2018 56th Annual Allerton Conference on Communication, Control, and Computing (Allerton)
Allerton Park and Retreat Center
Monticello, IL, USA, October 2-5, 2018

Fundamental Limits of Distributed Data Shuffling

Kai Wan*, Daniela Tuninetti†, Mingyue Ji‡, Pablo Piantanida*,
*L2S CentraleSupélec-CNRS-Université Paris-Sud, France, {kai.wan, pablo.piantanida}@l2s.centralesupelec.fr
‡University of Illinois at Chicago, Chicago, USA, danielat@uic.edu
†University of Utah, Salt Lake City, USA, mingyue.ji@utah.edu

***Abstract*—Data shuffling of training data among different computing nodes (workers) has been identified as a core element to improve the statistical performance of modern large scale machine learning algorithms. Data shuffling is often considered one of the most significant bottlenecks in such systems due to the heavy communication load. Under a master-worker architecture (where a master has access to the entire dataset and only communications between the master and workers is allowed) coding has been recently proved to considerably reduce the communication load. This work considers a different communication paradigm referred to as *distributed data shuffling*, where workers, connected by a shared link, are allowed to communicate with one another while no communication between the master and workers is allowed. Under the constraint of uncoded cache placement, first a general coded distributed data shuffling scheme is proposed, which achieves the optimal communication load within a factor two; then, an improved scheme achieving the exact optimality for either large memory size or at most four workers in the system.**

I. Introduction

Recent years have witnessed the emergence of big data and machine learning with wide applications in both business and consumer worlds. To cope with such a large size/dimension of data and the complexity of machine learning algorithms, it is increasingly popular to use distributed computing platforms such as Amazon Web Services Cloud, Google Cloud, and Microsoft Azure services, where large scale distributed machine learning algorithms can be implemented. The approach of data shuffling has been identified as one of the core elements to improve the statistical performance of modern large scale machine learning algorithms. In particular, data shuffling is to re-shuffle the training data among all computing nodes (workers) once very few iterations determined by the learning algorithms. However, due to the huge communication cost, data shuffling cannot be often used and is considered as one of the main bottlenecks in such systems. To tackle this problem, under a master-worker setup, where the master has access to the entire dataset, coded data shuffling has been recently proposed [1] to significantly reduce the communication load between master and workers, which is the one of the main focuses in this research area. Nevertheless, it can be observed that data shuffling involves multiple iterations of training data such that the entire set of training data has been stored across all works in the previous iteration. Hence, if workers are allowed to communicate with each other,[1] the communication bottleneck between master and workers can be completely eliminated, instead, the data shuffling can be implemented distributedly among workers. This can be advantageous if the transmission capacity among workers is much higher than that between the master and workers, and the communication load between this two setups are similar. In this work, we consider such a *distributed data shuffling* framework, where all workers, connected by a shared link network, are allowed to communicate while no communication between the master and workers is allowed. In the following, we will review the literature of coded data shuffling problems and introduce the *distributed coded shuffling* framework studied in this paper.

A. Shared-link Data Shuffling Problem

The coded data shuffling problem was originally proposed in [1] in a *master-worker shared-link model*, where a master with the access to the whole library is connected to K workers. Each shuffling epoch is divided into *data shuffling* and *storage update* phases. In the data shuffling phase, a subset of files are assigned to each worker and each worker should recover these files from the broadcasted packets of the master and its own cached content from the last epoch. In the storage update phase, each worker should store the assigned files in its cache and store some information of other files in its extra memory also based on the broadcasted packets from the master and the its own cached content from the last epoch. The authors in [1] firstly proposed a coded scheme to transmit packets from the master based on the constraint that each worker fills its extra memory independently at random, which leads a coded data shuffling gain of a factor of $O(\mathsf{K})$ in terms of communication load compared to the uncoded data shuffling scheme.

The coded data shuffling problem with coordinated uncoded cache placement was originally proposed in [3], [4] to minimize communication load for the worst-case shuffles. The optimal schemes under the constraint of uncoded cache placement for the cases where there is no extra memory for each worker or there are less than 3 workers in the systems were proposed in [3], [4]. Inspired by the achievable and converse bounds for the shared-link caching problem in [5]–[7], the authors then proposed a general coded data shuffling scheme in [8], which was shown to be order optimality within a factor if 2 under the constraint of uncoded cache placement. In [8], the same authors improved the performance of the general coded shuffling scheme in some memory regimes using a different coded shuffling scheme, which was shown to be optimal under the constraint of uncoded cache placement in such specific memory regimes.

[1] In practice, workers communicate with each other as described in [2].

978-1-5386-6596-1/18/$31.00 ©2018 IEEE

Recently, the authors in [9] proposed a linear coding based on interference alignment, which achieves the optimal worst-case total communication load under the constraint of uncoded cache placement. In addition, under the constraint of uncoded cache placement, the proposed coded data shuffling scheme was shown to be optimal for any shuffles when the number of files is equal to the number of workers.

B. Distributed Data Shuffling Problem

An important limitation of the shared-link framework is the assumption that workers can only receive packets from the master. Since the entire data set is stored distributedly across all workers in the previous epoch of the distributed learning algorithm, the master may not be needed in the data shuffling phase if workers can communicate with each other (e.g., [2]). In this paper, we consider the *distributed data shuffling problem*, where only communications among workers is allowed. This means that in the data shuffling phase, each worker broadcasts well designed coded packets based on its cached content in the last epoch. Workers takes turn in transmitting. Transmissions are assumed received error-free by any other worker, as in the shared-link model. The objective is to design a joint data shuffling and storage update phases in order to minimize the total communication load across all the workers in the worst-case scenario.

C. Relation to other Problems

The coded distributed data shuffling problem considered in this paper is related to the *coded device-to-device (D2D) caching problem* [10] and the *coded distributed computing problem* [11] – see also Remark 1.

The D2D caching problem includes *placement* and *delivery* phases. In the placement phase, each user stores some contents in its cache without knowing the later demands. In the delivery phase, each user requests one file. According to users' demands, each user broadcasts well designed (coded) packets based on its cached content to all other users via a shared link. An order-optimal two-phase linear coding scheme was proposed in [10].

Recently, the scheme for the coded D2D caching problem in [10] has been extended to the coded distributed computing problem [11], which consists of two stages named *Map* and *Reduce*. In the Map stage, workers compute a fraction of intermediate computation values using local input data points according to the designed Map functions. In the Reduce stage, cording to the designed Reduce functions, workers exchange among each other a set of well designed (coded) intermediate computation values, based on its local intermediate computation values, in order to compute the final output results. The coded distributed computing problem can be seen as a coded D2D caching problem under the constraint of uncoded and symmetric cache placement, where symmetry here means that each user uses the same storing function for each file. A converse bound was proposed in [11] to show that the proposed coded distributed computing scheme is exactly optimal in terms of communication load.

Compared to the coded D2D caching and the coded distributed computing problems, the distributed data shuffling problem differs as follows. On the one hand, an novel constraint on the cached contents for the workers is present (because each worker must store all bits of each assigned file in the previous epoch, which breaks the symmetry of the cached contents across files of the other settings). On the other hand, each worker also needs to update its cache based on the received packets and its own cache content in the last epoch, while in the other problems, each user fills its cache based on the whole library, after which the cache content is kept fixed.

D. Contributions and Paper Organization

In this paper, we study the distributed data shuffling problem under the constraint of uncoded cache placement where workers broadcast packets among themselves. In Appendix A, we propose a novel converse bound under the constraint of uncoded cache placement. By extending the shared-link data shuffling scheme to distributed model, in Appendix C we propose a general coded distributed data shuffling scheme. In Appendix D, we then improve on the above scheme for $\mathsf{M} = \mathsf{N}/\mathsf{K}$ and for $\mathsf{M} \in \big[\frac{(\mathsf{K}-2)\mathsf{N}}{\mathsf{K}}, \mathsf{N}\big]$, where M is the memory size per worker, K is the number of workers, and N is the cardinality of the dataset. We prove that the improved scheme is optimal under the constraint of uncoded cache placement for the above memory regimes. Based on this result, we can also characterize the exact optimality under the constraint of uncoded cache placement when $\mathsf{K} \leq 4$. Finally, we prove that the proposed schemes are generally order optimal under the constraint of uncoded cache placement within a factor of 2.

II. System Model

We use the following notation convention. Calligraphic symbols denote sets, bold symbols denote vectors, and sans-serif symbols denote system parameters. We use $|\cdot|$ to represent the cardinality of a set or the length of a vector; $[a:b] := \{a, a+1, \ldots, b\}$ and $[n] := [1, 2, \ldots, n]$; $\oplus$ represents bit-wise XOR.

The $(\mathsf{K}, \mathsf{q}, \mathsf{M})$ *distributed data shuffling problem* is defined as follows. There are $\mathsf{K} \in \mathbb{N}$ workers, each of which is charged to process and store $\mathsf{q} \in \mathbb{N}$ files from a library of $\mathsf{N} := \mathsf{K}\mathsf{q}$ files. Files are denoted as $(F_1, F_2, \ldots, F_{\mathsf{N}})$ and each file has B i.i.d. bits. Each worker has a local cache of size $\mathsf{M}\mathsf{B}$ bits, for $\mathsf{M} \in [\mathsf{q}, \mathsf{N}]$. The workers are interconnected through a noiseless multicast network. The computation process occurs over T time slots. At the end of time slot $t-1$, $t \in [\mathsf{T}]$, the content of the local cache of user $k \in [\mathsf{K}]$ is denoted by Z_k^{t-1}; the content of all caches is denoted by $Z^{t-1} := (Z_1^{t-1}, Z_2^{t-1}, \ldots, Z_{\mathsf{K}}^{t-1})$. At the beginning time slot $t \in [\mathsf{T}]$, the N files are partitioned into K disjoint batches, each containing q files. The files indexed by $\mathcal{A}_k^t \subseteq [N]$ are assigned to worker $k \in [\mathsf{K}]$ who must stored them in its local cache by the end of time slot $t \in [\mathsf{T}]$. The file partition in time slot $t \in [\mathsf{T}]$ is denoted by $\mathcal{A}^t = (\mathcal{A}_1^t, \mathcal{A}_2^t, \ldots, \mathcal{A}_{\mathsf{K}}^t)$ and must satisfy

$$|\mathcal{A}_k^t| = \mathsf{q}, \ \forall k \in [\mathsf{K}], \tag{1a}$$

$$\mathcal{A}_{k_1}^t \cap \mathcal{A}_{k_2}^t = \emptyset,\ \forall (k_1, k_2) \in [\mathsf{K}]^2 : k_1 \neq k_2, \tag{1b}$$

$$\cup_{k \in [\mathsf{K}]} \mathcal{A}_k^t = [\mathsf{N}] \quad \text{(file partition)}. \tag{1c}$$

The following two-phase scheme allows users to store the requested files.

Data Shuffling Phase: Given global knowledge of the stored content Z^{t-1} at all workers, and of the data shuffle from $\mathcal{A}^{t-1}$ to $\mathcal{A}^t$ (indicated as $\mathcal{A}^{t-1} \to \mathcal{A}^t$) worker $k \in [\mathsf{K}]$ broadcasts a message X_k^t of $\mathsf{B}\mathsf{R}_k^{\mathcal{A}^{t-1} \to \mathcal{A}^t}$ bits to all of the other workers, where X_k^t is based only on the its local cache content Z_k^{t-1}, that is,

$$H\Big(X_k^t | Z_k^{t-1}\Big) = 0 \quad \text{(encoding)}. \tag{2}$$

The collection of all sent messages is denoted by $X^t := (X_1^t, X_2^t, \ldots, X_{\mathsf{K}}^t)$. Each worker $k \in [\mathsf{K}]$ must recover all files indexed by $\mathcal{A}_k^t$ from the sent messages X^t and its local cache content Z_k^{t-1}, that is,

$$H\Big(\big(F_i : i \in \mathcal{A}_k^t\big) | Z_k^{t-1}, X^t\Big) = 0 \quad \text{(decoding)}. \tag{3}$$

Storage Update Phase: Each worker $k \in [\mathsf{K}]$ must also update its local cache based on the sent messages X^t and its local cache content Z_k^{t-1}, that is,

$$H\Big(Z_k^t | Z_k^{t-1}, X^t\Big) = 0 \quad \text{(cache update)}, \tag{4}$$

by placing in it all the recovered files, that is,

$$H\Big(\big(F_i : i \in \mathcal{A}_k^t\big) | Z_k^t\Big) = 0, \quad \text{(cache content)}. \tag{5}$$

Moreover, the local cache has limited size bounded by

$$H\Big(Z_k^t\Big) \le \mathsf{M}\mathsf{B},\ \forall k \in [\mathsf{K}], \quad \text{(cache size)}. \tag{6}$$

If there is "excess storage," that is, if $\mathsf{M} > \mathsf{q}$, besides the files indexed by $\mathcal{A}_k^t$, worker $k \in [\mathsf{K}]$ can store in its local cache parts of the files indexed by $[\mathsf{N}] \backslash \mathcal{A}_k^t$. The "excess storage" placement is said to be *uncoded* if each worker simply copies bits from the files in its local cache.

Objective: The objective is to minimize the *worst-case total communication load*, or just load for short in the following, among all possible consecutive data shuffles, that is

$$\mathsf{R}^\star := \min_{Z_k^t : k \in [\mathsf{K}]} \max_{\mathcal{A}^{t-1}, \mathcal{A}^t} \sum_{k \in [\mathsf{K}]} \mathsf{R}_k^{\mathcal{A}^{t-1} \to \mathcal{A}^t}. \tag{7}$$

The minimum load under the constraint of uncoded cache placement is denoted by $\mathsf{R}_{\mathrm{u}}^\star$. In general, $\mathsf{R}_{\mathrm{u}}^\star \ge \mathsf{R}^\star$.

Remark 1 (Distributed Data Shuffling vs D2D Caching)**.** The distributed D2D caching problem studied in [10] differs from our setting as follows: (i) in the distributed data shuffling problem one has the constraint on the cached content in (5), which imposes that each worker stores the whole requested files and which is not present in the D2D caching problem, and (ii) in the D2D caching problem each user fills its local cache by accessing the whole library, while in the distributed data shuffling problem, each worker updates its local cache based on the received packets in the current time slot and its cached content in the previous time slot as in (4). Because of these differences, achievable and converse bounds for the distributed data shuffling problem can not be obtained by trivial renaming of variables in the D2D caching problem. Finally, we note that the distributed computing problem in [11] is a special case of the D2D caching problem when one restricts attention to uncoded and symmetric (across files) cache placement. □

Remark 2 (Distributed vs Shared-Link Data Shuffling)**.** Data shuffling was originally proposed in [8] for the shared-link model, where there exists one master node equipped with all the files that broadcasts packets to all workers, that is, the K encoding functions in (2) are replaced by $H(X^t | Z^{t-1}) = 0$ where X^t is broadcasted by the master to all the workers. We revise next some key results from [8], which will be used in the following sections. We shall use the subscripts "u,sl,conv" and "u,sl,ach" for converse (conv) and achievable (ach) bounds, respectively, for the shared-link problem (sl) with uncoded cache placement (u).

For a $(\mathsf{K}, \mathsf{q}, \mathsf{M})$ shared-link data shuffling system, the worst-case total communication load under the constraint of uncoded cache placement is lower bounded by the lower convex envelope of the following memory-load pairs [8, Thm.2]

$$\left(\frac{\mathsf{M}}{\mathsf{q}} = m,\ \frac{\mathsf{R}}{\mathsf{q}} = \frac{\mathsf{K} - m}{m}\right)_{\text{u,sl,conv}},\ \forall m \in [\mathsf{K}]. \tag{8}$$

It was shown in [9, Thm.4] that the converse bound in (8) can be achieved by a scheme that uses linear network coding and interference elimination; such a scheme is however not extendable straightforwardly to the distributed setting considered in this paper because it heavily builds on "*centralized* interference alignment"-type ideas. A similar optimality result was shown in [8, Thm.4] but only for $m \in \{1, \mathsf{K}-2, \mathsf{K}-1\}$; note that $m = \mathsf{K}$ is trivial.

In [8] it was showed that the lower convex envelope of the following memory-load pairs is achievable with uncoded cache placement [8, Thm.1]

$$\left(\frac{\mathsf{M}}{\mathsf{q}} = 1 + g\frac{\mathsf{K}-1}{\mathsf{K}},\ \frac{\mathsf{R}}{\mathsf{q}} = \frac{\mathsf{K} - g}{g+1}\right)_{\text{u,sl,ach}},\ \forall g \in [0 : \mathsf{K}]. \tag{9}$$

The bound in (9) is order optimal to within a factor $\frac{\mathsf{K}}{\mathsf{K}-1} \le 2$ under the constraint of uncoded cache placement [8, Thm.3].

The scheme that achieves the load in (9) works as follows. Fix $g \in [0 : \mathsf{K}]$ and divide each file into $\binom{\mathsf{K}}{g}$ non-overlapping and equal-length subfiles of length $\mathsf{B}/\binom{\mathsf{K}}{g}$ bits. Let $F_i = (F_{i,\mathcal{W}} : \mathcal{W} \subseteq [\mathsf{K}] : |\mathcal{W}| = g),\ i \in [\mathsf{N}]$.

Storage Update Phase: Worker $k \in [\mathsf{K}]$ cashes all the subfiles of the required q files indexed by $\mathcal{A}_k^t$, and in addition all subfiles $F_{i,\mathcal{W}}$ with $i \in [\mathsf{N}] \backslash \mathcal{A}_k^t$ and $k \in \mathcal{W}$, thus

$$\mathsf{M} = \mathsf{q} + (\mathsf{N} - \mathsf{q})\frac{\binom{\mathsf{K}-1}{g-1}}{\binom{\mathsf{K}}{g}} = \Big(1 + g\frac{\mathsf{K}-1}{\mathsf{K}}\Big)\mathsf{q}. \tag{10}$$

Data Shuffling Phase: After the storage update phase just described, the new assignment $\mathcal{A}^{t+1}$ is revealed. Let

$$\mathcal{A}_{k,\mathcal{W}}^{t+1} := \{F_{i,\mathcal{W}} : i \in \mathcal{A}_k^{t+1} \setminus \mathcal{A}_k^t\}, \forall \mathcal{W} \subseteq [\mathsf{K}] : |\mathcal{W}| = g\}, \tag{11}$$

represent the subfiles required by worker $k \in [\mathsf{K}]$ and not present in its local cache. Note that $|\mathcal{A}^{t+1}_{k,\mathcal{W}}| \leq \mathsf{B}\frac{\mathsf{q}}{\binom{\mathsf{K}}{g}}$, with equality (i.e., worst case scenario) if and only if $\mathcal{A}^{t+1}_k \cap \mathcal{A}^t_k = \emptyset$. To allow the workers to recover their missing subfiles, the central server broadcasts X^{t+1} given by

$$X^{t+1} = (W^{t+1}_{\mathcal{J}} : \mathcal{J} \subseteq [\mathsf{K}] : |\mathcal{J}| = g+1) \text{ where} \tag{12}$$

$$W^{t+1}_{\mathcal{J}} := \oplus_{k \in \mathcal{J}} \mathcal{A}^{t+1}_{k,\mathcal{J}\setminus\{k\}}. \tag{13}$$

Since worker $k \in \mathcal{J}$ requests $\mathcal{A}^{t+1}_{k,\mathcal{J}\setminus\{k\}}$ and has cached all the remaining subflies in $W^{t+1}_{\mathcal{J}}$ defined in (13), it can recover $\mathcal{A}^{t+1}_{k,\mathcal{J}\setminus\{k\}}$ from $W^{t+1}_{\mathcal{J}}$ and thus all its missing subfiles from X^{t+1}. Hence, the worst-case total communication load is

$$\mathsf{R} = \mathsf{q}\frac{\binom{\mathsf{K}}{g+1}}{\binom{\mathsf{K}}{g}} = \mathsf{q}\frac{\mathsf{K}-g}{g+1}. \tag{14}$$

This concludes the description of the scheme in [8]. □

III. Main Results

We shall use the subscripts "u, dist,conv" and "u,dist,ach" for converse (conv) and achievable (ach) bounds, respectively, for the distributed problem (dist) with uncoded cache placement (u). In the following, Theorem 1 gives a converse bound for the distributed data shuffling problem under the constraint of uncoded cache placement (proof in Appendix A):

Theorem 1 (Converse)**.** *For a* $(\mathsf{K},\mathsf{q},\mathsf{M})$ *distributed data shuffling system, the worst-case load under the constraint of uncoded cache placement is lower bounded by the lower convex envelope of the following memory-load pairs*

$$\left(\frac{\mathsf{M}}{\mathsf{q}} = m, \frac{\mathsf{R}}{\mathsf{q}} = \frac{\mathsf{K}-m}{m}\frac{\mathsf{K}}{\mathsf{K}-1}\right)_{\text{u, dist,conv}}, \ \forall m \in [\mathsf{K}]. \tag{15}$$

The proof of Theorem 1 is inspired by the induction method proposed in [11, Thm.1] for the distributed computing problem. However, there are two main differences in our proof: (i) we need to account for the additional constraint on the cached content in (5), and (ii) our cache placement is not restricted to be symmetric (across files) by the problem definition. We note, by comparing the converse bounds in (8) and in (15), that the "price" of distributed communication under the constraint of uncoded cache placement could be a factor of $\frac{\mathsf{K}}{\mathsf{K}-1} \leq 2$.

By extending the shared-link data shuffling scheme in (9) (see Remark 2) to our distributed setting, the achievable worst-case load is given by the following theorem (proof in Appendix C):

Theorem 2 (Achievablity)**.** *For a* $(\mathsf{K},\mathsf{q},\mathsf{M})$ *distributed data shuffling system, the worst-case load under the constraint of uncoded cache placement is upper bounded by the lower convex envelope of the following memory-load pairs*

$$\left(\frac{\mathsf{M}}{\mathsf{q}} = 1 + g\frac{\mathsf{K}-1}{\mathsf{K}}, \frac{\mathsf{R}}{\mathsf{q}} = \frac{\mathsf{K}-g}{g}\right)_{\text{u, dist,ach}}, \ \forall g \in [\mathsf{K}]. \tag{16}$$

A limitation of the achievable scheme in Theorem 2 is that, in time slot $t+1$, each user $k \in [\mathsf{K}]$ does not leverage the cached subfiles $(F_{i,\mathcal{W}} : i \in \mathcal{A}^t_k, k \notin \mathcal{W})$. We overcome this limitation by the scheme in Theorem 3 (proof in Appendix D):

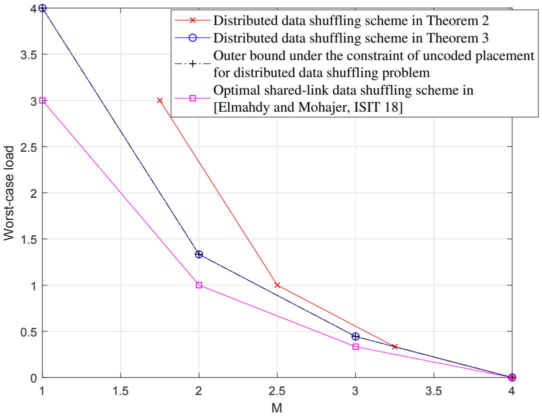

Fig. 1: The memory-load trade-off for a distributed data shuffling problem with $\mathsf{K} = \mathsf{N} = 4$.

Theorem 3 (Exact Optimality 1)**.** *For a* $(\mathsf{K},\mathsf{q},\mathsf{M})$ *distributed data shuffling system, the worst-case load under the constraint of uncoded cache placement in Theorem 1 is achievable for* $m \in \{1, \mathsf{K}-2, \mathsf{K}-1\}$*; the case* $m = \mathsf{K}$ *is trivial.*

We note that Theorem 3 is neither an extension of [8, Thm.4] nor of [9, Thm.4] from the shared-link to the distributed setting. As it will become clear from the details in Appendix D, our scheme has a simpler way to generate the multicast messages transmitted by the workers, and it applies to any shuffle, not just to the worst case one.

From Theorem 3 we can immediately conclude:

Corollary 1 (Exact Optimality 2)**.** *For a* $(\mathsf{K},\mathsf{q},\mathsf{M})$ *distributed data shuffling system, the lower bound on the worst-case load under the constraint of uncoded cache placement in Theorem 1 is achievable for* $\mathsf{K} \leq 4$.

Finally we have the following order optimality result (proof in Appendix E):

Theorem 4 (Order Optimality)**.** *For a* $(\mathsf{K},\mathsf{q},\mathsf{M})$ *distributed data shuffling system under the constraint of uncoded cache placement, the achievable schemes in Theorem 2 achieves the converse bound in Theorem 1 to within a factor* 2.

We conclude this section with some numerical results. Fig. 1 plots our converse and achievable bounds on the worst-case load under the constraint of uncoded cache placement for the $(\mathsf{K},\mathsf{q},\mathsf{M}) = (4,1,\mathsf{M})$ distributed data shuffling problem. For comparison, we also plot the optimal shared-link memory-load tradeoff in (8). In this case, Theorem 1 is tight and the distributed data shuffling increases the communication load of the share-link case by a factor of $\frac{\mathsf{K}}{\mathsf{K}-1} = \frac{4}{3}$, under the constraint of uncoded cache placement. We conjecture that Theorem 1 is tight under the constraint of uncoded placement for any number of workers and memory size.

IV. Conclusions

In this paper, we introduced the distributed data shuffling problem and studied its fundamental limits. We proposed a converse bound under the constraint of uncoded cache placement and two achievable schemes. In general, under the constraint of uncoded cache placement, our schemes are optimal to within a factor two, and exactly optimal for certain large memory sizes or no more than four workers.

Acknowledgments: The work of K. Wan is supported by Labex DigiCosme and D. Tuninetti is supported by in part by NSF 1527059.

Appendix A
Proof of Theorem 1

The worker who must store file $F_i, i \in [\mathsf{N}]$, by the end of time slot t is denoted by u_i^t, i.e.,

$$\mathsf{u}_i^t := k, \text{ if } i \in \mathcal{A}_k^t. \tag{17}$$

We want to lower bound $\max_{\mathcal{A}^{t+1}} \sum_{k\in[\mathsf{K}]} \mathsf{R}_k^{\mathcal{A}^t \to \mathcal{A}^{t+1}}$ for a fixed $\mathcal{A}^t$.

After the storage update phase in time slot t, without loss of generality we can divide each file into subfiles as

$$F_i = \big(F_{i,\mathcal{W}} : \mathcal{W} \subseteq [\mathsf{K}] \setminus \{\mathsf{u}_i^t\}\big), \tag{18}$$

where $F_{i,\mathcal{W}}$ represents the bits of F_i exclusively cached by workers in $\mathcal{W} \cup \{\mathsf{u}_i^t\}$ at the end of time slot t. For each file F_i, let $F_{i,\mathcal{W}} = \emptyset$ if $\mathsf{u}_i^t \in \mathcal{W}$. Note that, as opposed to D2D caching and distributed computation, not all the subfiles exist for each file, i.e., the division into subfiles is asymmetric across files.

At time slot $t+1$, we first consider a permutation of $[\mathsf{K}]$ denoted by $(d_1, \ldots, d_\mathsf{K})$, where $d_k \neq k$ for each $k \in [\mathsf{K}]$. We let $\mathcal{A}_k^{t+1} = \mathcal{A}_{d_k}^t$. Obviously, the worst-case load is not less than the load in this case. We define

$$\mathcal{V}_\mathcal{S} := \{k \in \mathcal{S} : d_k \in \mathcal{S}\}, \ \forall \mathcal{S} \subseteq [\mathsf{K}] : |\mathcal{S}| > 0 \tag{19}$$

where $\mathcal{V}_\mathcal{S}$ represents the set of workers $k \in \mathcal{S}$ whose demanded files indexed by $\mathcal{A}_k^{t+1} = \mathcal{A}_{d_k}^t$ should be cached by some workers in $\mathcal{S}$ by the end of time slot t. For example, if $\mathsf{K} = 4$ and $(d_1, \ldots, d_\mathsf{K}) = (2,3,4,1)$, we have $\mathcal{V}_{\{2,3\}} = \{2\}$ because $d_2 = 3$, $d_3 = 4$ and thus the requested files of user 2 in time slot $t+1$ are requested by user $d_2 = 3 \in \{2,3\}$ in time slot t. Similarly, we have $\mathcal{V}_{\{2,4\}} = \emptyset$ and $\mathcal{V}_{\{1,2,4\}} = \{1,4\}$. In addition, we define

$$\begin{aligned}
X_\mathcal{S}^{t+1} &:= \{X_k^{t+1} : k \in \mathcal{S}\} & (20)\\
Y_\mathcal{S}^{t+1} &:= \big\{F_i : i \in \cup_{k\in\mathcal{S}} \mathcal{A}_k^{t+1}\big\} \cup \{Z_k^t : k \in \mathcal{S}\} & (21)\\
&= \big\{F_i : i \in \cup_{k\in\mathcal{S}} (\mathcal{A}_k^{t+1} \cup \mathcal{A}_k^t)\big\} & (22)\\
&\cup \big\{F_{i,\mathcal{W}} : i \in [\mathsf{N}], \mathcal{W} \subseteq ([\mathsf{K}] \setminus \{\mathsf{u}_i^t\}), \mathcal{W} \cap \mathcal{S} \neq \emptyset\big\} & (23)
\end{aligned}$$

where $Y_\mathcal{S}^{t+1}$ in (21) represents the bits either cached or requested by any worker in $\mathcal{S}$.

With the above definitions, we have the following lemma (whose poof can be found in Appendix B and was inspired by the induction argument in [11]):

Lemma 1. *For each set non-empty $\mathcal{S} \subseteq [\mathsf{K}]$, we have*

$$H(X_\mathcal{S}^{t+1} | Y_{[\mathsf{K}]\setminus\mathcal{S}}^{t+1}) \geq \sum_{g=0}^{|\mathcal{S}|-1} \sum_{k\in\mathcal{V}_\mathcal{S}} \sum_{i\in\mathcal{A}_k^{t+1}} \sum_{\mathcal{W}\subseteq\mathcal{S}\setminus\{k,\mathsf{u}_i^t\}:|\mathcal{W}|=g} \frac{|F_{i,\mathcal{W}}|}{g+1}. \tag{24}$$

Lemma 1 is the key novel contribution of our proof. The bound in (24) can be thought of as follows: $H(X_\mathcal{S}^{t+1} | Y_{[\mathsf{K}]\setminus\mathcal{S}}^{t+1})$ is lower bounded only by requested subfiles by the workers in $\mathcal{V}_\mathcal{S}$ (instead of in $\mathcal{S}$ as in the distributed computing problem [11]) because each requested file by the workers in $\mathcal{S} \setminus \mathcal{V}_\mathcal{S}$ was requested in the previous time slot by some workers in $[\mathsf{K}] \setminus \mathcal{S}$ because of the cache constraint in (5) and the definition of $\mathcal{V}_\mathcal{S}$ in (19).

From Lemma 1 with $\mathcal{S} = [\mathsf{K}]$, we have

$$H(X_{[\mathsf{K}]}^{t+1}) \geq \sum_{g=0}^{\mathsf{K}-1} \sum_{k\in[\mathsf{K}]} \sum_{i\in\mathcal{A}_k^{t+1}} \sum_{\mathcal{W}\subseteq([\mathsf{K}]\setminus\{k,\mathsf{u}_i^t\}):|\mathcal{W}|=g} \frac{|F_{i,\mathcal{W}}|}{g+1}. \tag{25}$$

We next consider all the permutations $(d_1, \ldots, d_\mathsf{K})$ of $[\mathsf{K}]$ where $d_k \neq k$ for each $k \in [\mathsf{K}]$, and sum together the inequalities in the form of (25). For an integer $g \in [0 : \mathsf{K}-1]$, by the symmetry of the problem, the subfiles $F_{i,\mathcal{W}}$ where $i \in [\mathsf{N}]$, $\mathcal{W} \subseteq [\mathsf{K}] \setminus \{\mathsf{u}_i^t\}$ and $|\mathcal{W}| = g$ appear the same number of times in the final sum. In addition, the total number of these subfiles in general is $\mathsf{N}\binom{\mathsf{K}-1}{g}$ and the total number of such subfiles in each inequality in the form of (25) is $\mathsf{N}\binom{\mathsf{K}-2}{g}$. So we obtain

$$\begin{aligned}
\mathsf{R}_\mathsf{u}^\star &\geq \sum_{g=0}^{\mathsf{K}-1} \sum_{i\in[\mathsf{N}]} \sum_{\mathcal{W}\subseteq([\mathsf{K}]\setminus\{\mathsf{u}_i^t\}):|\mathcal{W}|=g} \frac{\binom{\mathsf{K}-2}{g}}{(g+1)\binom{\mathsf{K}-1}{g}} |F_{i,\mathcal{W}}| & (26)\\
&= \sum_{g=0}^{\mathsf{K}-1} x_g \mathsf{N} \frac{1 - g/(\mathsf{K}-1)}{g+1} & (27)\\
&= \sum_{g=0}^{\mathsf{K}-1} x_g \frac{\mathsf{K}-(g+1)}{g+1} \frac{\mathsf{K}}{\mathsf{K}-1} \mathsf{q}, & (28)
\end{aligned}$$

where we defined x_g as the total length of the subfiles cached by $g+1$ users normalized by NB,

$$0 \leq x_g := \sum_{i\in[\mathsf{N}]} \sum_{\mathcal{W}\subseteq([\mathsf{K}]\setminus\{\mathsf{u}_i^t\}):|\mathcal{W}|=g} \frac{|F_{i,\mathcal{W}}|}{\mathsf{NB}}, \tag{29}$$

which must satisfy

$$\sum_{g\in[0:\mathsf{K}-1]} x_g = 1, \text{ (total size of all files)}, \tag{30}$$

$$\sum_{g\in[0:\mathsf{K}-1]} g x_g + 1 \leq \frac{\mathsf{KM}}{\mathsf{N}}, \text{ (total cache size)}. \tag{31}$$

We use the method which we developped in [6] to bound $\mathsf{R}_\mathsf{u}^\star$ from (27) under the constraints in (30) and (31). For each integer $p \in [0 : \mathsf{K}-1]$, we multiply (30) by $\mathsf{N}\frac{2-p/(\mathsf{K}-1)}{p+2}$ to obtain

$$\sum_{g=0}^{\mathsf{K}-1} \frac{2-p/(\mathsf{K}-1)}{p+2}\mathsf{N}x_g = \frac{2-p/(\mathsf{K}-1)}{p+2}\mathsf{N}, \tag{32}$$

and we multiply (31) by $-\mathsf{N}\frac{1+1/(\mathsf{K}-1)}{(p+1)(p+2)}$ to have

$$\sum_{g=0}^{\mathsf{K}-1} -\mathsf{N}\frac{1+1/(\mathsf{K}-1)}{(p+1)(p+2)} g x_g \geq -\frac{1+1/(\mathsf{K}-1)}{(p+1)(p+2)}(\mathsf{KM}-\mathsf{N}). \tag{33}$$

We put (32) and (33) into (27) to obtain,

$$\begin{aligned}\mathsf{R}^{\star}_{\mathrm{u}} \geq & \sum_{g=0}^{\mathsf{K}} \frac{(p-g)(p-g+1)(1+\frac{1}{\mathsf{K}-1})}{(g+1)(p+1)(p+2)}\mathsf{N}x_g && (34)\\ & -\frac{\mathsf{K}+\mathsf{K}/(\mathsf{K}-1)}{(p+1)(p+2)}\mathsf{M} + \frac{2p+3-\frac{p^2+p-1}{\mathsf{K}-1}}{(p+1)(p+2)}\mathsf{N} && (35)\\ \geq & -\frac{\mathsf{K}+\mathsf{K}/(\mathsf{K}-1)}{(p+1)(p+2)}\mathsf{M} + \frac{2p+3-\frac{p^2+p-1}{\mathsf{K}-1}}{(p+1)(p+2)}\mathsf{N}. && (36)\end{aligned}$$

Hence, for each integer $p \in [0 : \mathsf{K}-1]$, the bound in (36) becomes linear in terms of M. When $\mathsf{M} = \mathsf{q}(p+1)$, from (36) we have $\mathsf{R}^{\star}_{\mathrm{u}} \geq \frac{1-p/(\mathsf{K}-1)}{p+1}\mathsf{N}$. When $\mathsf{M} = \mathsf{q}(p+2)$, from (36) we have $\mathsf{R}^{\star}_{\mathrm{u}} \geq \frac{1-(p+1)/(\mathsf{K}-1)}{p+2}\mathsf{N}$. In conclusion, we prove that $\mathsf{R}^{\star}_{\mathrm{u}}$ is lower bounded by the memory-sharing of the points $\left(\mathsf{M} = \mathsf{q}(g+1), \mathsf{R} = \mathsf{q}\frac{1-g/(\mathsf{K}-1)}{g+1}\right)$, where $g \in [0 : \mathsf{K}-1]$. This proves Theorem 1.

Appendix B
Proof of Lemma 1

This lemma is proved by induction, inspired by [11].

Case $|\mathcal{S}| = 1$: If $\mathcal{S} = \{k\}$ where $k \in [\mathsf{K}]$, we have that $\mathcal{V}_{\{k\}} = \emptyset$ (by $d_k \neq k$ because of the chosen permutations) and thus the RHS of (24) is 0; thus (24) holds for $|\mathcal{S}| = 1$ because entropy is non-negative.

Case $|\mathcal{S}| \leq s$: Assume that (24) holds for all non-empty $\mathcal{S} \subseteq [\mathsf{K}]$ where $|\mathcal{S}| \leq s$ for some integer $s \in [\mathsf{K}-1]$.

Case $|\mathcal{S}| = s+1$: Having assumed that the lemma holds for all $\mathcal{S} \subseteq [\mathsf{K}]$ where $|\mathcal{S}| \leq s$, we aim to show that for any set $\mathcal{J} \subseteq [\mathsf{K}]$ where $|\mathcal{J}| = s+1$, we have

$$H(X^{t+1}_{\mathcal{J}} | Y^{t+1}_{[\mathsf{K}]\setminus\mathcal{J}}) \geq \sum_{g=0}^{|\mathcal{J}|-1} \sum_{k\in\mathcal{V}_{\mathcal{J}}} \sum_{i\in\mathcal{A}^{t+1}_k} \sum_{\mathcal{W}\subseteq(\mathcal{J}\setminus\{k,\mathsf{u}^t_i\}):|\mathcal{W}|=g} \frac{|F_{i,\mathcal{W}}|}{g+1}. \tag{37}$$

From

$$\begin{aligned} & H(X^{t+1}_{\mathcal{J}} | Y^{t+1}_{[\mathsf{K}]\setminus\mathcal{J}}) \\ & = \frac{1}{|\mathcal{J}|}\sum_{k\in\mathcal{J}}\Big(H(X^{t+1}_{\mathcal{J}\setminus\{k\}} | X^{t+1}_k, Y^{t+1}_{[\mathsf{K}]\setminus\mathcal{J}}) + H(X^{t+1}_k | Y^{t+1}_{[\mathsf{K}]\setminus\mathcal{J}})\Big) \\ & \geq \frac{1}{|\mathcal{J}|}\Big(\sum_{k\in\mathcal{J}} H(X^{t+1}_{\mathcal{J}\setminus\{k\}} | X^{t+1}_k, Y^{t+1}_{[\mathsf{K}]\setminus\mathcal{J}}) + H(X^{t+1}_{\mathcal{J}} | Y^{t+1}_{[\mathsf{K}]\setminus\mathcal{J}})\Big), \end{aligned}$$

we have

$$\begin{aligned} & (|\mathcal{J}|-1) H(X^{t+1}_{\mathcal{J}} | Y^{t+1}_{[\mathsf{K}]\setminus\mathcal{J}}) && (38)\\ & \geq \sum_{k\in\mathcal{J}} H(X^{t+1}_{\mathcal{J}\setminus\{k\}} | X^{t+1}_k, Y^{t+1}_{[\mathsf{K}]\setminus\mathcal{J}}) && (39)\\ & \geq \sum_{k\in\mathcal{J}} H(X^{t+1}_{\mathcal{J}\setminus\{k\}} | Z^t_k, X^{t+1}_k, Y^{t+1}_{[\mathsf{K}]\setminus\mathcal{J}}) && (40)\\ & = \sum_{k\in\mathcal{J}} H(X^{t+1}_{\mathcal{J}\setminus\{k\}}, \{F_i : i \in \mathcal{A}^{t+1}_k\} | Z^t_k, X^{t+1}_k, Y^{t+1}_{[\mathsf{K}]\setminus\mathcal{J}}) && (41)\\ & = \sum_{k\in\mathcal{J}} H(\{F_i : i \in \mathcal{A}^{t+1}_k\} | Z^t_k, Y^{t+1}_{[\mathsf{K}]\setminus\mathcal{J}}) && (42)\\ & \quad + \sum_{k\in\mathcal{J}} H(X^{t+1}_{\mathcal{J}\setminus\{k\}} | \{F_i : i \in \mathcal{A}^{t+1}_k\}, Z^t_k, Y^{t+1}_{[\mathsf{K}]\setminus\mathcal{J}}) && (43)\\ & = \sum_{k\in\mathcal{J}} H(\{F_i : i \in \mathcal{A}^{t+1}_k\} | Z^t_k, Y^{t+1}_{[\mathsf{K}]\setminus\mathcal{J}}) && (44)\\ & \quad + \sum_{k\in\mathcal{J}} H\big(X^{t+1}_{\mathcal{J}\setminus\{k\}} | Y^{t+1}_{([\mathsf{K}]\setminus\mathcal{J})\cup\{k\}}\big), && (45) \end{aligned}$$

where (41) follows because $\{F_i : i \in \mathcal{A}^{t+1}_k\}$ is a function of (Z^t_k, X^{t+1}) (see decoding constraint in (3)), where (42)-(43) follow because X^{t+1}_k is a function of Z^t_k (see the encoding constraint in (2)), and (45) from the definition in (21).

Next we would like to bound (44) by using the independence of the subfiles and bound (45) by the induction assumption. More precisely, we first focus on (44). For each $k \in \mathcal{J}$, if $k \notin \mathcal{V}_{\mathcal{J}}$, we have $\{F_i : i \in \mathcal{A}^{t+1}_k\} \subseteq Y^{t+1}_{[\mathsf{K}]\setminus\mathcal{J}}$. So for each $k \in \mathcal{J}$, by independence of subfiles, we have

$$\begin{aligned} & H(\{F_i : i \in \mathcal{A}^{t+1}_k\} | Z^t_k, Y^{t+1}_{[\mathsf{K}]\setminus\mathcal{J}}) && (46)\\ & = \begin{cases} \sum_{g=0}^{|\mathcal{J}|-1} \sum_{i\in\mathcal{A}^{t+1}_k} \sum_{\mathcal{W}\subseteq(\mathcal{J}\setminus\{k,\mathsf{u}^t_i\}):|\mathcal{W}|=g} |F_{i,\mathcal{W}}|, & k \in \mathcal{V}_{\mathcal{J}} \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

and thus we rewrite (44) as

$$\sum_{k\in\mathcal{J}} H(\{F_i : i \in \mathcal{A}^{t+1}_k\} | Z^t_k, Y^{t+1}_{[\mathsf{K}]\setminus\mathcal{J}}) = \sum_{k\in\mathcal{V}_{\mathcal{J}}} \sum_{g\in[0:|\mathcal{J}|-1]} \sum_{i\in\mathcal{A}^{t+1}_k} \sum_{\mathcal{W}\subseteq(\mathcal{J}\setminus\{k,\mathsf{u}^t_i\}):|\mathcal{W}|=g} |F_{i,\mathcal{W}}|. \tag{47}$$

We then focus on (45). By the induction assumption,

$$\sum_{k\in\mathcal{J}} H\big(X^{t+1}_{\mathcal{J}\setminus\{k\}} | Y^{t+1}_{([\mathsf{K}]\setminus\mathcal{J})\cup\{k\}}\big) \geq \sum_{k\in\mathcal{J}} \sum_{u\in\mathcal{V}_{\mathcal{J}\setminus\{k\}}} \sum_{g=0}^{|\mathcal{J}|-2} \sum_{i\in\mathcal{A}^{t+1}_u} \sum_{\mathcal{W}\subseteq(\mathcal{J}\setminus\{k,u,\mathsf{u}^t_i\}):|\mathcal{W}|=g} \frac{|F_{i,\mathcal{W}}|}{g+1}. \tag{48}$$

In order to combine (47) with (48), both terms need to have the same form of summations. Let us focus on one worker $u' \in \mathcal{V}_{\mathcal{J}}$ and one subfile $F_{i',\mathcal{W}'}$ where $i' \in \mathcal{A}^{t+1}_{u'}$ and $\mathcal{W}' \subseteq \mathcal{J} \setminus \{u', \mathsf{u}^t_{i'}\} : |\mathcal{W}'| = g$. On the RHS of (48), for each $k \in \mathcal{J} \setminus (\mathcal{W}' \cup \{u'\} \cup \{\mathsf{u}^t_i\})$, it can be seen that $F_{i',\mathcal{W}'}$ appears once in the sum

$$\sum_{g\in[0:|\mathcal{J}|-2]} \sum_{u\in\mathcal{V}_{\mathcal{J}\setminus\{k\}}} \sum_{i\in\mathcal{A}^{t+1}_u} \sum_{\mathcal{W}\subseteq(\mathcal{J}\setminus\{k,u,\mathsf{u}^t_i\}):|\mathcal{W}|=g} \frac{|F_{i,\mathcal{W}}|}{g+1},$$

hence, the coefficient of $F_{i',\mathcal{W}'}$ in the RHS of (48) is $(|\mathcal{J}|-g-2)/(g+1)$. Thus, from (48), we have

$$\sum_{k\in\mathcal{J}} H\big(X^{t+1}_{\mathcal{J}\setminus\{k\}}|Y^{t+1}_{[\mathsf{K}]\setminus\mathcal{J})\cup\{k\}}\big) \tag{49}$$

$$\geq \sum_{u'\in\mathcal{V}_{\mathcal{J}}} \sum_{g\in[0:|\mathcal{J}|-2]} \sum_{i'\in\mathcal{A}^{t+1}_{u'}} \sum_{\mathcal{W}'\subseteq(\mathcal{J}\setminus\{u',\mathsf{u}^t_{i'}\}):|\mathcal{W}'|=g} \frac{|F_{i',\mathcal{W}'}|(|\mathcal{J}|-g-2)}{g+1} \tag{50}$$

$$= \sum_{u'\in\mathcal{V}_{\mathcal{J}}} \sum_{g\in[0:|\mathcal{J}|-1]} \sum_{i'\in\mathcal{A}^{t+1}_{u'}} \sum_{\mathcal{W}'\subseteq(\mathcal{J}\setminus\{u',\mathsf{u}^t_{i'}\}):|\mathcal{W}'|=g} \frac{|F_{i',\mathcal{W}'}|(|\mathcal{J}|-g-2)}{g+1}. \tag{51}$$

We take (47) and (51) into (45) to obtain,

$$H(X^{t+1}_{\mathcal{J}}|Y^{t+1}_{[\mathsf{K}]\setminus\mathcal{J}})$$

$$\geq \frac{1}{|\mathcal{J}|-1} \sum_{k\in\mathcal{V}_{\mathcal{J}}} \sum_{g=0}^{|\mathcal{J}|-1} \sum_{i\in\mathcal{A}^{t+1}_k} \sum_{\mathcal{W}\subseteq(\mathcal{J}\setminus\{k,\mathsf{u}^t_i\}):|\mathcal{W}|=g} |F_{i,\mathcal{W}}| + \frac{1}{|\mathcal{J}|-1} \sum_{k\in\mathcal{V}_{\mathcal{J}}} \sum_{g=0}^{|\mathcal{J}|-1} \sum_{i\in\mathcal{A}^{t+1}_k} \sum_{\mathcal{W}\subseteq(\mathcal{J}\setminus\{k,\mathsf{u}^t_i\}):|\mathcal{W}|=g} \frac{|F_{i,\mathcal{W}}|(|\mathcal{J}|-g-2)}{g+1} \tag{52a}$$

$$= \sum_{k\in\mathcal{V}_{\mathcal{J}}} \sum_{g\in[0:|\mathcal{J}|-1]} \sum_{i\in\mathcal{A}^{t+1}_k} \sum_{\mathcal{W}\subseteq(\mathcal{J}\setminus\{k,\mathsf{u}^t_i\}):|\mathcal{W}|=g} \frac{|F_{i,\mathcal{W}}|}{g+1}. \tag{52b}$$

Appendix C
Proof of Theorem 2

Storage Update Phase in Time Slot t: The storage update phase is the same as the scheme in Remark 2.

Data Shuffling Phase in Time Slot $t+1$: The data shuffling phase is inspired by the D2D caching delivery phase in [10]. Recall that $\mathcal{A}^{t+1}_{k,\mathcal{W}} := \{F_{i,\mathcal{W}} : i \in \mathcal{A}^{t+1}_k \setminus \mathcal{A}^t_k\}$, where $\mathcal{W} \subseteq [\mathsf{K}]$ and $|\mathcal{W}| = g$, and that $|\mathcal{A}^{t+1}_{k,\mathcal{W}}| \leq \mathsf{qB}/\binom{\mathsf{K}}{g}$, where the equality holds if and only if $\mathcal{A}^{t+1}_k \cap \mathcal{A}^t_k = \emptyset$. We divide each $\mathcal{A}^{t+1}_{k,\mathcal{W}}$ into $|\mathcal{W}|$ non-overlapping and equal-length pieces, $\mathcal{A}^{t+1}_{k,\mathcal{W}} = \{\mathcal{A}^{t+1}_{k,\mathcal{W}}(j) : j \in \mathcal{W}\}$. For each set $\mathcal{J} \subseteq [\mathsf{K}]$ where $|\mathcal{J}| = g+1$, each worker $j \in \mathcal{J}$ broadcasts

$$W^{t+1}_{j,\mathcal{J}} = \underset{k\in\mathcal{J}\setminus\{j\}}{\oplus} \mathcal{A}^{t+1}_{k,\mathcal{J}\setminus\{k\}}(j). \tag{53}$$

It can be seen that each subfile in $W^{t+1}_{j,\mathcal{J}}$ is cached in the memory of worker j at the end of time slot t. In addition, each worker $k \in \mathcal{J}\setminus\{j\}$ requests $\mathcal{A}^{t+1}_{j,\mathcal{J}\setminus\{k\}}(k)$ and knows $\mathcal{A}^{t+1}_{j,\mathcal{J}\setminus\{k_1\}}(k_1)$ where $k_1 \in \mathcal{J}\setminus\{k,j\}$ such that it can recover $\mathcal{A}^{t+1}_{j,\mathcal{J}\setminus\{k\}}(k)$. Hence, the worst-case load is $\mathsf{N}(1-g/\mathsf{K})/g = \mathsf{q}(\mathsf{K}-g)/g$ as claimed in Theorem 2.

Appendix D
Proof of Theorem 3

We focus on $\mathsf{M} = \mathsf{q}m$, where $m \in \{1, \mathsf{K}-2, \mathsf{K}-1\}$.

Storage Update Phase in Time Slot t: The storage update phase is the same as the improved shared-link data shuffling scheme in [8]. For each worker k and each file F_i where $i \in \mathcal{A}^t_k$, we divide F_i into $\binom{\mathsf{K}-1}{m-1}$ non-overlapping and equal-length subfile, each of which contains $\mathsf{B}/\binom{\mathsf{K}-1}{m-1}$ bits. Each subfile is denoted by $F_{i,\mathcal{W}}$ cached by workers in $\mathcal{W}\cup\{k\}$ for each $\mathcal{W} \subseteq ([\mathsf{K}]\setminus\{k\})$ where $|\mathcal{W}| = m-1$. Notice that for other sets $\mathcal{W}$, we let $F_{i,\mathcal{W}}$ be the empty set. It can be seen that worker k caches the whole file of F_i where $i \in \mathcal{A}^t_k$ and $\binom{\mathsf{K}-2}{m-2}\mathsf{B}/\binom{\mathsf{K}-1}{m-1}$ bits of each file F_j where $j \notin \mathcal{A}^t_k$. Hence, the total number of cached bits of worker k is

$$\mathsf{q} + (\mathsf{N}-\mathsf{q})\frac{\binom{\mathsf{K}-2}{m-2}}{\binom{\mathsf{K}-1}{m-1}} = \mathsf{q} + (m-1)(\mathsf{N}-\mathsf{q})/(\mathsf{K}-1) = m\mathsf{q}.$$

In addition, it can also be seen that each file F_i where $i \in \mathcal{A}^t_k$ is cached by worker k. So the cache constraints are satisfied.[2]

A. $\mathsf{M} = \mathsf{q}m$ *where* $m = 1$

Data Shuffling Phase in Time Slot $t+1$: Each worker $k \in [\mathsf{K}]$ broadcasts each file F_i where $i \in \mathcal{A}^{t+1}_k \setminus \mathcal{A}^t_k$. So the achieved worst-case load is N.

B. $\mathsf{M} = \mathsf{q}m$ *where* $m = \mathsf{K}-2$

Data Shuffling Phase in Time Slot $t+1$: We divide all the N files into q non-overlapping and equal-length groups, $[\mathsf{N}] = \{\mathcal{H}_i : i \in [\mathsf{q}]\}$, where each group contains K files. We impose that each file in one group is requested by a different worker in time slot $t+1$. In other words, for each group $\mathcal{H}_i$ and each worker $k \in [\mathsf{K}]$, we have $|\mathcal{H}_i \cap \mathcal{A}^{t+1}_k| = 1$.

We focus on one group $\mathcal{H}_i$. We denote the set of workers $k \in [\mathsf{K}]$ where $\big(\mathcal{H}_i \cap \mathcal{A}^{t+1}_k\big) \subseteq \mathcal{A}^t_k$ by $\mathcal{U}(\mathcal{H}_i)$. For each set $\mathcal{J} \subseteq [\mathsf{K}]$ where $|\mathcal{J}| = m+1 = \mathsf{K}-1$, we generate the following multicast message

$$V^{t+1}_{\mathcal{J}}(\mathcal{H}_i) = \underset{k\in\mathcal{J}}{\oplus} F_{\mathcal{H}_i\cap\mathcal{A}^{t+1}_k,\mathcal{J}\setminus\{k,\mathsf{u}^t_{\mathcal{H}_i\cap\mathcal{A}^{t+1}_k}\}}. \tag{54}$$

Since $\mathcal{H}_i \cap \mathcal{A}^{t+1}_k$ only contains one element, in (54) with an abuse of notation we let $\mathcal{H}_i \cap \mathcal{A}^{t+1}_k$ be this element. Obviously, each worker $k \in \mathcal{J}$ knows all the subfiles $F_{\mathcal{H}_i\cap\mathcal{A}^{t+1}(k_1),\mathcal{J}\setminus\{k_1,\mathsf{u}^t_{\mathcal{H}_i\cap\mathcal{A}^{t+1}(k_1)}\}}$, where $k_1 \in \mathcal{J}\setminus\{k\}$. For each worker $k \in \mathcal{J}$, we can see that if $\mathsf{u}^t_{\mathcal{H}_i\cap\mathcal{A}^{t+1}_k} \notin \mathcal{J}$ or $k \in \mathcal{U}(\mathcal{H}_i)$, the subfile $F_{\mathcal{H}_i\cap\mathcal{A}^{t+1}_k,\mathcal{J}\setminus\{k,\mathsf{u}^t_{\mathcal{H}_i\cap\mathcal{A}^{t+1}_k}\}}$ is empty, and thus $V^{t+1}_{\mathcal{J}}(\mathcal{H}_i)$ is already known by user k before the delivery phase.

We divide our consideration into three cases, $|\mathcal{U}(\mathcal{H}_i)| = 0$, $|\mathcal{U}(\mathcal{H}_i)| = 1$ and $|\mathcal{U}(\mathcal{H}_i)| > 1$.

[2] This storage update phase for each worker $k \in [\mathsf{K}]$ could be done with Z^{t-1}_k and X^t_j where $j \in [\mathsf{K}]\setminus\{k\}$. More precisely, For each file F_i where $i \in \mathcal{A}^t_k \setminus \mathcal{A}^{t-1}_k$, worker k stores the whole file F_i in the cache. For each file $\mathcal{A}^{t-1}_k \setminus \mathcal{A}^t_k$, instead of storing the whole file F_i, worker k only stores the bits of F_i which was cached by worker u^t_i at the end of time slot $t-1$. For other files, worker k does not change the cached bits.

- $|\mathcal{U}(\mathcal{H}_i)| = 0$. For each set $\mathcal{J} \subseteq [\mathsf{K}]$ where $|\mathcal{J}| = m+1 = \mathsf{K}-1$, since $|\mathcal{J}| = \mathsf{K}-1$, among all the workers in $\mathcal{J}$, there is exactly one worker in $\mathcal{J}$ (assumed to be k) where $\mathsf{u}^t_{\mathcal{H}_i \cap \mathcal{A}^{t+1}_k} \notin \mathcal{J}$. We then let user k transmit $V^{t+1}_{\mathcal{J}}(\mathcal{H}_i)$. Hence, the load to transmit the files in $\mathcal{H}_i$ denoted by $\mathsf{R}^{t+1}_{\mathcal{H}_i}$ is equal to $\mathsf{K}/\binom{\mathsf{K}-1}{m-1}$.
- $|\mathcal{U}(\mathcal{H}_i)| = 1$. We assume the user in $\mathcal{U}(\mathcal{H}_i)$ is j_1. So j_1 does not need to recover any file in $\mathcal{H}_i$. For the set $\mathcal{J} \subseteq [\mathsf{K}]$ where $|\mathcal{J}| = m+1 = \mathsf{K}-1$ and $j_1 \notin \mathcal{J}$, $V^{t+1}_{\mathcal{J}}(\mathcal{H}_i)$ contains exactly $|\mathcal{J}|$ subfiles. we divide each subfile $F_{i,\mathcal{W}}$ in (54) into $|\mathcal{J}|-1$ non-overlapping and equal-length pieces, $F_{i,\mathcal{W}} = \{F_{i,\mathcal{W},j} : j \in \mathcal{J} \setminus \{k_2\}\}$, where $i \in \mathcal{A}^{t+1}_{k_2}$. We let each worker $k_3 \in \mathcal{J}$ broadcast

$$V^{t+1}_{k_3,\mathcal{J}}(\mathcal{H}_i) = \underset{k \in \mathcal{J} \setminus \{k_3\}}{\oplus} F_{\mathcal{H}_i \cap \mathcal{A}^{t+1}_k, \mathcal{J} \setminus \{k, \mathsf{u}^t_{\mathcal{H}_i \cap \mathcal{A}^{t+1}_k}\}, k_3}. \tag{55}$$

We then focus on one user $j_2 \neq j_1$. For each set $\mathcal{J} \subseteq [\mathsf{K}]$ where $|\mathcal{J}| = m+1 = \mathsf{K}-1$ and $k \in \mathcal{J}$, if $\mathsf{u}^t_{\mathcal{H}_i \cap \mathcal{A}^{t+1}_{j_2}} \notin \mathcal{J}$, j_2 knows $V^{t+1}_{\mathcal{J}}(\mathcal{H}_i)$ before the delivery phase. Hence, among all the $\mathsf{K}-1$ messages $V^{t+1}_{\mathcal{J}}(\mathcal{H}_i)$ where $\mathcal{J} \subseteq [\mathsf{K}]$, $|\mathcal{J}| = m+1 = \mathsf{K}-1$ and $k \in \mathcal{J}$, each user in $[\mathsf{K}] \setminus \{j_1\}$ knows one message In addition, those $\mathsf{K}-1$ messages are known by user j_1. So we let user j_1 transmit $(\mathsf{K}-2)\mathsf{B}/\binom{\mathsf{K}-1}{m-1}$ random linear combinations of all bits in those $\mathsf{K}-1$ messages. Totally, in this case, $\mathsf{R}^{t+1}_{\mathcal{H}_i} = \frac{(m+1)/m+\mathsf{K}-2}{\binom{\mathsf{K}-1}{m-1}}$.
- $|\mathcal{U}(\mathcal{H}_i)| > 1$. For each set $\mathcal{J} \subseteq [\mathsf{K}]$ where $|\mathcal{J}| = m+1 = \mathsf{K}-1$ and $([\mathsf{K}] \setminus \mathcal{U}(\mathcal{H}_i)) \subseteq \mathcal{J}$, there must exist at least one user in $\mathcal{J} \cap \mathcal{U}(\mathcal{H}_i)$ who knows $V^{t+1}_{k_3,\mathcal{J}}(\mathcal{H}_i)$. Hence, we let this user to transmit $V^{t+1}_{\mathcal{J}}(\mathcal{H}_i)$. The number of such sets $\mathcal{J}$ is $|\mathcal{U}(\mathcal{H}_i)|$. We then focus on one user $j_2 \notin \mathcal{U}(\mathcal{H}_i)$. Among all of the remaining $\mathsf{K} - |\mathcal{U}(\mathcal{H}_i)|$ sets $\mathcal{J}$, there exists exactly one $\mathcal{J}$ such that $V^{t+1}_{\mathcal{J}}(\mathcal{H}_i)$ is known by j_2. In addition, each user in $\mathcal{U}(\mathcal{H}_i)$ knows $V^{t+1}_{\mathcal{J}}(\mathcal{H}_i)$ for all of the remaining $\mathsf{K} - |\mathcal{U}(\mathcal{H}_i)|$ sets $\mathcal{J}$. Hence, we select one user in $\mathcal{U}(\mathcal{H}_i)$ to transmit $(\mathsf{K} - |\mathcal{U}(\mathcal{H}_i)| - 1)\mathsf{B}/\binom{\mathsf{K}-1}{m-1}$ random linear combinations of all bits in those $\mathsf{K} - |\mathcal{U}(\mathcal{H}_i)|$ messages. Totally, in this case, $\mathsf{R}^{t+1}_{\mathcal{H}_i} = \frac{\mathsf{K}-1}{\binom{\mathsf{K}-1}{m-1}}$.

In conclusion, considering all the groups, the achieved worst-case load is $\mathsf{q}\mathsf{K}/\binom{\mathsf{K}-1}{m-1} = \frac{\mathsf{K}-m}{m}\frac{\mathsf{K}}{\mathsf{K}-1}\mathsf{q}$.

C. $\mathsf{M} = \mathsf{q}m$ *where* $m = \mathsf{K}-1$

Data Shuffling Phase in Time Slot $t+1$*:* For each worker $k \in [\mathsf{K}]$, we let $\mathcal{D}^{t+1}(k) := \mathcal{A}^{t+1}_k \setminus \mathcal{A}^t_k$. Since $m = \mathsf{K}-1$ and each file $i \in \mathcal{D}^{t+1}(k)$ is known by worker u^t_i, we can see that $\mathcal{D}^{t+1}(k)$ is known by all the workers $[\mathsf{K}] \setminus \{k\}$. Hence, we divide $\mathcal{D}^{t+1}(k)$ into $\mathsf{K}-1$ non-overlapping and equal-length pieces, $\mathcal{D}^{t+1}(k) = \{\mathcal{D}^{t+1}_{k_1}(k) : k_1 \in ([\mathsf{K}] \setminus \{k\})\}$.

We then let each worker $k \in [\mathsf{K}]$ broadcasts

$$T^{t+1}_{k,\mathcal{J}} := \underset{j \in ([\mathsf{K}] \setminus \{k\})}{\oplus} \mathcal{D}^{t+1}_k(j). \tag{56}$$

Hence, the worst-case load is $\mathsf{N}\frac{1-(m-1)/(\mathsf{K}-1)}{m} = \frac{\mathsf{K}-m}{m}\frac{\mathsf{K}}{\mathsf{K}-1}\mathsf{q}$.

APPENDIX E
PROOF OF THEOREM 4

Let $\mathsf{R}_{\text{Thm.(2)}}(\mathsf{M})$ be the load achieved by the scheme in Theorem 2. From Theorem 2, for each $g \in [\mathsf{K}-1]$, we have

$$\mathsf{M}_1 = \big(1 + g\frac{\mathsf{K}-1}{\mathsf{K}}\big)\mathsf{q},\ \mathsf{R}_{\text{Thm.(2)}}(\mathsf{M}_1) = \mathsf{q}\frac{\mathsf{K}-g}{g};$$
$$\mathsf{M}_2 = \big(1 + (g+1)\frac{\mathsf{K}-1}{\mathsf{K}}\big)\mathsf{q},\ \mathsf{R}_{\text{Thm.(2)}}(\mathsf{M}_2) = \mathsf{q}\frac{\mathsf{K}-g-1}{g+1}.$$

Hence, by memory-sharing between above two point we get the load for $\mathsf{M}_3 = (1+g)\mathsf{q}$ as

$$\mathsf{R}_{\text{Thm.(2)}}(\mathsf{M}_3) = \frac{\mathsf{N}\big(\mathsf{K}^2(g+1) + g(g+1) - \mathsf{K}(g^2+3g+1)\big)}{(\mathsf{K}-1)\mathsf{K}g(g+1)}. \tag{57}$$

From Theorem 1, we know that

$$\mathsf{R}^{\star}_{\mathrm{u}}(\mathsf{M}_3) \geq \mathsf{N}\frac{1-g/(\mathsf{K}-1)}{g+1}. \tag{58}$$

Hence, from (57) and (58), we have

$$\begin{aligned}\frac{\mathsf{R}_{\text{Thm.(2)}}(\mathsf{M}_3)}{\mathsf{R}^{\star}_{\mathrm{u}}(\mathsf{M}_3)} &\leq \frac{\big(\mathsf{K}^2(g+1) + g(g+1) - \mathsf{K}(g^2+3g+1)\big)}{(\mathsf{K}-1)\mathsf{K}g\big(1-g/(\mathsf{K}-1)\big)} \\ &= 1 + \frac{\mathsf{K}^2 + g^2 + g - 2\mathsf{K}g - \mathsf{K}}{\mathsf{K}g(\mathsf{K}-g-1)} \\ &= 1 - \frac{1}{\mathsf{K}} + \frac{1}{g} < 2. \end{aligned} \tag{59}$$

When $\mathsf{M} = \mathsf{q}$, the achievable scheme in Appendix D-A achieves the converse bound $\mathsf{R}^{\star}_{\mathrm{u}} = \mathsf{N}$. Hence, from (59), $\mathsf{R}^{\star}_{\mathrm{u}} = \mathsf{N}$, and Theorem 1, we can see our proposed schemes are order optimal within a constant factor of f, where $\mathsf{f} < 2$.

REFERENCES

[1] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," *IEEE Trans. Inf. Theory*, vol. 64, no. 3, Mar. 2018.

[2] J. Chung, K. Lee, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Ubershuffle: Communication-efficient data shuffling for sgd via coding theory," in *NIPS 2017, ML Systems Workshop*.

[3] M. A. Attia and R. Tandon, "Information theoretic limits of data shuffling for distributed learning," *in IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2016.

[4] ——, "On the worst-case communication overhead for distributed data shuffling," *in 54th Annual Allerton Conf. on Commun., Control, and Computing (Allerton)*, Sep. 2016.

[5] M. A. Maddah-Ali and U. Niesen, "Fundamental limits of caching," *IEEE Trans. Infor. Theory*, vol. 60, no. 5, pp. 2856–2867, May 2014.

[6] K. Wan, D. Tuninetti, and P. Piantanida, "On the optimality of uncoded cache placement," *in IEEE Infor. Theory Workshop*, Sep. 2016.

[7] Q. Yu, M. A. Maddah-Ali, and S. Avestimehr, "The exact rate-memory tradeoff for caching with uncoded prefetching," *in IEEE Int. Symp. Inf. Theory*, Jun. 2017.

[8] M. A. Attia and R. Tandon, "Near optimal coded data shuffling for distributed learning," *arXiv:1801.01875*, Jan. 2018.

[9] A. Elmahdy and S. Mohajer, "On the fundamental limits of coded data shuffling," *in IEEE Int. Symp. Inf. Theory*, Jun. 2018.

[10] M. Ji, G. Caire, and A. Molisch, "Fundamental limits of caching in wireless d2d networks," *IEEE Trans. Inf. Theory*, vol. 62, no. 1, pp. 849–869, 2016.

[11] S. Li, M. A. Maddah-Ali, Q. Yu, and A. S. Avestimehr, "A fundamental tradeoff between computation and communication in distributed computing," *IEEE Trans. Inf. Theory*, vol. PP (99), Sep. 2017.