} \\ \textsc{Ec}^{\ell}\left(\mathbf{x}^{(1)}\mathbf{A}^{(1)}, \ldots, \mathbf{x}^{(k)}\mathbf{A}^{(k)}\right)\big|_{\{t\}\times[\ell]}, & \text{otherwise.} \end{cases}$$

Let $\mathbf{C}$ be the resulting information stored over the n nodes. The reconstruction algorithm may be used unaltered provided that some pre-processing of $\mathbf{C}|_{T\times[\ell]}$ is performed first. Specifically, suppose that $t \in T$ and t is a systematic node with $\mathbf{C}|_{\{t\}\times[\ell]} = \mathbf{x}^{(s)}$ for some $s \in [k]$. We modify this information to $\mathbf{x}^{(s)}\mathbf{A}^{(s)}$. If $\mathbf{C}'$ denotes the array resulting from the previous computation, reconstruction is performed according to $\textsc{Rc}_T^{*\ell}(\mathbf{C}')$.

(b) *Data blocks of variable lengths*. For $s \in [k]$, let $\mathbf{x}^{(s)}$ be of length ℓ_s and let $\mathbf{A}^{(s)}$ be a *right* invertible $\ell_s \times \ell$ matrix, where $\ell = \max_{s\in[k]} \ell_s$. We define $\textsc{Ec}^{*\ell} : \mathbb{F}_q^{\ell_1} \times \mathbb{F}_q^{\ell_2} \times \cdots \times \mathbb{F}_q^{\ell_k} \to \mathbb{F}_q^{n\times\ell}$ via (1a). If we perform reconstruction via (2a), we arrive at an $[n, k; \ell]$ MDS array code. Note that the right inverse of $\mathbf{A}^{(s)}$ is required for reconstruction.

Appendix C
Hybrid Schemes

We describe in detail the hybrid scheme that combines Schemes P and V. Define $\ell^* = (1-\gamma)\ell$. Then, for $s \in [k]$, initialize

$$\mathbf{A}^{(s)} \leftarrow \begin{pmatrix} \mathbf{V} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{pmatrix},$$

where $\mathbf{V}$ is a Vandermonde matrix of dimension $(\ell - \ell^*) \times (\ell - \ell^*)$ and $\mathbf{I}$ is an identity matrix of dimension ℓ^*. For subsequent edits, we proceed as outlined in Algorithm Scheme H.

Proposition 10 (Scheme H): Consider an $[n, k; \ell]$ MDS array code and assume a single deletion per user. The updates in accordance to Scheme H result in an $[n, k; \ell - 1]$ MDS array code. In the worst case, each user introduces a storage overhead of $\max\{\log(\gamma\ell), 0\}$ *bits and incurs a communication cost of* $\max\{\log(\gamma\ell) + \log q, (1-\gamma)\ell\log q\}$ *bits for its edit.*

The proof is similar to the proofs of Proposition 4 and Proposition 5 and is therefore omitted.

Scheme H: Assumption: $\mathbf{x}^{(s)}$ is deleted at coordinate i_s.

```
if $i_s \leq \ell - \ell^*$ then
    User $s$ sends to all connected storage nodes $N(s)$ the value of the deleted symbol – $x_{i_s}^{(s)}$ ($\log q$ bits) and the coordinate $i_s$ ($\log \ell$ bits)
    $\mathbf{A}^{(s)} \leftarrow \mathbf{A}^{(s)}$ with the $i_s$th row removed
else
    User $s$ computes the vector
$\mathbf{d} \leftarrow \left(x_{\ell-\ell^*+1}^{(s)}, x_{\ell-\ell^*+2}^{(s)}, \ldots, x_\ell^{(s)}\right) - \left(x_{\ell-\ell^*+1}^{(s)}, \ldots, x_{i_s-1}^{(s)}, x_{i_s+1}^{(s)}, \ldots, x_\ell^{(s)}, 0\right).$
    User $s$ sends to all connected storage nodes $N(s)$ the vector $\mathbf{d}$ ($\ell^* \log q$ bits)
    User $s$ pads $\mathbf{x}^{(s)}$ with a zero.
end
for $t \in N(s)$ do
    Subtract $\mathbf{D}|_{\{t\}\times[\ell]}$ where
$\mathbf{D} = \begin{cases} \textsc{Ec}^{\ell}\left(\mathbf{0}, \ldots, x_{i_s}^{(s)}\mathbf{A}|_{\{i_s\}\times[\ell]}, \ldots, \mathbf{0}\right), & \text{if } i_s \leq \ell - \ell^*, \\ \textsc{Ec}^{\ell}\left(\mathbf{0}, \ldots, (\mathbf{0}, \mathbf{d}), \ldots, \mathbf{0}\right), & \text{otherwise.} \end{cases}$
end
```

Appendix D
Probabilistic Analysis of the Communication Cost

In what follows, we derive the formulae for the expected communication costs given in Section VII and estimate these values in the asymptotic regime. In particular, we show that even on average, Scheme T requires communicating $\Omega(\ell \log q)$ bits per one single round of editing.

We first prove the results (12), (13) and (14) of Section VII.

For Scheme T, recall the definitions of $i_{\max}$, $i_{\min}$ and I from Section III. For the model UD, we have $|I| = i_{\max} - i_{\min}$, while $|I| = \ell + 1 - i_{\min}$ holds for model PND. Hence, $\eta(\ell)\ell = E[|I|] = E[i_{\max}] - E[i_{\min}] = (\ell+1) - 2E[i_{\min}]$ for model UD, and $\eta(\ell)\ell = (\ell+1) - E[i_{\min}]$ for model PND. Therefore, our problem reduces to estimating $E[i_{\min}]$ for the various schemes under consideration. To do so, we follow the standard derivation methods in order statistics (see David and Nagaraja [36]).

Recall that $E[i_{\min}] = \sum_{i=1}^{\ell} \text{Prob}(i_{\min} \geq i)$ and observe that

$$\text{Prob}(i_{\min} \geq i) = \begin{cases} \left(\frac{\binom{\ell-i+1}{p\ell}}{\binom{\ell}{p\ell}}\right)^k, & \text{for model UD,} \\ (1-p)^{k(i-1)}, & \text{for model PND.} \end{cases}$$

This establishes (12) and (13).

On the other hand, the communication cost between a user and a connected storage node for Schemes P/V is given by $d(\log q + \log \ell)$, where d denotes the corresponding number of edits of the user. Since for the models UD and PND the expected number of edits is $p\ell$, (14) follows after some straightforward algebraic manipulations.

Let C_T and C_P be the expected communication cost of Scheme T and Scheme P/V, respectively. In the following analysis, asymptotics are computed in ℓ and we assume that q is either fixed or that $q = O(\ell)$. Furthermore, we assume that $p \geq 1/\ell$ and that $p = o(1/\log \ell)$ or $p \log \ell \to 0$ as $\ell \to \infty$. Under these assumptions, we have $\lim_{\ell\to\infty} C_P/C_T = 0$.

1) For model UD, note that $\binom{i}{p\ell}/\binom{\ell}{p\ell} \leq i/\ell$ for all $i \leq \ell$ when $p\ell \geq 1$. Hence, $E[i_{\min}]/\ell \leq \sum_{i=1}^{\ell} i^k/\ell^{k+1}$ and so, $\lim_{\ell\to\infty} E[i_{\min}]/\ell \leq 1/(k+1)$, or
$$\lim_{\ell\to\infty} \eta(\ell) \geq \frac{k-1}{k+1}.$$
So, $C_T = \Omega(\ell \log q)$, while $C_P = p\ell(\log \ell + \log q)$. Hence, $\lim_{\ell\to\infty} C_P/C_T = 0$ if $p = o(1/\log \ell)$.

2) For model PND, note that $(1-p)^k \leq (1-1/\ell)$ when $p\ell \geq 1$ and $k \geq 1$. Hence, $E[i_{\min}]/\ell \leq 1-(1-1/\ell)^{\ell}$ and so, $\lim_{\ell\to\infty} E[i_{\min}]/\ell \leq 1-1/e$, or
$$\lim_{\ell\to\infty} \eta(\ell) \geq \frac{1}{e}.$$
Again, $C_T = \Omega(\ell \log q)$, while $C_P = p\ell(\log \ell + \log q)$. Hence, $\lim_{\ell\to\infty} C_P/C_T = 0$ if $p = o(1/\log \ell)$.

Consequently, for both probabilistic models and our proposed protocols, we achieve a communication complexity that is asymptotically negligible compared to that of Scheme T.

Appendix E
Extensions for General Codes

We demonstrate in what follows that our schemes are applicable to general DSSs. For this purpose, consider an information vector $\left(x^{(1)}, x^{(2)}, \ldots, x^{(k)}\right) \in \mathbb{F}_q^k$ to be stored in a DSS. We make use of the following definitions.

1) An *encoding* function is a map $\textsc{Ec} : \mathbb{F}_q^k \to \mathbb{F}_q^{n\times\alpha}$.

2) For any κ-subset T of $[n]$, a map $\textsc{Rc}_T : \mathbb{F}_q^{\kappa\times\alpha} \to \mathbb{F}_q^k$ is termed a *reconstruction* function if for $\left(x^{(1)}, x^{(2)}, \ldots, x^{(k)}\right) \in \mathbb{F}_q^k$,
$$\textsc{Rc}_T\left(\textsc{Ec}\left(x^{(1)}, \ldots, x^{(k)}\right)\Big|_{T\times[\alpha]}\right) = \left(x^{(1)}, \ldots, x^{(k)}\right).$$

3) Given a $t \in [n]$ and r-subset T of $[n] \setminus \{t\}$, a map $\textsc{Rp}(t,T) : \mathbb{F}_q^{r\times\alpha} \to \mathbb{F}_q^{\alpha}$ 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.

Depending on the set of subsets T over which a repair function is defined, one recovers various families of codes:

1) *Maximum distance separable (MDS) codes* [28, Ch. 11]. Here, we have $k = \kappa = r$ and $\alpha = 1$, and the reconstruction and repair algorithms coincide.

2) *Regenerating codes* [6]. Here, $r \geq \kappa$, and we require a repair function $\textsc{Rp}_{t,T}$ for all $t \in [n]$ and all r-subsets T of $[n] \setminus \{t\}$. We note that in bandwidth-limited DSSs, it is of importance to define an additional function, mapping words from $\mathbb{F}_q^{r\times\alpha}$ to $\mathbb{F}_q^{\alpha}$. More precisely, a repair algorithm $\textsc{Rp}_{t,T}$ maps inputs from $\mathbb{F}_q^{r\times\alpha}$ to $\mathbb{F}_q^{r\times\beta}$ and then to $\mathbb{F}_q^{\alpha}$, where β is usually smaller than α and indicates the required amount of downloaded bits. The intermediary function does not change our analysis.

3) *Locally Repairable Codes (LRC)* [8]. Here, $r < \kappa$, and for all $t \in [n]$, we only require a repair function $\textsc{Rp}_{t,T}$ for *some* r-subset T of $[n] \setminus \{t\}$. Efficient repair is achieved by minimizing the number of nodes that needs to be contacted.

We refer to the three aforementioned families of encodings for DSSs as $(n, \kappa, r, \alpha, k)$ DSS codes, and focus on code maps that are linear. As in Section II-B, we assume k users and associate each user with a data block of length ℓ. Also, as before, we can extend an $(n, \kappa, r, \alpha, k)$ DSS code to an $(n, \kappa, r, \alpha\ell, k\ell)$ DSS code. In this $(n, \kappa, r, \alpha\ell, k\ell)$ DSS code, we may regard the given information as a k-tuple of *data blocks*, each of length ℓ, written as $\left(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(k)}\right)$.

The traditional encoding function $\textsc{Ec}^{\ell} : \mathbb{F}_q^{k\times\ell} \to \mathbb{F}_q^{n\times\alpha\times\ell}$ is defined in such a way that for $i \in [\ell]$,
$$\textsc{Ec}^{\ell}\left(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(k)}\right)\Big|_{[n]\times[\alpha]\times\{i\}} = \textsc{Ec}\left(x_i^{(1)}, \ldots, x_i^{(k)}\right). \tag{16}$$

Hence, the information stored at the n nodes represents an $n \times \alpha \times \ell$ tensor. The corresponding reconstruction and repair functions are similar to the functions $\textsc{Rc}_T^{\ell} : \mathbb{F}_q^{\kappa\times\alpha\times\ell} \to \mathbb{F}_q^{k\times\ell}$ for κ-subsets T of $[n]$; and $\textsc{Rp}_{t,T}^{\ell} : \mathbb{F}_q^{r\times\alpha\times\ell} \to \mathbb{F}_q^{\alpha\times\ell}$ for $t \in [n]$ and r-subsets T of $[n] \setminus \{t\}$, respectively, defined such that for $i \in [\ell]$,
$$\textsc{Rc}_T^{\ell}\left(\mathbf{C}|_{T\times[\alpha]\times[\ell]}\right)\Big|_{[k]\times\{i\}} \triangleq \textsc{Rc}_T\left(\mathbf{C}|_{T\times[\alpha]\times\{i\}}\right), \tag{17}$$
$$\textsc{Rp}_{t,T}^{\ell}\left(\mathbf{C}|_{T\times[\alpha]\times[\ell]}\right)\Big|_{[\alpha]\times\{i\}} \triangleq \textsc{Rp}_{t,T}\left(\mathbf{C}|_{T\times[\alpha]\times\{i\}}\right), \tag{18}$$
where $\mathbf{C}$, for brevity, is the encoded data $\textsc{Ec}^{\ell}\left(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(k)}\right)$ stored in the DSS. Using the above traditional encoding functions as building blocks, we derive an intermediary encoding scheme for general DSSs codes.

Proposition 11: Consider an $(n, \kappa, r, \alpha, k)$ DSS. For any integer ℓ and k invertible $\ell \times \ell$ matrices $\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(k)}$, the function $\textsc{Ec}^{*}$ *given by* (1a) *describe an $(n, \kappa, r, \alpha\ell, k\ell)$ DSS code.*

Given the above intermediary encoding scheme for general DSSs, we may adapt Schemes P and V, where we simply regard the net change matrix $\mathbf{D}$ as a three-dimensional tensor.

Acknowledgment

The authors are grateful to Prof. D. Goeckel and the anonymous reviewers for their insightful and helpful suggestions that greatly improved the presentation of the paper.

References

[1] S. El Rouayheb, S. Goparaju, H. M. Kiah, and O. Milenkovic, "Synchronizing edits in distributed storage networks," in *Proc. IEEE Int. Symp. Inf. Theory*, Hong Kong, 2015, pp. 1475–1479.

[2] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop distributed file system," in *Proc. IEEE MSST*, 2010, pp. 1–10.

[3] C. Huang *et al.*, "Erasure coding in Windows Azure storage," in *Proc. USENIX Annu. Tech. Conf.*, 2012, pp. 15–26.

[4] F. Chang *et al.*, "Bigtable: A distributed storage system for structured data," *ACM Trans. Comput. Syst.*, vol. 26, no. 2, p. 4, 2008.

[5] C. Vecchiola, S. Pandey, and R. Buyya, "High-performance cloud computing: A view of scientific applications," in *Proc. 10th ISPAN*, 2009, pp. 4–16.

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

[7] 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, Comput.*, 2009, pp. 1243–1249.

[8] P. Gopalan, C. Huang, H. Simitci, and S. Yekhanin, "On the locality of codeword symbols," *IEEE Trans. Inf. Theory*, vol. 58, no. 11, pp. 6925–6934, Nov. 2012.

[9] P. Gopalan, C. Huang, B. Jenkins, and S. Yekhanin, "Explicit maximally recoverable codes with locality," in *Proc. ECCC*, 2013, vol. 20, no. 104.

[10] K. A. S. Immink*, Codes for Mass Data Storage Systems*. Eindhoven, The Netherlands: Shannon Found., 2004.

[11] I. Drago *et al.*, "Inside Dropbox: Understanding personal cloud storage services," in *Proc. ACM Internet Meas. Conf.*, 2012, pp. 481–494.

[12] Y. Zhang, Y. Wu, and G. Yang, "Droplet: A distributed solution of data deduplication," in *Proc. 13th ACM/IEEE Int. Conf. Grid Comput.*, 2012, pp. 114–121.

[13] D. Meister and A. Brinkmann, "Multi-level comparison of data deduplication in a backup scenario," in *Proc. SYSTOR*, 2009, no. 8.

[14] A. Tridgell *et al.*, "The rsync algorithm," 1996.

[15] T. Knauth and C. Fetzer, "Dsync: Efficient block-wise synchronization of multi-gigabyte binary data," in *Proc. LISA*, 2013, pp. 45–58.

[16] C. Phipps, "zsync—Optimised rsync over HTTP," 2005.

[17] A. Orlitsky and K. Viswanathan, "One-way communication and error-correcting codes," *IEEE Trans. Inf. Theory*, vol. 49, no. 7, pp. 1781–1788, Jul. 2003.

[18] Y. Minsky, A. Trachtenberg, and R. Zippel, "Set reconciliation with nearly optimal communication complexity," *IEEE Trans. Inf. Theory*, vol. 49, no. 9, pp. 2213–2218, Sep. 2003.

[19] 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, Comput.*, 2010, pp. 1412–1419.

[20] N. Ma, K. Ramchandran, and D. Tse, "Efficient file synchronization: A distributed source coding approach," in *Proc. IEEE Int. Symp. Inf. Theory*, 2011, pp. 583–587.

[21] S. S. T. Yazdi and L. Dolecek, "Synchronization from deletions through interactive communication," in *Proc. 7th ISTC*, 2012, pp. 66–70.

[22] S. Tabatabaei Yazdi and L. Dolecek, "A deterministic, polynomial-time protocol for synchronizing from deletions," *IEEE Trans. Inf. Theory*, vol. 60, no. 1, pp. 397–409, Jan. 2014.

[23] A. S. Rawat, S. Vishwanath, A. Bhowmick, and E. Soljanin, "Update efficient codes for distributed storage," in *Proc. IEEE Int. Symp. Inf. Theory*, 2011, pp. 1457–1461.

[24] A. Mazumdar, V. Chandar, and G. W. Wornell, "Update-efficiency and local repairability limits for capacity approaching codes," *IEEE J. Sel. Areas Commun.*, vol. 32, no. 5, pp. 976–988, May 2014.

[25] P. Strzelczak *et al.*, "Concurrent deletion in a distributed content-addressable storage system with global deduplication," in *Proc. FAST*, 2013, pp. 161–174.

[26] S. Arora and B. Barak*, Computational Complexity: A Modern Approach*, 1st ed. New York, NY, USA: Cambridge Univ.Press, 2009.

[27] M. Blaum, J. Bruck, and A. Vardy, "MDS array codes with independent parity symbols," *IEEE Trans. Inf. Theory*, vol. 42, no. 2, pp. 529–542, Feb. 1996.

[28] F. J. MacWilliams and N. J. A. Sloane*, The Theory of Error-Correcting Codes*. Amsterdam, The Netherlands: Elsevier, 1977, vol. 16.

[29] V. I. Levenshtein, "Efficient reconstruction of sequences from their subsequences or supersequences," *J. Combinat. Theory, Ser. A*, vol. 93, no. 2, pp. 310–332, 2001.

[30] A. Orlitsky, "Communication issues in distributed computing," Ph.D. dissertation, Stanford University, Stanford, CA, USA, 1987.

[31] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions and reversals," *Soviet Phys. Doklady*, vol. 10, p. 707, 1966.

[32] N. J. Sloane, "On single-deletion-correcting codes," in *Codes and Designs*. Berlin, Germany: de Gruyter, 2002, pp. 273–291.

[33] M. C. Davey and D. J. MacKay, "Reliable communication over channels with insertions, deletions, and substitutions," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 687–698, Feb. 2001.

[34] M. Fu *et al.*, "Design tradeoffs for data deduplication performance in backup workloads," in *Proc. 13th USENIX Conf. File Storage Technol.*, 2015, pp. 331–344.

[35] J. Luo, K. D. Bowers, A. Oprea, and L. Xu, "Efficient software implementations of large finite fields $GF(2^n)$ for secure storage applications," *ACM Trans. Storage*, vol. 8, no. 1, 2012, Art. no. 2.

[36] H. A. David and H. N. Nagaraja*, Order Statistics*. New York, NY, USA: Wiley, 1970.

Salim El Rouayheb (S'07–M'09) received the Diploma degree in electrical engineering from the Lebanese University, Faculty of Engineering, Roumieh, Lebanon, in 2002, and the M.S. degree in computer and communications engineering from the American University of Beirut, Lebanon, in 2004. He received the Ph.D. degree in electrical engineering from Texas A&M University, College Station, in 2009. He was a Postdoctoral Research Fellow at UC Berkeley (2010–2011) and a Research Scholar at Princeton University (2012–2013).

He is currently an Assistant Professor in the ECE Department at the Illinois Institute of Technology, Chicago. His research interests are in the broad area of information theory and coding theory with a focus on network coding, coding for distributed storage and information theoretic security.

Sreechakra Goparaju is a postdoctoral employee in the California Institute for Telecommunications and Information Technology (Calit2) at the University of California, San Diego. He received his Ph.D. in electrical engineering at Princeton University, where he joined as a Gordon Wu Fellow from 2008. He received a B.Tech. and an M.Tech. (dual degree) in electronics and electrical communication engineering from Indian Institute of Technology, Kharagpur. His research interests are information theory, coding on storage, and combinatorics.

Han Mao Kiah received his Ph.D. degree in mathematics from the Nanyang Technological University, Singapore in 2014. From 2014 to 2015, he was a Postdoctoral Research Associate at the Coordinated Science Laboratory, University of Illinois at Urbana-Champaign. Currently, he is a Lecturer at the School of Physical and Mathematical Sciences, Nanyang Technological University, Singapore. His research interests include combinatorial design theory, coding theory, and enumerative combinatorics.

Olgica Milenkovic (M'04–SM'12) received her Ph.D. in electrical engineering from the University of Michigan, Ann Arbor and is currently a Professor in the Electrical Engineering Department of University of Illinois. She is a recipient of the NSF Career Award, the DARPA Young Faculty Award, and the Dean's Excellence in Research Award. In 2012 she was named a Center for Advanced Studies (CAS) Associate, and in 2013 she became a Willet scholar. In 2015, she was named Distinguished Lecturer of the IEEE Information Theory Society. She served on the Editorial Board for the IEEE Transactions on Communications, the IEEE Transactions on Signal Processing and the IEEE Transactions on Information Theory. Her research interests are in bioinformatics, coding theory, compressive sensing and social sciences.