Information Theoretic Limits of Data Shuffling for Distributed Learning

Mohamed Adel Attia  Ravi Tandon
Department of Electrical and Computer Engineering
University of Arizona, Tucson, AZ, 85721
Email: \{madel, tandonr\}@email.arizona.edu

Abstract—Data shuffling is one of the fundamental building blocks for distributed learning algorithms, that increases the statistical gain for each step of the learning process. In each iteration, different shuffled data points are assigned by a central node to a distributed set of workers to perform local computations, which leads to communication bottlenecks. The focus of this paper is on formalizing and understanding the fundamental information-theoretic tradeoff between storage (per worker) and the worst-case communication overhead for the data shuffling problem. We completely characterize the information theoretic tradeoff for \( K = 2 \), and \( K = 3 \) workers, for any value of storage capacity, and show that increasing the storage across workers can reduce the communication overhead by leveraging coding. We propose a novel and systematic data delivery and storage update strategy for each data shuffle iteration, which preserves the structural properties of the storage across the workers, and aids in minimizing the communication overhead in subsequent data shuffling iterations.

I. INTRODUCTION

Distributed computing systems for large data-sets have gained a lot of interest recently as they enable the processing of data-intensive tasks for machine learning, model tracing, and data analysis over a large number of commodity machines, and servers (e.g., Apache Spark\(^1\), and MapReduce\(^2\)). Generally speaking, a master node, which has the entire data-set, sends data blocks to be processed at distributed worker nodes. The workers subsequently respond with locally computed functions to the master node for the desired data analysis. This enables the processing of many terabytes of data over thousands of distributed servers in a time efficient manner.

One of the core elements in distributed learning algorithms is data shuffling\(^3\). Before each iteration of the learning process, the entire data is randomly shuffled before being assigned to the worker nodes. This shuffling operation enables the worker nodes to process different data batches at each iteration, which presents large statistical gains\(^4\),\(^5\). The statistical advantages provided by data shuffling come at the unavoidable cost of the communication overhead between the master and the worker nodes which must be incurred for every shuffling iteration. Thus, there exists a fundamental tradeoff between the communication overhead and the storage capacity of each worker node. To exemplify this tradeoff, consider two extreme scenarios: an ideal scenario in which the storage at the distributed workers is large enough to store the entire data-set, thus no communication has to be done from the master node for any shuffle. On the other extreme, when the storage is just enough to store the batch under processing, the communication load is expected to be maximum.

The focus of this paper is on formalizing and understanding this fundamental information-theoretic tradeoff between storage and the worst-case communication overhead for the data shuffling problem. Each iteration of data shuffling can be divided into two phases: data delivery, and storage update. In the data delivery phase, depending on the shuffled data points, the master node communicates a function of the data to the workers, so that each worker obtains its assigned data points. The second phase is termed as the storage update phase, which as shown in this paper is extremely critical in reducing the communication overhead of subsequently shuffling iterations. We next summarize the main contributions of this paper:

- We first present an information-theoretic formulation of the data shuffling problem involving both data delivery and update phases, accounting for the respective constraints and formalizing the tradeoff between worst-case communication overhead and the storage capacity of distributed workers.

- We also completely characterize this tradeoff for \( K = 2 \) and \( K = 3 \) workers, for any value of storage capacity. One of the most interesting aspects of the result for \( K = 3 \) worker problem is the design of data delivery and storage update algorithms. In particular, for data delivery phase, we show that transmitting coded data from the master node to the workers can significantly reduce the communication overhead. More interestingly, the proposed storage update algorithm maintains the structural properties of the storage at the workers over time. This structural invariance placement is extremely critical in leveraging the gains of coding for different shuffles.

Related Work: In the past few years, there has been a flurry of research activity in understanding the benefits of coding for caching starting from the work of Maddah-Ali and Niesen\(^6\) who showed that exploiting multi-casting opportunities by coding can reduce the communication for caching. In\(^7\), coding for MapReduce was proposed in order to reduce the communication cost between mappers and reducers, however the underlying focus of\(^7\) is significantly different than the problem considered in this work, where we care about the communication between the master node and the workers. The paper most closely related to this work is\(^8\), where the idea of coding for data shuffling problem is presented to reduce the communication overhead between the master node and worker nodes.\(^9\) provides a probabilistic scheme of leveraging coding based on a random storage placement. In contrast to\(^8\), in this paper we provide a deterministic and systematic storage update scheme, which increases the coding opportunities in the delivery phase. The underlying metric used here is the worst-case communication cost over all the possible shuffles, unlike the average cost considered in\(^8\). Finally, we also present the first information theoretic lower bounds on the communication overhead for the data shuffling problem.
II. System Model

We assume a master node which has access to the entire data-set $A = [x_1, x_2, \ldots, x_N]^T$ of size $Nd$ bits, i.e., $A$ is a matrix containing $N$ data points, denoted by $x_1, x_2, \ldots, x_N$, where $d$ is the dimensionality of each data point. Treating $A$, and its data points $x_n$ as random variables, we therefore have the entropies of these random variables as

$$H(A) = N \times H(x_n) = Nd, \quad \forall n \in \{1, 2, \ldots, N\}. \quad (1)$$

At each iteration, indexed by $t$, the master node divides the data-set $A$ into $K$ data batches given as $A_1^t, A_2^t, \ldots, A_K^t$, where the batch $A_k^t$ is designated to be processed by worker $w_k$, and these batches correspond to the random permutation of the data-set, $\pi^t : A \rightarrow \{A_1^t, \ldots, A_K^t\}$. Note that these data chunks are disjoint, and span the whole data-set, i.e.,

$$A_i^t \cap A_j^t = \phi, \quad \forall i \neq j, \quad (2a)$$

$$A_i^t \cup A_j^t \cup \ldots \cup A_K^t = A, \quad \forall t. \quad (2b)$$

Hence, the entropy of any batch $A_k^t$ is given as

$$H(A_k^t) = \frac{1}{K} H(A) = \frac{N}{K} d, \quad \forall k \in \{1, \ldots, K\}. \quad (3)$$

After getting the data batch, each worker locally computes a function (as an example, this function could correspond to the gradient or sub-gradients of the data points assigned to the worker). The local functions from the $K$ workers are processed subsequently at the master node. We assume that each worker $w_k$ has a storage $Z_k^t$ of size $Sd$ bits, for a real number $S$. For processing purposes, the assigned data blocks are needed to be stored by the workers, therefore, each worker $w_k$ must at least store the data block $A_k^t$ at time $t$. If we consider $Z_k^t$ as a random variable then the storage constraint is given by

$$H(Z_k^t) = Sd \geq H(A_k^t), \quad \forall k \in \{1, \ldots, K\}. \quad (4)$$

According to [3] and [4], we get the minimum storage per worker $S \geq \frac{N}{K}$. We also have the processing constraint as

$$H(A_k^t | Z_k^t) = 0, \quad \forall k \in \{1, \ldots, K\}. \quad (5)$$

In the next epoch $t+1$, the data-set is randomly reshuffled at the master node according to a random permutation $\pi^{t+1} : A \rightarrow \{A_1^{t+1}, A_2^{t+1}, \ldots, A_K^{t+1}\}$. The main communication bottleneck occurs during Data Delivery since the master node needs to communicate the new data batches to the workers. Trivially, if the storage (per worker) exceeds $Nd$ bits, i.e., $S \geq N$, then each worker can store the whole data-set, and no communication has to be done between the master node and the workers for any shuffle. Therefore from the constraint on minimum storage per worker, we can write the possible range for storage as $\frac{N}{K} \leq S \leq N$.

We next proceed to describe the data delivery mechanism, and the associated encoding and decoding functions. The main process can be divided into 2 phases, namely the data delivery phase and the storage update phase as described next:

A. Data Delivery Phase

At time $t+1$, the master node sends a function of the data batches for the subsequent shuffles $(\pi_t, \pi_{t+1})$, $X(\pi_t, \pi_{t+1}) = \phi(A_1^t, \ldots, A_K^t, A_1^{t+1}, \ldots, A_K^{t+1}) = \phi(\pi_t, \pi_{t+1})(A)$ over the shared link, where $\phi$ is the data delivery encoding function

$$\phi : \left[\frac{2N}{K}d\right]^{2K} \rightarrow [2^{R_{t}(\pi_{t}, \pi_{t+1})}d], \quad (6)$$

where $R_{t}(\pi_{t}, \pi_{t+1})$ is the rate of the shared link based on the shuffles $(\pi_t, \pi_{t+1})$. Therefore, we have

$$H(X(\pi_t, \pi_{t+1}) | A) = 0, \quad H(X(\pi_t, \pi_{t+1})) = R_{t}(\pi_{t}, \pi_{t+1}) d. \quad (7)$$

Each worker $w_k$ should decode the desired batch $A_k^{t+1}$ out of the transmitted function $X(\pi_t, \pi_{t+1})$, and the data stored in the previous time slot denoted as $Z_k^t$. Therefore, the desired data is given by $A_k^{t+1} = \psi(X(\pi_t, \pi_{t+1}), Z_k^t)$, where $\psi$ is the decoding function at the workers

$$\psi : \left[2^{R_{t}(\pi_{t}, \pi_{t+1})d}\right] \times [2^{Sd}] \rightarrow [2^{N}]d, \quad (8)$$

which can be written in terms of a decodability constraint, at each worker as follows

$$H(A_k^{t+1} | Z_k^t, X(\pi_t, \pi_{t+1})) = 0, \quad \forall k \in \{1, \ldots, K\}. \quad (9)$$

B. Storage Update Phase

At each iteration, every worker updates its stored content as follows: the new storage content $Z_k^{t+1}$ is a function of the old storage content $Z_k^t$ as well as transmitted function $X(\pi_t, \pi_{t+1})$, i.e., $Z_k^{t+1} = \mu(X(\pi_t, \pi_{t+1}), Z_k^t)$, where $\mu$ is the update function

$$\mu : \left[2^{R_{t}(\pi_{t}, \pi_{t+1})d}\right] \times [2^{Sd}] \rightarrow [2^{S}]d. \quad (10)$$

This implies the following storage-update constraint

$$H(Z_k^{t+1} | Z_k^t, X(\pi_t, \pi_{t+1})) = 0, \quad \forall k \in \{1, \ldots, K\}. \quad (11)$$

The excess storage, if any, can be used to store opportunistically a function of the remaining data batches. Since the shuffling process at each time is done randomly, all the remaining batches are of equal importance. Consequently, the amount of excess storage, given by $(S - \frac{N}{K})d$ bits, is divided equally among the remaining $K-1$ batches. For the scope of this work, we assume that the placement of the excess storage is uncoded, which means that $(S - \frac{N}{K})d$ bits of the excess storage are dedicated to store a function of only one of the remaining $K-1$ batches. We give the notation $A_{i,k}^{t+1}$, where $i \neq k$, as the part of data that worker $w_k$ stores about $A_{i,k}^{t+1}$ in the excess storage at time $t+1$. Considering $A_{i,k}^{t+1}$ as a random variable, then

$$H(A_{i,k}^{t+1} | Z_k^t, Z_{i,k}^{t+1}) = 0, \quad (12)$$

for $k, i \in \{1, \ldots, K\}$, and $i \neq k$.

We next define the worst-case communication as follows:

Definition 1 (Worst-Case Communication). For any achievable scheme characterized by the functions $(\phi, \psi, \mu)$, the worst-case communication overhead over all possible consecutive data shuffles $(\pi_t, \pi_{t+1})$ is defined as

$$R_{\text{worst-case}}(K, S) = \max_{(\pi_t, \pi_{t+1})} R_{t}(\phi, \psi, \mu)(K, S). \quad (13)$$
Our goal in this work is to characterize the optimal worst-case communication $R^\ast_{\text{worst-case}}(K, S)$ defined as
\[
R^\ast_{\text{worst-case}}(K, S) = \min_{(\phi, \psi, \mu)} R^\ast_{\text{worst-case}}(K, S). \tag{14}
\]

We next present a claim which shows that the optimal communication $R^\ast$ (for any shuffle including the worst-case) is a convex function of the storage $S$.

Claim 1. $R^\ast$ is a convex function of $S$, where $S$ is the available storage at each worker.

Proof: Claim 1 follows from a simple memory sharing argument which shows that for any two available storage values $S_1$ and $S_2$, if $(S_1, R^\ast(K, S_1))$, and $(S_2, R^\ast(K, S_2))$ are achievable optimal schemes, then for any storage $S = \alpha S_1 + (1-\alpha) S_2$, $0 \leq \alpha \leq 1$, there is a scheme which achieves a communication overhead of $\bar{R} = \alpha R^\ast(K, S_1) + (1-\alpha) R^\ast(K, S_2)$.

This is done as follows: First, we divide the data-set $A$ across $d$ dimensions into 2 batches namely: $A^{(\alpha)}$, and $A^{(1-\alpha)}$ of dimensions $\alpha d$, and $(1-\alpha)d$, for each point respectively. Then, we divide the storage for every worker $w_t$ into 2 parts namely: $Z_t^{(\alpha)}$, and $Z_t^{(1-\alpha)}$ of size $S_t \alpha d$, and $S_t (1-\alpha)d$, respectively. The former batch $A^{(\alpha)}$ will be shuffled among the former part of the storage $Z_t^{(\alpha)}$ to achieve the point $(S_1, R^\ast(K, S_1))$, while the latter batch $A^{(1-\alpha)}$ will be shuffled among the latter part of the storage $Z_t^{(1-\alpha)}$ to achieve the point $(S_2, R^\ast(K, S_2))$. Therefore, the total achievable load is given by
\[
H(X) = R^\ast(K, S_1) \alpha d + R^\ast(K, S_2)(1-\alpha) d = \bar{R} d. \tag{15}
\]

We next note that the optimal communication rate $R^\ast(K, S)$ is upper bounded by $\bar{R}(K, S)$, the rate of the memory sharing scheme, which completes the proof.

III. MAIN RESULTS

Theorem 1. For the distributed shuffling problem with $K = 2$ workers of storage $S d$ bits each, and a data-set of size $N d$ bits, the optimal communication versus storage tradeoff is given by
\[
R^\ast_{\text{worst-case}}(2, S) = N - S, \quad \frac{N}{2} \leq S \leq N. \tag{16}
\]

Theorem 2. For a distributed shuffling problem system with $K = 3$ workers, the optimal communication versus storage tradeoff is given by
\[
R^\ast_{\text{worst-case}}(3, S) = \begin{cases} \frac{7N}{6} - \frac{3S}{2}, & \frac{N}{3} \leq S \leq \frac{2N}{3} \\ \frac{N}{2} - \frac{S}{2}, & \frac{2N}{3} < S \leq N \end{cases}. \tag{17}
\]

One interesting implication of Theorem 2 for $K = 3$ workers is that the corner point $(\frac{N}{3}, \frac{N}{2})$ as in Fig. 1 is better than memory sharing between the two points $(\frac{N}{3}, \frac{2N}{3})$, and $(N, 0)$, which falls on the line connecting the two points, i.e., memory sharing here is not optimal, and coding can be leveraged to reduce the communication overhead.

IV. PROOF OF THEOREM 1 (K = 2 WORKERS)

A. Achievability for $K = 2$ workers

We start with the achievability of the corner points $S = N$ and $S = \frac{N}{2}$. The point $S = N$ is trivial and represents the case when the workers can store the whole data-set. In this case, no communication is necessary, i.e., $(S, R^\ast) = (N, 0)$ is achievable.

The point corresponding to $S = \frac{N}{2}$ is not trivial, where each worker can only store half of the data-set. Let us assume the data batches at time $t$ for the workers $w_1$, and $w_2$ are $A_1^t$, and $A^t_2$, respectively. These batches should be stored at the corresponding workers, which are just enough to store them, i.e., $Z_t^1 = A_t^1$, and $Z_t^2 = A_t^2$. Recall that these batches are equally sized of $\frac{N}{2} d$ bits. For the next iteration $t + 1$, the data is randomly shuffled at the master node such that the new batches are $A_{t+1}^1$, and $A_{t+1}^2$, also of the same equal size $\frac{N}{2} d$. In the delivery phase, the master node sends
\[
X_{(\pi_t, \pi_{t+1})} = A_t^1 \oplus A_t^2. \tag{18}
\]

worker $w_1$ uses $X_{(\pi_t, \pi_{t+1})}$ and $Z_t^1 = A_t^1$ to decode $A_t^2$, and get access to the whole data-set $A = \{A_t^1, A_t^2\} = \{A_t^{t+1}, A_t^{t+2}\}$. The storage update is only storing the desired batch $A_t^{t+1}$. The same procedure applies for $w_2$. Note that this choice of the transmitted function in (18) works for any possible shuffling, which gives a constant communication load $H(X_{(\pi_t, \pi_{t+1})}) = \frac{N}{2} d$, and hence the corner point $(\frac{N}{3}, \frac{N}{2})$ is achievable.

With the achievability of the corner points, any point in between for any real number $S$ can be simply achieved by memory sharing (see Claim 1). Therefore, the upper bound for the optimal worst-case communication rate is given by
\[
R^\ast_{\text{worst-case}}(2, S) \leq N - S, \quad \frac{N}{2} \leq S \leq N. \tag{19}
\]

Remarks 1 (From worst-case to any shuffle). If there is an overlap between $A_k^t$ and $A_k^{t+1}$ for $k \in \{1, 2\}$, then for $S = \frac{N}{2}$, the communication cost of the above scheme can be further improved by sending
\[
X_{(\pi_t, \pi_{t+1})}^{\text{any-shuffle}} = (A_t^1 \setminus A_t^{t+1}) \oplus (A_t^2 \setminus A_t^{t+2}), \tag{20}
\]
where $A_t^1 \setminus A_t^{t+1}$ represents the part of the old batch $A_t^1$ at worker $w_k$, which is not needed any more in the new batch $A_t^{t+1}$. For an overlap $|A_t^k \cap A_t^{t+1}| = b$ data points, where $b$ is an integer number $b \in \{0, \ldots, \frac{N}{2}\}$, then we achieve the corner point $(\frac{N}{3}, \frac{N}{2} - b)$, with the worst-case rate when $b = 0$. 

![Fig. 1. The worst-case communication overhead is depicted versus different values of storage for $K = 3$ workers. The shaded grey region is the information theoretically optimal tradeoff.](image-url)
B. Converse for K = 2 workers

In this section, we present an information theoretic lower bound for the worst-case communication rate which matches the above scheme for K = 2 workers.

Remark 2 (Basic idea for the converse). Since we do not know a priori the shuffle that gives the worst-case communication, we assume first a shuffle (πa, πa+1) with the optimal rate Rw* (πa, πa+1), and then we lower bound Rw (πa, πa+1). Since the optimal worst-case rate is larger than the rate for any shuffle, i.e., Rw (πa, πa+1) ≥ Rw, the lower bound over Rw (πa, πa+1) serves also as a lower bound for the worst-case communication Rw. The novel part in our proof is choosing the right shuffle which leads to the optimal lower bound (also see Section B for the application of this idea to the converse proof for Theorem 2).

Now let us assume the data is shuffled such that A12+1 = A22, and A22+1 = A12, then from decodability constraint in (9):

\[ H(A12+1|Z1, X(\pia, \pia+1)) = H(A22|Z1, X(\pia, \pia+1)) = 0. \]  

(21)

Hence, we can find that H(A|Z1, X(\pia, \pia+1)) = 0 as follows

\[
(a) \quad H(A)|Z1, X(\pia, \pia+1) = H(A12, A22|Z1, X(\pia, \pia+1))
\]

\[
(b) \quad H(A12|Z1, X(\pia, \pia+1)) + H(A22|Z1, X(\pia, \pia+1))
\]

\[
(c) \quad H(A12|Z1) + H(A22|Z1, X(\pia, \pia+1)) = 0,
\]  

(22)

where (a) follows from (3), (b) follows from the fact that H(A, B) ≤ H(A) + H(B), (c) is because conditioning reduces entropy, and (d) is from (5) and (21). We next prove the lower bound as follows

\[
Nd = H(A) - H(A|Z1, X(\pia, \pia+1)) - I(A; Z1, X(\pia, \pia+1))
\]

\[
(c) \quad H(Z1, X(\pia, \pia+1)) - H(Z1|X(\pia, \pia+1)|A)
\]

\[
(d) \quad H(Z1) + H(X(\pia, \pia+1)) ≤ Sd + Rw (\pia, \pia+1)d.
\]  

(23)

where (a) follows from (11), (b), and (c) follow from the fact that I(A; B) = H(A) - H(A|B) ≤ H(A) + H(B), as well as (22), (d) is due to the fact that H(A, B) ≤ H(A) + H(B)

\[ Rw (\pia, \pia+1), \quad d ≤ 0, \]  

(22)

Hence, from Remark 2 the lower bound for the worst-case rate is characterized as

\[ Rw (\pia, \pia+1), \quad S ≤ N \leq N. \]  

(24)

Hence, the proof of Theorem 1 is complete from (19) and (24).

V. PROOF OF THEOREM 2 (K = 3 WORKERS)

A. Achievability for K = 3 workers

From Fig. 11 the achievability involves three corner points: S = N/2, S = N/3, and S = N. Achieving the point (N, 0) is trivial. The scheme for the point S = N/2 is similar to the point S = N/3 in the K = 2 worker case, where each worker can only store the desired data batches. Similar to (18) the transmission at time t + 1 is X(\pia, \pia+1) = \{A12, A22, A22, A12\}, which is sufficient for each worker to access all the data-set and store what it needs, achieving the corner point (N/2, N/2).

In this section, we focus on the achievability of the corner point (N/3, N/3), which is perhaps the most interesting aspect of this result. For each worker wk at time t, k ∈ {1, 2, 3}, half of the storage (N/3) is used to store the desired batch Akl, while the remaining half (excess storage of N/3 points) is used opportunistically in order to minimize the communication overhead by storing some parts of the remaining batches given by the sub-batches Akl, i ∈ {1, 2, 3} \ k.

These parts will be formed as follows: we consider the data batch Akl assigned for wk, of N points, each point x ∈ Akl divided across d dimensions into two equal sub-divisions labeled as x(i), i ∈ {1, 2, 3} \ k. Then, each one of these sub-divisions x(i) is placed in the corresponding sub-batch Akl, i ∈ {1, 2, 3} \ k. Worker wk will use half of its excess storage to store Akl, while worker wk will use half of its excess storage to store Akl.

The storage update procedure we present here maintains the above structural property of the stored data over time. The consequence of such structural invariance is that for any data point is required to be at a worker, at least half of this point is guaranteed to be already present at the worker, which decreases the communication overhead of the shuffling process.

Data Delivery: With this placement strategy, for the subsequent shuffles (πa, πa+1), the transmitted function is given as

\[ X(\pia, \pia+1) = (A12+1|Z1) + (A22+1|Z2) + (A22+1|Z3). \]  

(25)

We claim that the above transmission is sufficient for all the three workers to obtain the required new points for any shuffle. Without loss of generality, let us consider worker wk. According to (25), for wk to obtain the needed points not available in its storage, A12+1, A22+1, Z1, it must have A22+1, Z2, and A22+1, Z3 in its storage Z1. This is indeed the case and can be proved according to the following argument:

In the following, we prove that A12+1, Z2 ∈ Z1, and using the same argument we can show that A32+1, Z2 ∈ Z1. Consider a data point x ∈ A22+1, which is newly assigned to worker wk at time t + 1 and is not fully present in its storage Z2, i.e., x /∈ A22. Therefore, there are two possibilities: a) x ∈ A22 was being processed by worker wk at time t, which directly implies that x is already available at wk; or b) x ∈ A22 was being processed by worker wk at time t, which implies that the sub-divisions of x are x(1), x(2) and the needed part by wk, x(1) = x ∩ (A22+1 \ Z2), is available at wk by definition (since x(2) is the subdivision that is already present at wk).

According to (25), the worst-case scenario for this scheme happens if there is no overlap between Akl and Akl (completely new assignments). However, half of each data point x ∈ Akl is already stored in the excess storage at wk, labeled x(k), then the worst-case rate of Akl, and eventually X(πa, πa+1) is N/6, achieving the corner point (N/3, N/3).

Storage Update: Now, we present a deterministic storage update strategy, which maintains the structural properties of the storage at time t. Without loss of generality, let us analyze
a data point \( x = \{ x^{(2)}, x^{(3)} \} \in A_t^{1+1} \). The update procedure is done according to the following three cases at time \( t \):

- **Case 1:** \( x = \{ x^{(2)}, x^{(3)} \} \in A_1^1 \). In this case, since the point \( x \) was already being processed at worker \( w_1 \) at time \( t \), hence no storage update is necessary at time \( t+1 \) for this point across workers.

- **Case 2:** \( x = \{ x^{(1)}, x^{(3)} \} \in A_2^1, x^{(1)} \in A_{2,1}^1, x^{(3)} \in A_{2,3}^1 \). After receiving \( x^{(3)} \) from the delivery phase, worker \( w_1 \) stores the full-point \( x \) in \( A_t^{1+1} \). For worker \( w_3 \), \( x^{(3)} \) leaves \( A_3^1 \) and enters \( A_{3,3}^{1+1} \), and we can notice that \( x^{(3)} \) remains within the excess storage of \( Z_{3,1}^{1+1} \). Worker \( w_2 \) removes the data point \( x \) from the processing batch \( A_t^{1+1} \), and stores \( x^{(1)} \) in \( A_{1,2}^{1+1} \) after relabelling it as \( x^{(2)} \) (now stored at the excess storage of \( w_2 \)), i.e., it simply moves one half of \( x \) in its excess storage.

- **Case 3:** \( x = \{ x^{(1)}, x^{(2)} \} \in A_3^1, x^{(1)} \in A_{3,1}^1, x^{(2)} \in A_{3,2}^1 \). This case is similar to case 2, where \( w_1 \) stores \( x \) in \( A_t^{1+1} \) and removes \( x^{(1)} \) from \( A_{3,1}^{1+1} \). Worker \( w_2 \) moves \( x^{(2)} \) into \( A_{1,2}^{1+1} \) instead of \( A_{3,2}^{1+1} \), and worker \( w_3 \) removes \( x \) from \( A_{3,1}^{1+1} \) and stores \( x^{(1)} \) (now labeled as \( x^{(3)} \)) in \( A_{3,3}^{1+1} \).

**Example 1 (N = 3, K = 3, S = 2N/3 = 2).** Let us now take a representative example depicted in Fig. 2 to illustrate our proposed data-delivery and storage-update phases for the corner point \( (\pi_3, \pi_3) \). Consider a system with \( K = 3 \) workers, and \( N = 3 \) data points, \( \{ x_1, x_2, x_3 \} \). We assume that storage per worker is \( S = 2N/3 = 2 \) points, i.e., each worker can store one extra data point in addition to the one under processing. We first clarify the color code used in this example to indicate the data point assigned to a certain worker labeled with these colors: blue for \( x_1 \), red for \( x_2 \), and yellow for \( x_3 \).

At time \( t \), consider the dataset is shuffled such that \( A_1^t = x_1 \), \( A_2^t = x_2 \), and \( A_3^t = x_3 \). The corresponding storage placement at time \( t \) in Fig. 2(a) is as follows: After using half the storage to store the desired data point (which can be depicted in this example as the desired color), each data point is divided equally among the unintended workers (depicted in this example as the unintended colors). For example, if we take the batch \( A_1^t = x_1 = \{ x_1^{(2)}, x_1^{(3)} \} \), worker \( w_2 \) stores \( A_{1,2}^t = x_1^{(2)} \), and worker \( w_3 \) stores \( A_{1,3}^t = x_1^{(3)} \).

At time \( t+1 \) in Fig. 2(b), the data is randomly shuffled again such that the new batches are: \( A_1^{t+1} = x_3 \), \( A_2^{t+1} = x_1 \), and \( A_3^{t+1} = x_2 \). We take this particular shuffle since it represents one of the possible worst cases, where every worker is assigned a completely different batch. According to the previous storage content shown in Fig. 2(a), worker \( w_1 \) already has \( x_3^{(1)} \) but still needs \( x_2^{(2)} \), which is stored at \( w_2 \) and \( w_3 \). Similarly, worker \( w_2 \) needs \( x_2^{(3)} \) which is stored at \( w_1 \) and \( w_3 \), and worker \( w_3 \) needs \( x_1^{(1)} \) which is stored at \( w_1 \) and \( w_2 \). Following the data delivery as in (25), the master node transmits:

\[
X_{(\pi_1, \pi_{t+1})} = x_3^{(2)} \oplus x_1^{(3)} \oplus x_2^{(1)}. \tag{26}
\]

Each worker has two out of these three subdivisions, therefore it can decode the remaining needed one. The rate of this transmission is \( R = \frac{2}{3} \), which is \( \frac{N}{K} \) where \( N = 3 \).

For the storage update in Fig. 2(b), we only discuss the changes in \( A_{1,1}^{t+1} \) and the corresponding \( A_{1,1}^{t+1} \) and \( A_{1,3}^{t+1} \). The storage update of the remaining parts can be done in a similar manner. From the delivery phase and the previous storage, worker \( w_1 \) gets \( x_3 \) (labeled yellow) and stores it in the processing half \( A_1^{t+1} \) (above the dotted line). Worker \( w_2 \) already has a part of \( A_{1,1}^{t+1}, x_2^{(3)} \) (labeled yellow), which was previously stored as \( A_{1,2}^{t+1} \). Therefore, it remains in the excess storage (below the dotted line) as \( A_{1,2}^{t+1} \). Worker \( w_3 \) already has \( x_3 \) previously labeled as \( A_{1,3}^t \), so it keeps in its excess storage the part that is not stored in \( A_{1,2}^{t+1} \), i.e., \( x_3^{(1)} \), to be stored in \( A_{1,3}^{t+1} \), after relabelling it to \( x_3^{(3)} \) (now stored in \( w_3 \)).

Using the achievability of the three corner points for \( K = 3 \), and Claim 1 we get an upper bound on \( R_{\text{worst-case}}(3, S) \) as

\[
R_{\text{worst-case}}(3, S) \leq \frac{2N}{6} - \frac{3S}{2}, \quad \frac{N}{3} \leq S \leq \frac{2N}{3}. \tag{27}
\]

**B. Converse for \( K = 3 \) workers**

We now present the information theoretic lower bounds for the three-worker case, which matches the above scheme. Following Remark 2 we first assume subsequent data shuffles at times \( \{ t, t+1, t+2 \} \) such that \( A_1^{t+1} = A_2^t \) and \( A_1^{t+2} = A_3^t \), then from the decodability constraint in (9), we have

\[
H(A_2^{t} | Z_{1}^{t}, X_{(\pi_1, \pi_{t+1})}) = H(A_3^{t} | Z_{1}^{t}, X_{(\pi_{t+1}, \pi_{t+2})}) = 0. \tag{28}
\]
Hence, in a similar proof to (22) we get
\[ H(A|Z_1^t, X_{(\pi, \pi+t-1)}, X_{(\pi+t, \pi+2)}) \]
\[ \leq H(A_1^t, A_2^t, A_3^t|Z_1^t, Z_1^{t+1}, X_{(\pi, \pi+t-1)}, X_{(\pi+t+1, \pi+2)}) \]
\[ \leq H(A_1^t|Z_1^t) + H(A_2^t|Z_1^t, X_{(\pi, \pi+t-1)}|A) \]
\[ \leq Sd + R_{(\pi, \pi+t-1)}^* d + \frac{S - N}{2} d, \quad (33) \]
where (a) follows from (1), (b), and (c) from (32), and due to the fact that \( I(A; B) = H(A) - H(A|B) = H(B) - H(B|A) \), (d) follows from (9) because \( Z_1^t, Z_2^t \) are all functions of the whole data-set \( A \), (e) from (9) where \( A_1^t + 1 = A_3^t \) must be decoded from \( Z_1^t \), and because \( A_1^t \) and \( A_2^t \) are stored within \( Z_2^t \), (f) because conditioning reduces entropy, (g) because after obtaining \( A_1^t, A_3^t, A_2^t \), the only remaining part in \( Z_2^t \) is \( A_3^t \), and finally (h) follows from (4), (7), and (12). From Remark 2 and by rearranging (33), we get the following bound
\[ R_{\text{worst-case}}(3, S) \geq \frac{7N}{6} - \frac{3S}{2}, \quad \frac{N}{3} \leq S \leq N. \quad (34) \]
Therefore, from (33), and (31), we get the following lower bound on \( R_{\text{worst-case}}(3, S) \):
\[ R_{\text{worst-case}}(3, S) \geq \left\{ \begin{array}{ll} \frac{7N}{6} - \frac{3S}{2}, & \frac{N}{3} \leq S \leq \frac{2N}{3} \\ \frac{N}{2} - \frac{S}{2}, & \frac{2N}{3} < S \leq N \end{array} \right. \quad (35) \]

Finally, the proof of Theorem 2 follows from (29) and (35).

VI. CONCLUSIONS

In this paper, we presented information theoretic formulation of the data shuffling problem, where we studied the tradeoff between the worst-case communication overhead and the storage available at the worker nodes. We completely characterized the optimal worst-case communication for \( K = 2 \), and \( K = 3 \) workers with any storage capacity, where we leveraged excess storage and coding to minimize the communication overhead in subsequent data shuffling iteration. A systematic storage update and delivery scheme was presented, which preserves the structural properties of the storage across workers. Generalizing these results for any number of workers (\( K > 3 \)) is part of our ongoing work.

REFERENCES

[1] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, “Spark: Cluster computing with working sets,” in Proceedings of the 2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud), 2010.

[2] J. Dean and S. Ghemawat, “MapReduce: Simplified data processing on large clusters,” in Proceedings of the 6th Symposium on Operating System Design and Implementation (OSDI), 2004.

[3] O. Shamir, “Without-replacement sampling for stochastic gradient methods: Convergence results and application to distributed optimization,” CoRR, vol. abs/1603.00570v2, 2016. [Online]. Available: https://arxiv.org/abs/1603.00570v2

[4] M. Gürbüzbalaban, A. Ozdaglar, and P. Parrilo, “Why random reshuffling beats stochastic gradient descent,” CoRR, vol. abs/1501.08560, 2015. [Online]. Available: https://arxiv.org/abs/1510.08560

[5] S. Ioffe and C. Szegedy, “Batch normalization accelerating deep network training by reducing internal covariate shift,” CoRR, vol. abs/1502.03167, 2015. [Online]. Available: https://arxiv.org/abs/1502.03167

[6] M. A. Maddah-Ali and U. Niesen, “Fundamental limits of caching,” IEEE Transactions on Information Theory, vol. 60, no. 5, pp. 2856–2867, Feb. 2015.

[7] S. Li, M. A. Maddah-Ali, and S. Avestimehr, “Coded MapReduce,” in Proceedings of the 53rd Annual Allerton conference on Communication, Control, and Computing, Monticello, IL, Sep. 2015.

[8] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, “Speeding up distributed machine learning using codes,” in Proceedings of Neural Information Processing Systems Conference (NIPS), Dec. 2015.