Subspace Synchronization: A Network-Coding Approach to Object Reconciliation

Vitaly Skachek
Institute of Computer Science
University of Tartu, Estonia
vitaly.skachek@ut.ee

Michael G. Rabbat
Department of Electrical and Computer Engineering
McGill University, Montréal, Canada
michael.rabbat@mcgill.ca

***Abstract*—Assume that two users possess two different subspaces of an ambient linear space. We show that the problem of synchronization of such vector spaces can be easily solved by an efficient algorithm. By building on this observation, we propose an algorithm for synchronization of two collections of binary files of length n each, stored in the cloud in a distributed manner. By further employing techniques akin to network coding, we propose a more efficient file synchronization algorithm that has communication complexity $O(d \cdot n)$ bits and computational complexity $O(k^2 \cdot n)$ operations, where k is the total number of files and d is the number of files that differ. The algorithm successfully reconciles two sets of files in 3 communication rounds with high probability.**

***Index Terms*—Distributed storage, cloud storage, file reconciliation, network coding, subspace codes.**

I. Introduction

As cloud storage gradually becomes more and more popular, there is a need for efficient supporting algorithms. One particular problem arises when multiple copies of a collection of files are stored on several devices, in a distributed manner. Each file may be stored on only some of the servers, and it might be necessary to reconcile the data stored in different servers such that, at the end of the process, each server has all the files in the collection.

A simple approach to this problem is to have tables or lists of files stored at each server. In some cases, however, this approach might not work well, as the tables can be outdated. For example, in applications such as Dropbox, the user may access the same distributed storage system from several devices, while each individual device may only be intermittently connected to the internet.

Let $\mathcal{A}$ be a reconciliation algorithm. In the analysis of $\mathcal{A}$, we follow the framework that was defined in [8]. Important parameters in reconciliation are communication cost $\textsc{Communication}(\mathcal{A})$ (the worst case number of bits sent between the devices), computational complexity $\textsc{Computation}(\mathcal{A})$ (the worst case number of operations performed at each device) and time $\textsc{Time}(\mathcal{A})$ (the length of the largest chain of messages in the communication protocol). Here, following [8], a chain of messages $\boldsymbol{m}_1, \boldsymbol{m}_2, \cdots, \boldsymbol{m}_\ell$ has length ℓ, if the reception of the message $\boldsymbol{m}_i$ necessarily precedes the sending of $\boldsymbol{m}_{i+1}$.

A number of algorithms for efficient reconciliation of files have been proposed. These algorithms typically consider only two devices A and B, which are connected through an error-free channel. An approach in [6] uses interpolation of characteristic polynomials akin to Reed-Solomon codes. The algorithm proposed in [6] has $\textsc{Communication}(\mathcal{A}) = O(d \log u)$, $\textsc{Computation}(\mathcal{A}) = O(d^3)$ and $\textsc{Time}(\mathcal{A}) = O(\log k)$ with high probability. Here k is the total number of objects in possession of A and B, d is the number of objects possessed by only one user, and u is the size of the space where the objects are taken from.

Another reconciliation algorithm, based on invertible Bloom filters, was recently proposed in [2], [3]. That algorithm has parameters $\textsc{Communication}(\mathcal{A}) = O(d \log u)$, $\textsc{Computation}(\mathcal{A}) = O(d)$ and $\textsc{Time}(\mathcal{A}) = 3$ (with high probability). Another recent algorithm performing set reconciliation with high probability by using *Biff codes* was proposed in [7]. That algorithm has parameters $\textsc{Communication}(\mathcal{A}) = O(d \log u)$, $\textsc{Computation}(\mathcal{A}) = O(k \log u)$ and $\textsc{Time}(\mathcal{A}) = 3$.

In this work, we build on the ideas from the area of network coding [1], [5]. In particular, we use the idea that information can be represented by vector spaces instead of vectors [4], [9]. For the case of two users, we first show in Section III that the problem of subspace syncronization can be easily solved by an efficient algorithm. Then, in Section IV, we extend that approach and propose a new, yet not practical, algorithm for synchronization of two collections of files by using a mapping based on error-correcting codes. In Section V, we further employ ideas from network coding to propose a new file synchronization algorithm. That algorithm has communication complexity $\textsc{Communication}(\mathcal{A}) = O(d \cdot \log u)$ bits, computational complexity $\textsc{Computation}(\mathcal{A}) = O(k^2 \cdot \log u)$ and number of rounds $\textsc{Time}(\mathcal{A}) = 3$. Finally, we discuss existence of suitable hash functions for the use in the above algorithm in Section VI.

[1]The work of V. Skachek is supported in part by the research grants PUT405 and IUT2-1 from the Estonian Research Council, by the EU COST Action IC1104 on random network coding and designs over $\mathbb{F}_q$, and by the European Regional Development Fund through the Estonian Center of Excellence in Computer Science, EXCS.

[2]The work of M. Rabbat is supported in part by the Natural Science and Engineering Research Council of Canada.

II. Notation and Problem Setup

Let ℓ be a positive integer, and let $\boldsymbol{v}_1, \boldsymbol{v}_2, \cdots, \boldsymbol{v}_\ell$ be vectors over the finite field $\mathbb{F}$. Denote their span by $\langle \boldsymbol{v}_1, \boldsymbol{v}_2, \cdots, \boldsymbol{v}_\ell \rangle$. Denote by $\mathbf{0}$ the vector of zeros, whose length will be clear from the context, and by $\boldsymbol{I}_s$ the $s \times s$ identity matrix.

Let W be an ambient space, and let $U, V \subseteq W$. Denote by $U + V$ the subspace sum of the subspaces U and V,

$$U + V = \{\boldsymbol{u} + \boldsymbol{v} \; : \; \boldsymbol{u} \in U, \; \boldsymbol{v} \in V\} \; .$$

If $U \cap V = \{\mathbf{0}\}$, then $U + V$ is a direct sum, which we denote by $U \oplus V$. The definition of the vector sum naturally generalizes to the sum of more than two subspaces.

Denote by $\mathsf{dist}(U, V)$ the subspace distance metric between subspaces, defined as

$$\begin{aligned} \mathsf{dist}(U, V) &= \dim(U) + \dim(V) - 2\dim(U \cap V) \\ &= 2\dim(U \cup V) - \dim(U) - \dim(V) \; . \end{aligned}$$

(See [4] for details.) The distance between the subspaces being synchronized plays a key role in the complexity of the algorithm described below.

Let $\mathcal{G} = (\mathcal{D}, \mathcal{E})$ be a directed graph, $\mathcal{E} \subseteq \mathcal{D} \times \mathcal{D}$, and assume that the nodes are allowed to communicate only along the directed edges in $\mathcal{E}$. Assume that each node $v \in \mathcal{D}$ owns a set of objects $O_v \subseteq U$, where U is a "universe" of size u. In practice, O_v can be a collection (set) of files, which can be viewed as binary vectors of maximum length n, or as vectors over some extension field of $\mathbb{F}_2$. The *synchronization problem* is defined as computing $\cup_{v \in \mathcal{D}} O_v$ at all nodes $v \in \mathcal{D}$, while trying to minimize communication, computation and time complexities. We will refer to an algorithm that achieves such a synchronization as a *set reconciliation algorithm.*

A related problem is a *subspace synchronization problem.* Let V_v be a vector space associated with every node $v \in \mathcal{D}$, $V_v \subseteq W$, and W is an ambient space. Given initial subspaces V_v at each node $v \in \mathcal{D}$, the *subspace synchronization problem* entails computing the subspace $O = \sum_{v \in \mathcal{D}} V_v$ at every vertex $v \in \mathcal{D}$.

III. Two-Party Subspace Synchronization

Let $\mathbb{F}$ be a finite field with q elements. Let $\mathcal{G}_2 = (\mathcal{D}, \mathcal{E})$ denote the graph that has exactly two nodes, $\mathcal{D} = \{w, v\}$, connected with two anti-parallel edges; i.e., $\mathcal{E} = \{(w, v), (v, w)\}$. Assume also that the nodes w and v initially possess vector spaces U and V, respectively, which are subspaces of an ambient space W over the finite field $\mathbb{F}$; i.e., $U, V \subseteq W \subseteq \mathbb{F}^n$ with n being a positive integer. Let u be the size of the universe of the vectors, i.e., $u = q^n$.

The proposed synchronization algorithm $\mathcal{A}$ works in the following way. Let $d = \mathsf{dist}(U, V)$ and $k = \max\{\dim(U), \dim(V)\}$. The algorithm proceeds in rounds. At each iteration of the algorithm $\mathcal{A}$, the node w draws a nonzero vector $\boldsymbol{x}$ randomly and uniformly in U. It communicates this vector to v. The node v checks if the vector $\boldsymbol{x}$ is in V. If not, v expands V by this vector; namely it performs

$$V \leftarrow V \oplus \langle \boldsymbol{x} \rangle \; .$$

After this operation is repeated for $\Theta(d)$ rounds, the roles of w and v are switched: now it is v that sends random vectors to w for $\Theta(d)$ rounds.

Lemma 1. Let $\Delta = \dim(U) - \dim(U \cap V)$. If $\Delta \geq 1$, then for a random nonzero vector $\boldsymbol{x}$ drawn uniformly from U, the probability $\mathsf{Prob}(\boldsymbol{x} \notin V) \geq \frac{q-1}{q}$.

Proof. The selected vector $\boldsymbol{x}$ is in U. The number of nonzero vectors in U is $q^{\dim(U)} - 1$, and the number of vectors in $U \cap V$ is $q^{\dim(U \cap V)} - 1$. The probability that $\boldsymbol{x}$ is not in V is

$$\begin{aligned} \mathsf{Prob}(\boldsymbol{x} \notin V) &= 1 - \mathsf{Prob}(\boldsymbol{x} \in V) \\ &= 1 - \frac{q^{\dim(U \cap V)} - 1}{q^{\dim(U)} - 1} \\ &\geq 1 - q^{\dim(U \cap V) - \dim(U)} \\ &\geq 1 - \frac{1}{q} \; . \end{aligned}$$

□

It follows from Lemma 1, that at each iteration, the dimension of V grows by one with probability at least $\frac{q-1}{q} \geq \frac{1}{2}$. Therefore, after $c \cdot d$ steps, where c is a sufficiently large constant, $\dim(U \cap V) = \dim(U)$ with high probability, and so $U \subseteq V$. By a similar argument, after an additional $O(d)$ steps, $V \subseteq U$ and $U \subseteq V$ with high probability. This leads us to the following conclusion.

Theorem 1. The randomized algorithm $\mathcal{A}$ performs subspace synchronization over the graph $\mathcal{G}_2$ with communication cost $\textsc{Communication}(\mathcal{A}) = O(d \cdot n \log q) = O(d \cdot \log u)$, computation complexity $\textsc{Computation}(\mathcal{A}) = O(k^2 \cdot n)$, and time $\textsc{Time}(\mathcal{A}) = 2$, with high probability.

Proof. We have already shown that the expected number of communicated vectors is $O(d)$ with high probability. Each vector consists of n symbols in $\mathbb{F}$, and each of the symbols has $O(\log q)$ bits in its representation.

The next parameter in question is computation time. The receiving node can first write all its vectors as rows in a matrix, and to bring it into the row-echelon form, which takes $O(k^2 \cdot n)$ operations. Next, testing whether a new vector lies in the row space of the given matrix takes $O(k \cdot n)$ operations. Since there are only $O(d)$ such vectors, and $d \leq 2k$, the total complexity is $O(k^2 \cdot n)$.

Due to Lemma 1, if the new received vector lies in the row space of the vectors that the node already holds, then with high probability all the required vectors are already in the possession of the node. This probability can be further improved by repeating this check with another (independent) vector(s). At that point, the node has all the required vectors with high probability, and the synchronization round is complete. This yields that $\textsc{Time}(\mathcal{A}) = 2$. □

Clearly, if $d = \mathsf{dist}(U, V)$ then, in the absence of additional assumptions, any synchronization protocol requires at least $\Omega(d \cdot n \log q)$ communication, and so the algorithm $\mathcal{A}$ is order optimal.

IV. Reconciliation Using Error-Correcting Codes

In this section, we describe a method for constructing subspaces for the synchronization problem using error-correcting codes. The main idea of this approach is to introduce a mapping of arbitrary vectors onto linearly-independent vectors (which are taken as columns of the parity-check matrix of a Reed-Solomon code). In that case, these linearly independent vectors can serve as a basis of the vector subspace. This approach is suitable if the number of objects is relatively small when compared to their size.

Consider a classical $[\mathsf{n},\mathsf{k},\mathsf{d}]$-linear code $\mathcal{C}$ over the finite field $\mathbb{F}=\mathbb{F}_q$, such that $\mathsf{n}\geq 2^m$ for some integer $m>0$. For example, we can use a Reed-Solomon code with $\mathsf{n}+1=\mathsf{k}+\mathsf{d}$. Let the $(\mathsf{n}-\mathsf{k})\times\mathsf{n}$ parity-check matrix of $\mathcal{C}$ be

$$H=[\boldsymbol{h}_1 \mid \boldsymbol{h}_2 \mid \cdots \mid \boldsymbol{h}_{\mathsf{n}}] ,$$

where $\boldsymbol{h}_i$'s are the columns of H.

With every vector $\boldsymbol{x}\in\{0,1\}^m$ we associate a unique integer index $\phi(\boldsymbol{x})\in[\mathsf{n}]$. Thus if $\boldsymbol{x}_1\neq\boldsymbol{x}_2$, we have $\phi(\boldsymbol{x}_1)\neq\phi(\boldsymbol{x}_2)$. Assume that $O=\{\boldsymbol{x}_i\}_{i\in S}$ is a collection of objects for some $S\subseteq[\mathsf{n}]$. We represent the collection O by the vector space

$$\Phi(O)\triangleq\langle\boldsymbol{h}_{\phi(\boldsymbol{x})}\rangle_{\boldsymbol{x}\in O} , \tag{1}$$

i.e., the vector space that is obtained by taking a linear span of the columns in H corresponding to the objects (vectors) in O.

In order to perform reconciliation of two sets of objects, O_1 and O_2, the corresponding vector spaces V_1 and V_2 are constructed, such that $V_i=\Phi(O_i)$ for $i=1,2$. Then the synchronization algorithm $\mathcal{A}$ is applied to V_1 and V_2.

Theorem 2. Let $O_1\neq O_2$ be two nonempty sets of objects to be reconciled. If d is chosen such that

$$\mathsf{d}\geq 4\cdot\max\{|O_1|,|O_2|\}+1 , \tag{2}$$

then the following three statements hold:

1) $\Phi(O_1)\neq\Phi(O_2)$;
2) The pre-image of the subspace $\Phi(O_1)+\Phi(O_2)$ is unique; and
3) $\Phi(O_1)+\Phi(O_2)=\Phi(O_1\cup O_2)$.

Proof. 1) Based on the choice of d in (2), any set of less than d columns in H is linearly independent. Let S_1 and S_2 be sets of indices of objects in O_1 and O_2, respectively, and $O_1\neq O_2$. Then, for any combination of α's and β's from $\mathbb{F}$, not all zeros,

$$\sum_{\boldsymbol{x}\in O_1}\alpha_{\phi(\boldsymbol{x})}\boldsymbol{h}_{\phi(\boldsymbol{x})}+\sum_{\boldsymbol{x}\in O_2}\beta_{\phi(\boldsymbol{x})}\boldsymbol{h}_{\phi(\boldsymbol{x})}\neq\mathbf{0} .$$

This is due to the fact that there exists $\boldsymbol{x}\in O_1\backslash O_2$ (or $\boldsymbol{x}\in O_2\backslash O_1$), and the number of such elements $\boldsymbol{x}$ is less than d. Therefore, the sum of the corresponding linearly independent columns is non-zero, and so $\Phi(O_1)\neq\Phi(O_2)$ for any $O_1\neq O_2$.

2) Let $V=V_1+V_2$. Then, from (2), $\dim(V)<\mathsf{d}/2$. Suppose that there exists a set $\hat{S}$ of $|\hat{S}|<\mathsf{d}/2$ columns in H, such that $V=\langle\boldsymbol{h}_i\rangle_{i\in\hat{S}}$. Take any nonzero vector $\boldsymbol{v}\in V$. We can write

$$\boldsymbol{v}=\sum_{i\in\hat{S}}\beta_i\boldsymbol{h}_i=\sum_{\boldsymbol{x}\in O_1\cup O_2}\alpha_{\phi(\boldsymbol{x})}\boldsymbol{h}_{\phi(\boldsymbol{x})} .$$

The second equality, which holds for all $\boldsymbol{v}\in V$, implies that $O_1\cup O_2=\hat{S}$ since any collection of $\mathsf{d}-1$ or fewer distinct vectors are linearly independent. We conclude that V has a unique pre-image under Φ, which is the set $O_1\cup O_2$.

3) The consistency $\Phi(O_1)+\Phi(O_2)=\Phi(O_1\cup O_2)$ follows directly from the definition of the mapping Φ in (1).

□

Next, we analyze the selection of the parameters. If a Reed-Solomon code is used and the size of the underlying field $\mathbb{F}$ is large enough, then there exists a code $\mathcal{C}$ with $\mathsf{n}=\mathsf{k}+\mathsf{d}-1$. For such code, the columns in H have length $\mathsf{d}-1$, i.e., the parameters n and k can always be chosen as in (2).

Note, that the expected number of communicated vectors is bounded by $O(\mathsf{d})$. Thus, the proposed reconciliation algorithm has communication complexity $O(\mathsf{d}\cdot(\mathsf{n}-\mathsf{k}))=O(\mathsf{d}^2)$ symbols over $\mathbb{F}$, or equivalently, $O(\mathsf{d}^2\log q)$ bits.

In order for an appropriate Reed-Solomon code to exist, we should have that $q\geq\mathsf{n}$. Therefore, the size of the alphabet should be proportional to n. Thus, the number of bits needed to represent one symbol from $\mathbb{F}$ is logarithmic in n. The overall communication complexity of the proposed approach is $\textsc{Communication}(\mathcal{A})=O(\mathsf{d}^2\log_2\mathsf{n})$.

We note that the number of various binary vectors of length m is $u=2^m\approx\mathsf{n}$. Therefore, $\textsc{Communication}(\mathcal{A})=O(\mathsf{d}^2m)=O(\mathsf{d}^2\log u)$. This communication complexity is higher than its counterpart in [3], [6], [7].

Next, we analyze $\textsc{Computation}(\mathcal{A})$. Essentially, the user should find the columns of H that serve as a basis of the received space. This can be done in $O(\mathsf{d}^2\cdot\mathsf{n})$ time by solving a system of $O(\mathsf{d})$ linear equations with $O(\mathsf{n})$ unknowns. Then, the user should be able to efficiently map between a subset of the set of columns of H and the information vector. One way to achieve that is by enumerative encoding. Another way is to use a large look-up table, in which case the computational complexity is $\log_2\mathsf{n}$ by using binary search. The total computational complexity in that case is $O(\mathsf{d}^2\cdot u)$.

As for the time, $\textsc{Time}(\mathcal{A})=2$, similarly to the analysis of the protocol described in Section III.

V. Reconciliation algorithm using network coding and hash functions

In this section, building on the ideas from Section III, we present a new method for reconciliation of sets of large files. The method uses ideas from network coding — in particular, from non-coherent network coding [4].

We assume a scenario with two users, A and B, connected by antiparallel links. Let $\mathbb{F}$ be a finite field as before, and let

n be a large integer (the length of files). Here we make the simplifying assumption that all files are of the same length. The results can be easily extended to the case when n is an upper bound on the length of a file.

Let k be the total number of objects (files) in joint possession of A and B. We index them by the elements of the set $\mathcal{X} = [k]$. Denote by $O_A = \{\boldsymbol{x}_i \in \mathbb{F}^n\}_{i \in \mathcal{X}_A}$ and $O_B = \{\boldsymbol{x}_i \in \mathbb{F}^n\}_{i \in \mathcal{X}_B}$ the set of objects, which are unique to A and to B, respectively. Also denote by $O_C = \{\boldsymbol{x}_i \in \mathbb{F}^n\}_{i \in \mathcal{X}_O}$ the set of objects which are possessed by both A and B. Here, $\mathcal{X}_A, \mathcal{X}_B$ and $\mathcal{X}_O$ are subsets of $\mathcal{X}$. Let $s = |\mathcal{X}_A|$ and $\tau = |\mathcal{X}_A \cup \mathcal{X}_O|$. As before, let $d = |\mathcal{X}_A \cup \mathcal{X}_B|$ be the number of different files for A and B. We assume that s, or a tight upper bound on it, is known to both A and B. Towards the end of the section, we discuss a modification for the case when s is not known to A and B.

User A performs the following operations. First, it creates s arbitrary linear combinations of the form

$$\boldsymbol{y}_j = \sum_{i \in \mathcal{X}_A \cup \mathcal{X}_O} \alpha_{j,i} \boldsymbol{x}_i \ , \ j \in [s] \ ,$$

where the coefficient vectors $\boldsymbol{\alpha}_j = (\alpha_{j,i})_{i \in \mathcal{X}_A \cup \mathcal{X}_O}$ (for each $j \in [s]$) are selected uniformly at random in $\mathbb{F}^\tau$. User A checks that all vectors $\boldsymbol{\alpha}_j$, $j \in [s]$, are linearly independent. (If they are not, A replaces linearly dependant vectors, and repeats the process again.)

The protocol uses a hash function $\mathcal{H} \ : \ \mathbb{F}^n \rightarrow \mathbb{K}$, where $\mathbb{K}$ is the finite set of possible keys. (We will discuss hash functions in more detail in the sequel.) User A applies $\mathcal{H}$ to $\boldsymbol{x}_i$ for all $i \in \mathcal{X}_A \cup \mathcal{X}_O$ to produce hash values $\mathcal{H}(\boldsymbol{x}_i)$ for all i. These values are transmitted to B. Their goal is to impose an order on the vectors in $\mathcal{X}_A \cup \mathcal{X}_O$, so that the transmitted coefficients can be matched to the objects by B with high probability of success.

To this end, A transmits to B the following data:

- the header $\boldsymbol{h}$, which contains the sorted list of values $\mathcal{H}(\boldsymbol{x}_i)$, $i \in \mathcal{X}_A \cup \mathcal{X}_O$;
- for all $j \in [s]$, the vector pairs $(\boldsymbol{\alpha}_j, \boldsymbol{y}_j)$.

Let $\boldsymbol{X}$ be a $\tau \times n$ matrix over $\mathbb{F}$, whose rows are all vectors $\boldsymbol{x}_i$ indexed by $[\tau]$. Similarly, let $\boldsymbol{Y}$ be a $s \times n$ matrix, whose rows are vectors $\boldsymbol{y}_i$ for all $i \in [s]$. Denote

$$\boldsymbol{A} = \begin{pmatrix} \alpha_{1,1} & \alpha_{1,2} & \cdots & \alpha_{1,\tau} \\ \alpha_{2,1} & \alpha_{2,2} & \cdots & \alpha_{2,\tau} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_{s,1} & \alpha_{s,2} & \cdots & \alpha_{s,\tau} \end{pmatrix} .$$

The transmitted vector pairs can be viewed as the rows of the matrix

$$\boldsymbol{A} \cdot [\, \boldsymbol{I}_\tau \mid \boldsymbol{X} \,] = [\, \boldsymbol{A} \mid \boldsymbol{Y} \,] \ ,$$

where

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \vdots \\ \boldsymbol{x}_\tau \end{bmatrix}$$

and

$$\boldsymbol{Y} = \boldsymbol{A}\boldsymbol{X} = \begin{bmatrix} \boldsymbol{y}_1 \\ \boldsymbol{y}_2 \\ \vdots \\ \boldsymbol{y}_s \end{bmatrix} .$$

After receiving this message, user B performs the following operations to recover the missing objects.

1) Compute values of the function $\mathcal{H}$ applied to the vectors in its possession. By comparing these values to the values in the header $\boldsymbol{h}$, it finds the indices corresponding to elements in $\mathcal{X}_O$.
2) For each $j \in [s]$, subtract vectors $\sum_{i \in \mathcal{X}_O} \alpha_{j,i} \boldsymbol{x}_i$ from $\boldsymbol{y}_j$. Compute the resulting matrix with s rows:
$$\left[\, \tilde{\boldsymbol{A}} \mid \tilde{\boldsymbol{Y}} \,\right] ,$$
where rows of $\tilde{\boldsymbol{Y}}$ are the vectors
$$\tilde{\boldsymbol{y}}_j = \boldsymbol{y}_j - \sum_{i \in \mathcal{X}_O} \alpha_{j,i} \boldsymbol{x}_i \ ,$$
and $\tilde{\boldsymbol{A}}$ is an invertible $s \times s$ matrix obtained from $\boldsymbol{A}$ by removing the columns corresponding to the vectors indexed by $\mathcal{X}_O$. The matrix $\tilde{\boldsymbol{A}}$ is invertible due to linear independence of the columns in $\boldsymbol{A}$.
3) Compute the matrix
$$\left[\, \boldsymbol{I} \mid \tilde{\boldsymbol{A}}^{-1} \tilde{\boldsymbol{Y}} \,\right] = \left[\, \boldsymbol{I} \mid \tilde{\boldsymbol{X}} \,\right] ,$$

where, if there are no hashing collisions, $\tilde{\boldsymbol{X}}$ is exactly the matrix $\boldsymbol{X}$ having rows corresponding to the vectors indexed by $\mathcal{X}_A$.

The algorithm can fail at Step 1 if the hash function $\mathcal{H}$ maps more than one vector in $\mathcal{X}$ to the same key value. Therefore, the hash function needs to be selected from a set of sufficiently large number of hash functions, uniformly at random. If the image of $\mathcal{H}$ is sufficiently large, and $\mathcal{H}$'s are sufficiently independent, then the collision probability can be made small.

To estimate the communication complexity of the proposed algorithm, observe that A sends s vectors to B. Each vector consists of the data (n entries from $\mathbb{F}$) and the header (τ entries from $\mathbb{F}$). Typically, $\tau \ll n$. Therefore, the total communication complexity is $O(d \cdot n)$ symbols from $\mathbb{F}$, or equivalently $\textsc{Communication}(\mathcal{A}) = O(d \cdot n \log q)$ (when measured in bits).

User A needs to verify that the vectors $\boldsymbol{\alpha}_j$ are linearly independent. This can be done by Gaussian elimination, requiring $O(s^3)$ operations. User B has to subtract linear combinations of vectors of length n. This requires $O(\tau^2 \cdot n)$ operations. Additionally, inverting the matrix $\tilde{\boldsymbol{A}}$ requires $O(s^3)$ operations, and computation of $\tilde{\boldsymbol{X}}$ requires additional $O(s^2 \cdot n)$ operations. Recall that $\tau = O(k)$ and $s = O(k)$. We conclude that the total computational complexity is $\textsc{Computation}(\mathcal{A}) = O(k^2 \cdot n)$.

Finally, if s is known, then $\textsc{Time}(\mathcal{A}) = 2$ since A sends linear combinations to B and vice versa. If s is not known, then before the execution of the protocol, B sends to A the list of hash values of each of its files. In this way A can determine the value of s. In that case, the number of rounds becomes $\textsc{Time}(\mathcal{A}) = 3$.

We note that the proposed algorithm has essentially optimal communication complexity. Its number of communication rounds is better than that of its counterpart in [6], and similar to that of [2], [3], [7]. The computational complexity, however is inferior to the algorithms [2], [3], [7].

VI. Avoiding collisions

As it was pointed out earlier, the proposed reconciliation algorithm can fail if two or more different files are mapped to the same value by the hash function. This problem can be resolved in various ways. For example, if the hash function is selected well, then the expected number of such collisions is small. Then, the two-stage algorithm can be used. In the first stage, the servers exchange the lists of hash values of the files in their possession and then A sends to B the files, which have the same values of hash function. In the second stage, the algorithm proceeds as described in the previous section. However, this solution is not very elegant.

Another solution is based on the use of a large number of different hash functions. For each round of the algorithm, the hash function is selected randomly from a pre-determined pool of hash functions. The pool is known to both users. Before the beginning of the execution of the protocol, the user A sends to B also the ID number of the selected hash function he is using.

In this section, we discuss the existence of suitable random hash functions (which are not necessarily computationally efficient) that can potentially be used in such an algorithm. More specifically, we analyze the size of the key space of such hash functions required to have a desired level of performance.

Assume that we have a collection $\mathbb{S}$ of k different files in $\{0,1\}^n$. Let $\mathbb{H}$ be a set of all functions $\mathcal{H} : \{0,1\}^n \to \mathbb{K}$, where $\mathbb{K}$ is the set of all possible keys. We assume hereafter that $k \ll |\mathbb{K}| \ll 2^n$.

Now, denote by P the probability that for any pair of distinct files $\boldsymbol{x}_i, \boldsymbol{x}_j \in \mathbb{S}$, $\mathcal{H}(\boldsymbol{x}_i) \neq \mathcal{H}(\boldsymbol{x}_j)$. Since the values of the function $\mathcal{H}$ are uniformly random, the probability that its values on the k given files are all different is

$$\begin{aligned} P &= \frac{|\mathbb{K}|(|\mathbb{K}|-1)\cdots(|\mathbb{K}|-k+1)}{|\mathbb{K}|^k} \\ &= \left(1-\frac{1}{|\mathbb{K}|}\right)\left(1-\frac{2}{|\mathbb{K}|}\right)\cdots\left(1-\frac{k-1}{|\mathbb{K}|}\right) \\ &\geq \left(1-\frac{k-1}{|\mathbb{K}|}\right)^{k-1} \\ &= \left(\left(1-\frac{1}{|\mathbb{K}|/(k-1)}\right)^{|\mathbb{K}|/(k-1)}\right)^{(k-1)^2/|\mathbb{K}|} \\ &\approx \mathsf{e}^{-\frac{(k-1)^2}{|\mathbb{K}|}}, \end{aligned}$$

where the last transition holds for sufficiently large values of $|\mathbb{K}|$. We conclude that if $\mathbb{K}$ is selected such that $|\mathbb{K}| > c \cdot (k-1)^2$ for some large constant $c > 0$, then the probability of success is at least $\mathsf{e}^{-1/c}$. It can be made as close to one as desired by taking sufficiently large c. Of course, working with the set $\mathbb{H}$ of all possible hash functions is not feasible, as we need too many bits to encode their indices. However, this suggests that working with a relatively small subset of hash functions will do well in most of the cases.

VII. Discussion

In this work, we first show that subspace synchronization can be performed efficiently. Next, we propose new file reconciliation algorithms, which mimic the idea of subspace synchronization and use ideas from the area network coding.

One advantage of the proposed algorithm is that it can also be used in one-directional reconciliation. For example, if only a one-directional channel from A to B is available, and there is no channel in the opposite direction, the general algorithm still can be used for B to obtain a reconciled set of files using just one communications round (i.e., $\textsc{Time}(\mathcal{A}) = 1$). This is generally not the case in more straight-forward algorithms, which are based on tables or lists and assume exchange of the information in two directions.

We note that the new algorithm has essentially optimal communication complexity. Its number of communication rounds is better than that of its counterpart in [6], and similar to that of [2], [3], [7]. The computational complexity, however is inferior to the algorithms [2], [3], [7]. Although the proposed methods are less computationally efficient, the approach proposed in this paper offers a different perspective on file reconciliation which may lead to improved algorithms in the future. It is interesting to compare the proposed algorithm to its counterpart in [6]. If the size of the files, n, is large, then the latter requires performing multiplications in the field with 2^n elements, which can be very time-consuming. By contrast, the proposed algorithm requires only additions of vectors of length n and multiplications in the ground field.

References

[1] R. Ahlswede, N. Cai, S.-Y. R. Li and R.W. Yeung, "Network information flow", *IEEE Trans. on Inform. Theory*, no. 46, vol. 4, Jul. 2000.

[2] D. Eppstein, M. Goodrich, F. Uyeda, and G. Varghese, "What's the difference? Efficient set reconciliation without prior context", in *Proc. ACM SIGCOMM*, pp. 218—229, 2011.

[3] M. Goodrich and M. Mitzenmacher, "Invertible Bloom lookup tables", *Proc. 49th Annual Allerton Conference*, pp. 792–799, 2011.

[4] R. Koetter and F.R. Kschischang, "Coding for errors and erasures in random network coding", *IEEE Trans. on Inform. Theory*, no. 54, vol. 8, pp. 3579-3591, Aug. 2008.

[5] R. Koetter and M. Médard, "An algebraic approach to network coding", *IEEE/ACM Trans. Networking*, no. 11, vol. 5, pp. 782–795, 2003.

[6] Y. Minsky, A. Trachtenberg, and R. Zippel, "Set reconciliation withnearly optimal communication complexity", *IEEE Transactions on Information Theory,* vol. 49(9):2213–2218, 2003.

[7] M. Mitzenmacher and G. Varghese, "Biff (Bloom filter) codes: fast error correction for large data sets, *Proc. International Symposium on Information Theory (ISIT)*, 2012.

[8] M. Mitzenmacher and G. Varghese, "The Complexity of Object Reconciliation, and Open Problems Related to Set Difference and Coding", *Proc. Allerton Conference,* 2012.

[9] D. Silva, F.R. Kschischang and R. Koetter, "A rank-metric approach to error control in random network coding", *IEEE Trans. on Inform. Theory*, no. 54, vol. 9, pp. 3951-3967, 2008.