Coded Distributed Computing with Node Cooperation Substantially Increases Speedup Factors

Emanuele Parrinello, Eleftherios Lampiris & Petros Elia
Communcations Department, EURECOM
Sophia Antipolis, France
Email: {parrinel,lampiris,elia}@eurecom.fr

***Abstract*— This work explores a distributed computing setting where K nodes are assigned fractions (subtasks) of a computational task in order to perform the computation in parallel. In this setting, a well-known main bottleneck has been the internode communication cost required to parallelize the task, because unlike the computational cost which could keep decreasing as K increases, the communication cost remains approximately constant, thus bounding the total speedup gains associated to having more computing nodes. This bottleneck was substantially ameliorated by the recent introduction of coded techniques in the context of MapReduce which allowed each node — at the computational cost of having to preprocess approximately t times more subtasks — to reduce its communication cost by approximately t times. In reality though, the associated speed up gains were severely limited by the requirement that larger t and K necessited that the original task be divided into an extremely large number of subtasks. In this work we show how node cooperation, along with a novel assignment of tasks, can help to dramatically ameliorate this limitation. The result applies to wired as well as wireless distributed computing and it is based on the idea of having groups of nodes compute identical mapping tasks and then employing a here-proposed novel D2D coded caching algorithm. In this context, the new approach here manages to achieve a virtual decomposition of the fully connected D2D setting into parallel ones, which significantly reduces the required subpacketization.**

I. Introduction

Parallel computing exploits the presence of more than one available computing node, in order to allow for faster execution of a computational task. This effort usually involves dividing the original computational task into different subtasks and then assigning these subtasks to different nodes which, after some intermediate steps, compute the final task in parallel.

While some rare tasks are by nature already parallel, most computational problems need to be parallelized, and this usually involves an intermediate preprocessing step and a subsequent information exchange between the nodes. One such special class of distributed computing algorithms follows the MapReduce model [1], which is a parallel processing tool that simplifies the parallel execution of tasks, by abstracting the original problem into the following three phases:

1) the *mapping phase*, where each element of the dataset is assigned to one or more computing nodes and where the nodes perform an intermediate computation aiming to "prepare" for parallelization,
2) the *shuffling phase* (or communication phase), where nodes communicate between each other the preprocessed data that is needed to make the process parallel, and
3) the *reduce phase*, where nodes work in parallel to provide the final output that each is responsible for.

Classes of tasks that can be parallelized under a MapReduce framework include Sorting [2], Data Analysis and Clustering [3], [4] and Word Counting [5], among others.

A. Communication bottleneck of distributed computing

While though MapReduce allows for parallelization, it also comes with different bottlenecks involving for example struggling nodes [6], [7] and non-fine-tuned algorithms [8]. The main bottleneck, though, which bounds the performance of MapReduce is the duration of the shuffling phase, especially as the dataset size becomes larger and larger. While having more nodes can speed up computational time, the aforementioned information exchange often yields unchanged or even increased communication load and delays, leading to a serious bottleneck in the performance of distributed computing algorithms.

Phase delays: In particular, consider a setting where there are K computing nodes, operating on a dataset of F elements. Assuming that each element of the dataset can appear in t different computing nodes, and assuming that $T_{\text{map}}(F)$ represents the time required for one node to map the entire dataset, then the map phase will have duration approximately $T_{\text{map}}(t\frac{F}{K})$ which generally reduces with K. Similarly the final reduce phase enjoys the same decreased delay $T_{\text{red}}(F/K)$, where $T_{\text{red}}(F)$ denotes the time required for a single node to reduce the entire mapped dataset[1].

The problem lies with the communication delay $T_{\text{com}}(F)$. For T_c denoting the time required to transmit the entire mapped dataset, from one node to another without any interference from the other nodes[2], and accounting for a reduction by the factor $(1-\gamma)$ due to the fact that each node already has a fraction $\gamma = t/K$ of the dataset, then the delay of the

This work was supported by the European Research Council under the EU Horizon 2020 research and innovation program / ERC grant agreement no. 725929.

[1]We here assume for simplicity of exposition, uniformity in the amount of mapped data that each node uses in the final reduce phase. We also assume a uniformity in the computational capabilities of each node.

[2]T_c accounts for the ratio between the capacity of the communication link and the dataset size F.

shuffling phase takes the form $T_{\text{com}}(F) = T_c \cdot (1-\gamma)$, which does not decrease with K.

Hence for the basic MapReduce (MR) algorithm — under the traditional assumption that the three phases are performed sequentially — the overall execution time becomes

$$T_{\text{tot}}^{\text{MR}}(F,K) = T_{\text{map}}(\frac{t}{K}F) + T_c \cdot (1-\gamma) + T_{\text{red}}(\frac{F}{K})$$

which again shows that, while the joint computational cost $T_{\text{map}}(\frac{t}{K}F) + T_{\text{red}}(\frac{F}{K})$ of the map and reduce phases can decrease by adding more nodes, the communication time $T_c \cdot (1-\gamma)$ is not reduced and thus the cost of the shuffling phase emerges as the actual bottleneck of the entire process.

B. Emergence of Coded MapReduce: exploiting redundancy

For a general distributed computing problem fitting the aforementioned MapReduce model, a method of reducing the communication load was introduced in [9] (see also [10], [11]), which modified the mapping phase, in order to allow for the shuffling phase to employ coded communication. The main idea of the method — which is referred to as Coded MapReduce (CMR) — was to assign and then force each node to map a fraction $\gamma > 1/K$ of the whole dataset (such that each element of the dataset is mapped in $t = K\gamma$ computing nodes) and then — based on the fact that such a mapping would allow for common mapped information at the different nodes — to eventually perform coded communication, where during the shuffling phase, the packets were not sent one after the other, but were rather combined together into XORs and sent as one. The reason this speedup would work is because the recipients of these packets could use part of their (redundant) mapped packets in order to remove the interfering packets from the received XOR, and acquire their own requested packet. This allowed for communicating (during the shuffling phase) to $t = K\gamma$ nodes at a time, thus reducing the shuffling phase duration, from $T_c \cdot (1-\gamma)$ to $\frac{1}{t}T_c \cdot (1-\gamma) = \frac{1}{K\gamma}T_c \cdot (1-\gamma)$.

C. Subpacketization bottleneck of distributed computing

Despite the fact that the aforementioned coded method promises, in theory, big delay reductions by a factor of $t = K\gamma$ compared to conventional uncoded schemes, these gains are heavily compromised by the fact that the method requires that the dataset be split into an unduly large number of $S = \binom{K}{t}$ packets[3], where $\binom{K}{t}$ denotes the binomial coefficient.

Specifically, S grows exponentially in K and t while the finite-sized dataset can only be divided into a limited number of packets, which limits the values of speedup parameter t that can be achieved, because the corresponding subpacketization S must be kept below some maximum allowable subpacketization $S_{\max}$, which, also, must be less than the number of elements F in the dataset. If this number $S = \binom{K}{t}$ exceeds the maximum allowable subpacketization $S_{\max}$, then coded communication is limited to include coding that spans only

$$\bar{K} = \arg\max_{K} \left\{ \binom{K}{t} \leq S_{\max} \right\} \tag{1}$$

nodes at a time, forcing us to repeat the coded communication $K/\bar{K}$ times, thus resulting in a smaller, actual gain

$$\bar{t} = \bar{K}\gamma < K\gamma$$

which can be far smaller than the theoretical communication gain due to coding. Such high subpacketization can naturally limit the coding gains t, but it can also further delay the shuffling phase because it implies more transmissions and thus higher packet overheads, as well as because smaller packets are more prone to having mapped outputs that are unevely sized, thus requiring more zero padding.

In what follows, we will ameliorate the above problems with a novel group-based method of distributing the dataset across the computing nodes and a novel method of cooperation/coordination between nodes in the transmission, which will jointly yield a much reduced subpacketization, allowing for a wider range of t values to be feasible, thus eventually allowing substantial reductions in the overall execution time for a large class of distributed computing algorithms.

Before describing our solution and its performance, let us first elaborate on the exact channel model.

D. Channel model: Distributed computing in a D2D setting

In terms of the communication medium, we will focus on the wireless fully-connected setting, because in the wireless setting the nature of multicasting and the impact of link bottlenecks are clearer. As we will discuss later on though, the ideas here apply directly to the wired case as well.

We assume that the K computing nodes are all fully connected via a wireless shared channel as in the classical fully-connected D2D wireless network. At each point there will be a set of active receivers, and active transmitters. Assuming a set of L active transmitters jointly transmitting vector $\mathbf{x} \in \mathbb{C}^{L\times 1}$, then the received signal at a receiving node k takes the form

$$y_k = \mathbf{h}_k^T \mathbf{x} + w_k, \quad k \in \{1, \cdots, K\} \triangleq [K], \tag{2}$$

where as always $\mathbf{x}$ satisfies a power constraint $\mathbb{E}(||\mathbf{x}||^2) < P$, where $\mathbf{h}_k \in \mathbb{C}^{L\times 1}$ is the (potentially random) fading channel between the transmitting set of nodes and the receiving node k, and where w_k denotes the unit-power AWGN noise at receiver k. We assume the system to operate in the high SNR regime (high P), and we assume perfect channel state information (CSI) (and for the wired case, perfect network coding coefficients) at the active receivers and transmitters.

E. Notation

If $\mathcal{A}$ is set, then $|\mathcal{A}|$ will denote its cardinality, and $\mathcal{A}(j)$ will denote its jth element if $\mathcal{A}$ is totally ordered. For sets $\mathcal{A}$ and $\mathcal{B}$, $\mathcal{A}\backslash\mathcal{B}$ denotes the difference set.

[3]The subpacketization S strictly refers to the number of chunks the dataset has to be split into in order to prepare the mapping phase. At the time when the shuffling phase takes place, CMR requires to further split each mapped chunk into t equally-sized smaller parts. This extra subpacketization, which occurs at a bit level, is negligible compared to S and is not taken into account in our analysis.

II. Main result

We proceed to describe the performance of the new proposed algorithm, which will be presented in the next section. Key to this algorithm — which we will refer to as the Group-based Coded MapReduce (GCMR) algorithm — is the concept of user grouping. We will group the K nodes into K/L groups of L nodes each, and then every node in a group will be assigned the same subset of the dataset and will produce the same mapped output. By properly doing so, this will allow us to use in the shuffling phase a new — developed in this work — D2D coded caching communication algorithm which assigns the D2D nodes an adaptive amount of content overlap[4]. This, in turn, will dramatically reduce the required subpacketization, thus substantially boosting the speedup in communication and overall execution time.

For the sake of comparison, let us first recall that under the subpacketization constraint $S_{\max}$, the original Coded MapReduce approach achieves communication delay

$$T_{com}^{\text{CMR}} = \frac{1-\gamma}{\bar{t}} T_c \tag{3}$$

where

$$\bar{t} = \gamma \cdot \arg\max_{K} \{ \binom{K}{K\gamma} \leq S_{\max} \}$$

is the maximum achievable effective speedup (due to coding) in the shuffling phase.

We proceed with the main result.

Theorem 1. *In the K-node distributed computing setting where the dataset can only be split into at most S_{max} identically sized packets, the proposed Group-based Coded MapReduce algorithm with groups of L users, can achieve communication delay*

$$T_{com}^{GCMR} = \frac{1-\gamma}{\bar{t}_L} T_c$$

for

$$\bar{t}_L = \gamma \cdot \arg\max_{K} \{ \binom{K/L}{K\gamma/L} \leq S_{\max} \}.$$

Proof. The proof follows directly from the description of the scheme in Section III. □

The above implies the following corollary, which reveals that in the presence of subpacketization constraints, simple node grouping can further speedup the shuffling phase by a factor of up to L.

Corollary 1. *In the subpacketization-constrained regime where $S_{\max} \leq \binom{K/L}{K\gamma/L}$, the new algorithm here allows for shuffling delay*

$$T_{com}^{GCMR} = \frac{1-\gamma}{\bar{t}_L} T_c = \frac{T_{com}^{CMR}}{L}$$

which is L times smaller than the delay without grouping for the same choice of the parameter γ.

[4]This general idea draws from the group-based cache-placement idea developed in [12] for the cache-aided MISO broadcast channel.

Proof. The proof is direct from the theorem. □

Finally the following also holds.

Corollary 2. *When $S_{max} \geq \binom{K/L}{K\gamma/L}$, the new algorithm allows for the unconstrained theoretical execution time*

$$T_{tot}^{GCMR} = T_{map}(\gamma F) + \frac{(1-\gamma)}{K\gamma} T_c + T_{red}\left(\frac{F}{K}\right). \tag{4}$$

Proof. The proof is direct from the theorem. □

III. Description of scheme

We proceed to describe the scheme. We consider a dataset Φ consisting of F elements and a computational task that asks for $Q \geq K$ output values $u_q = \phi_q(\Phi),\ q = 1, \cdots, Q$. The general aim is to distribute this task across the K nodes, hence the dataset is split into S disjoint packets $W_s,\ s = 1, \cdots, S$ ($\cup_{s=1}^{S} W_s = \Phi$). We recall that, as is common in MapReduce, each function ϕ_q is decomposable as

$$\phi_q(\Phi) = r_q(m_q(W_1), \cdots, m_q(W_S)) \tag{5}$$

where the *map functions* $\{m_q,\ q \in [Q]\}$ map packet W_s into Q intermediate values $W_s^q = m_q(W_s),\ q \in [Q]$, which are used by the reduce function r_q to calculate the desired output value $u_q = r_q(W_1^q, \cdots, W_S^q)$.

We proceed to describe the *Assignment-and-Map, Shuffle* and *Reduce* phases.

A. Dataset assignment phase

We split the K nodes into $K' \triangleq \frac{K}{L}$ groups, as follows

$$\mathcal{G}_i = \{i, i+K', ..., i+(L-1)K'\}, \quad i \in [K'] \tag{6}$$

with L nodes per group, and we split the dataset into

$$S = \binom{K'}{K'\gamma} \tag{7}$$

packets, where $\gamma \in \{\frac{1}{K'}, \frac{2}{K'}, \cdots, 1\}$ is a parameter of choice defining the redundancy factor of the mapping phase later on. At this point, each $s \in [S]$ is associated to a unique subset $\mathcal{T}$ of $[K']$ of cardinality $|\mathcal{T}| = K'\gamma$ so that the dataset can be seen as being segmented into $\{W_{\mathcal{T}},\ \mathcal{T} \subset [K'] : |\mathcal{T}| = K'\gamma\}$. Each node in group $\mathcal{G}_i$ is then assigned the set of packets

$$\mathcal{M}_{\mathcal{G}_i} = \{W_{\mathcal{T}} : \mathcal{T} \ni i\} \tag{8}$$

and each of the Q reduce functions r_q is assigned to a given node. For simplicity we assume that $Q = K$.

B. Map Phase

This phase consists of each node k computing the map functions m_q of all packets in $\mathcal{M}_{\mathcal{G}_i}, \mathcal{G}_i \ni k$ for all $q \in [Q]$. At the end of the phase, node $k \in \mathcal{G}_i$ has computed the intermediate results $W_{\mathcal{T}}^q = m^q(W_{\mathcal{T}})$ for all $W_{\mathcal{T}} \in \mathcal{M}_{\mathcal{G}_i}$.

C. Shuffle Phase

Each node $\mathcal{G}_i(j)$ of group $\mathcal{G}_i$, must retrieve from the other nodes (except from those in $\mathcal{G}_i$), the intermediate values $\{W_{\mathcal{T}}^{\mathcal{G}_i(j)} : W_{\mathcal{T}} \notin \mathcal{M}_{\mathcal{G}_i}\}$ that it has not computed locally. Each node $\mathcal{G}_i(j)$ will thus create a set of symbols $\{x_{\mathcal{G}_i(j),\mathcal{Q}\setminus\{i\}}\}$, intended for all the nodes in groups $\mathcal{G}_k, k \in \mathcal{Q}\setminus\{i\}$ for some $\mathcal{Q} \subseteq [K']$ of size $|\mathcal{Q}| = K'\gamma+1$, where of course each symbol $x_{\mathcal{G}_i(j),\mathcal{Q}\setminus\{i\}}$ is a function of the intermediate values computed in the map phase. We use

$$\mathbf{x}_{i,\mathcal{Q}\setminus\{i\}} \triangleq [x_{\mathcal{G}_i(1),\mathcal{Q}\setminus\{i\}}, \cdots, x_{\mathcal{G}_i(L),\mathcal{Q}\setminus\{i\}}]^T$$

to denote the vector of symbols that are jointly created by the users in $\mathcal{G}_i$ and which are intended for the users in $\mathcal{G}_k, k \in \mathcal{Q}\setminus\{i\}$. Each symbol is communicated (multicasted) by the corresponding node $\mathcal{G}_i(j)$, to all the other nodes. We proceed to provide the details for transmission and decoding.

a) Transmission: For each subset $\mathcal{Q} \subseteq [K']$ of size $|\mathcal{Q}| = K'\gamma+1$, we sequentially pick element $i \in \mathcal{Q}$ so that the users in group $\mathcal{G}_i$ act as a single distributed transmitter. For each $\mathcal{G}_{k'}(j) \in \mathcal{G}_{k'}, k' \in \mathcal{Q}\setminus\{i\}$, users in group $\mathcal{G}_i$ partition $W_{\mathcal{Q}\setminus\{k'\}}^{\mathcal{G}_{k'}(j)}$ into $K'\gamma$ equally-sized disjoint segments:

$$W_{\mathcal{Q}\setminus\{k'\}}^{\mathcal{G}_{k'}(j)} = \{W_{\mathcal{Q}\setminus\{k'\},p}^{\mathcal{G}_{k'}(j)} : p \in \mathcal{Q}\setminus\{k'\}\}$$

where $W_{\mathcal{Q}\setminus\{k'\},i}^{\mathcal{G}_{k'}(j)}$ is the segment associated to group $\mathcal{G}_i$. At this point, these users in $\mathcal{G}_i$ construct the following vector of symbols

$$\mathbf{x}_{i,\mathcal{Q}\setminus\{i\}} = \sum_{k'\in\mathcal{Q}\setminus\{i\}} \mathbf{H}_{i,k'}^{-1} \left[W_{\mathcal{Q}\setminus\{k'\},i}^{\mathcal{G}_{k'}(1)}, \cdots, W_{\mathcal{Q}\setminus\{k'\},i}^{\mathcal{G}_{k'}(L)}\right]^T \quad (9)$$

where $\mathbf{H}_{i,k'}^{-1}$ is the ZF precoding matrix for the channel $\mathbf{H}_{i,k'} \in \mathbb{C}^{L\times L}$ between transmitting group $\mathcal{G}_i$ and receiving group $\mathcal{G}_{k'}$, and where $\{W_{\mathcal{Q}\setminus\{k'\},i}^{\mathcal{G}_{k'}(j)}\}_{j=1}^{L}$ is a set of intermediate values desired by the nodes in $\mathcal{G}_{k'}$. Each user $\mathcal{G}_i(j)$ now transmits the j-th element of the constructed vector $\mathbf{x}_{i,\mathcal{Q}\setminus\{i\}}$.

b) Decoding: Node $\mathcal{G}_p(j), p \in \mathcal{Q}\setminus\{i\}$ receives the signal

$$y_{\mathcal{G}_p(j)} = \mathbf{h}_{\mathcal{G}_p(j)}^T \mathbf{x}_{i,\mathcal{Q}\setminus\{i\}} + w_{\mathcal{G}_p(j)} \quad (10)$$

and removes out-of-group interference by employing the intermediate values it has computed locally in the map phase. Specifically each node $\mathcal{G}_p(j)$, and all the nodes in $\mathcal{G}_p, p \in \mathcal{Q}$, remove from their $y_{\mathcal{G}_p(j)}$ the signal

$$\mathbf{h}_{\mathcal{G}_p(j)}^T \sum_{k'\in\mathcal{Q}\setminus\{i,p\}} \mathbf{H}_{i,k'}^{-1} \left[W_{\mathcal{Q}\setminus\{k'\},i}^{\mathcal{G}_{k'}(1)}, \cdots, W_{\mathcal{Q}\setminus\{k'\},i}^{\mathcal{G}_{k'}(L)}\right]^T \quad (11)$$

to stay with a residual signal

$$\mathbf{h}_{\mathcal{G}_p(j)}^T \mathbf{H}_{i,p}^{-1} \left[W_{\mathcal{Q}\setminus\{p\},i}^{\mathcal{G}_p(1)}, \cdots, W_{\mathcal{Q}\setminus\{p\},i}^{\mathcal{G}_p(L)}\right]^T + w_{\mathcal{G}_p(j)}. \quad (12)$$

The choice of $\mathbf{H}_{i,p}^{-1}$ to be a ZF precoder, removes intra-group interference, thus allowing each node $\mathcal{G}_p(j)$ to receive its desired intermediate value $W_{\mathcal{Q}\setminus\{p\},i}^{\mathcal{G}_p(j)}$. The shuffling phase is concluded by going over all the aforementioned sets $\mathcal{Q} \subset [K']$ of size $K'\gamma + 1$.

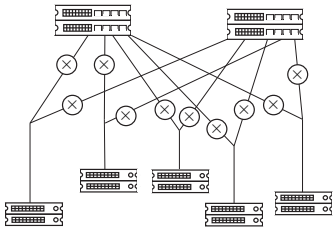

Fig. 1. Illustration of the wired setting. $\times$ denotes a network coding operation.

D. Reduce Phase

At this point, each node uses the symbols received during the shuffling phase, together with the intermediate values computed locally, in order to construct the inputs $W_1^q, ..., W_S^q$ that are required by the reduce function r_q to calculate the desired output value $u_q = r_q(W_1^q, \cdots, W_S^q)$.

E. Calculation of shuffling delay

We first see from (7) that the subpacketization is equal to

$$S = \binom{K'}{K'\gamma} = \binom{K/L}{K\gamma/L}. \quad (13)$$

Let us now verify that the shuffling delay is $T_{com}^{GCMR} = \frac{1-\gamma}{\bar{t}_L} T_c$. To do this, let us first assume that $S_{max} \geq S$ in which case we want to show that $T_{com}^{GCMR} = \frac{1-\gamma}{K\gamma} T_c$. To verify the first term ($K\gamma$), we just need to note that during the shuffling phase no segment is ever sent more than once, and then simply note that the scheme serves a total of $K'\gamma$ groups at a time, thus a total of $K'\gamma L = K\gamma$ nodes at a time. Finally to justify the term $1-\gamma$, we just need to recall that — due to the placement redundancy — a fraction γ of all the shuffled data is already at their intended destination.

Lastly when $S_{max} \geq S$, we simply have to recall that we are allowed — without violating the subpacketization constraint — to encode over $\bar{K}_L = \arg\max_K \{\binom{K/L}{K\gamma/L} \leq S_{max}\}$ nodes at a time, which yields the desired $\bar{t}_L = \gamma \cdot \bar{K}_L$. This concludes the proof of the results.

F. Extension to the Wired Setting

As a last step, we quickly note that the same vector precoding used to separate the users of the same group (cf. (9),(12)) can be directly applied in the wired setting where the intermediate nodes (routers, switches, etc.) in the links, can perform pseudo-random network coding operations on the received data (cf. [13]). This would then automatically yield a linear invertible relationship between the input vectors and the received signals, thus allowing for the design of the precoders that cancel intra-group interference.

G. Example of the scheme

Let us consider a setting with $K = 32$ computing nodes, a chosen redundancy of $K\gamma = 16$, and a cooperation parameter $L = 8$. The nodes are split into $K' = K/L = 4$ groups

$$\mathcal{G}_1 = \{1, 5, 9, ..., 29\}, \qquad \mathcal{G}_2 = \{2, 6, 10..., 30\},$$
$$\mathcal{G}_3 = \{3, 7, 11, ..., 31\}, \qquad \mathcal{G}_4 = \{4, 8, 12, ..., 32\}.$$

and the dataset is split into $\binom{4}{2}$=6 packets as $\{W_{12}, W_{13}, W_{14}, W_{23}, W_{24}, W_{34}\}$, which are distributed to the nodes as follows:

$$\mathcal{M}_{\mathcal{G}_1} = \{W_{12}, W_{13}, W_{14}\}, \quad \mathcal{M}_{\mathcal{G}_2} = \{W_{12}, W_{23}, W_{24}\},$$
$$\mathcal{M}_{\mathcal{G}_3} = \{W_{13}, W_{23}, W_{34}\}, \quad \mathcal{M}_{\mathcal{G}_4} = \{W_{14}, W_{24}, W_{34}\}.$$

In the map phase, packet $W_{\mathcal{T}}$ is mapped into $\{W_{\mathcal{T}}^q\}_{q=1}^K$ such that, for example, $W_{\mathcal{T}}^1$ is the output of the first mapping function after acting on $W_{\mathcal{T}}$. Finally, after the segmentation of each mapped packet into $K'\gamma = 2$ smaller chunks, the transmissions are[5]:

$$\begin{aligned}
\mathbf{x}_{1,23} &= \mathbf{H}_{1,2}^{-1}\mathbf{W}_{13,1}^{\mathcal{G}_2} + \mathbf{H}_{1,3}^{-1}\mathbf{W}_{12,1}^{\mathcal{G}_3} \\
\mathbf{x}_{1,24} &= \mathbf{H}_{1,2}^{-1}\mathbf{W}_{14,1}^{\mathcal{G}_2} + \mathbf{H}_{1,4}^{-1}\mathbf{W}_{12,1}^{\mathcal{G}_4} \\
\mathbf{x}_{1,34} &= \mathbf{H}_{1,3}^{-1}\mathbf{W}_{14,1}^{\mathcal{G}_3} + \mathbf{H}_{1,4}^{-1}\mathbf{W}_{13,1}^{\mathcal{G}_4} \\
\mathbf{x}_{2,13} &= \mathbf{H}_{2,1}^{-1}\mathbf{W}_{23,2}^{\mathcal{G}_1} + \mathbf{H}_{2,3}^{-1}\mathbf{W}_{12,2}^{\mathcal{G}_3} \\
\mathbf{x}_{2,14} &= \mathbf{H}_{2,1}^{-1}\mathbf{W}_{24,2}^{\mathcal{G}_1} + \mathbf{H}_{2,4}^{-1}\mathbf{W}_{12,2}^{\mathcal{G}_4} \\
\mathbf{x}_{2,34} &= \mathbf{H}_{2,3}^{-1}\mathbf{W}_{24,2}^{\mathcal{G}_3} + \mathbf{H}_{2,4}^{-1}\mathbf{W}_{23,2}^{\mathcal{G}_4} \\
\mathbf{x}_{3,12} &= \mathbf{H}_{3,1}^{-1}\mathbf{W}_{13,3}^{\mathcal{G}_1} + \mathbf{H}_{3,2}^{-1}\mathbf{W}_{13,3}^{\mathcal{G}_2} \\
\mathbf{x}_{3,14} &= \mathbf{H}_{3,1}^{-1}\mathbf{W}_{34,3}^{\mathcal{G}_1} + \mathbf{H}_{3,4}^{-1}\mathbf{W}_{13,3}^{\mathcal{G}_4} \\
\mathbf{x}_{3,24} &= \mathbf{H}_{3,2}^{-1}\mathbf{W}_{34,3}^{\mathcal{G}_2} + \mathbf{H}_{3,4}^{-1}\mathbf{W}_{23,3}^{\mathcal{G}_4} \\
\mathbf{x}_{4,12} &= \mathbf{H}_{4,1}^{-1}\mathbf{W}_{24,4}^{\mathcal{G}_1} + \mathbf{H}_{4,2}^{-1}\mathbf{W}_{14,4}^{\mathcal{G}_2} \\
\mathbf{x}_{4,13} &= \mathbf{H}_{4,1}^{-1}\mathbf{W}_{34,4}^{\mathcal{G}_1} + \mathbf{H}_{4,3}^{-1}\mathbf{W}_{14,4}^{\mathcal{G}_3} \\
\mathbf{x}_{4,23} &= \mathbf{H}_{4,2}^{-1}\mathbf{W}_{34,4}^{\mathcal{G}_2} + \mathbf{H}_{4,3}^{-1}\mathbf{W}_{24,4}^{\mathcal{G}_3},
\end{aligned}$$

where $\mathbf{W}_{\mathcal{T},i}^{\mathcal{G}_g}$ denotes a vector of $L = 8$ elements consisting of the intermediate results intended for nodes in group $\mathcal{G}_g$.

Observing for example the first transmission, we see that the nodes in group $\mathcal{G}_2$ can remove any interference caused by the intermediate results intended for group $\mathcal{G}_3$ since these intermediate values have been calculated by each node in $\mathcal{G}_2$ during the map phase. After noting that the precoding matrix $\mathbf{H}_{1,2}^{-1}$ removes intra-group interference, we can conclude that each transmission simultaneously serves each of the 16 users with one of their desired intermediate values, which in turn implies a 16-fold speedup over the uncoded case.

IV. Conclusion

The work provided a novel algorithm that employs node-grouping in the mapping and shuffling phases, to substantially reduce the shuffling-phase delays that had remained large due to the acute subpacketization bottleneck.

A. Minimal overhead for group-based node cooperation

It is interesting to note that the described node cooperation does not require any additional overhead communication of data (dataset entries) between the nodes. The only additional communication-overhead is that of having to exchange CSI between active receiving and transmitting nodes from $K\gamma/L+1$ groups. In static settings — where computing nodes are not moving fast, as one might expect to happen in data centers — and in particular in wired settings where the network coding coefficients are fixed and known, the CSI overhead can be very small compared to the volumes of the communicated datasets.

[5]Please note that to keep the notation simple, the indices may often – when there is no reason for confusion – appear without commas.

B. Impact of reduced packetization on distributed computing

The reduced subpacketization comes with a variety of positive ramifications.

a) Boosting the speedup factor t in the shuffling phase: As we have discussed, the much reduced subpacketization allows for a substantial increase in the number of nodes we can encode over, thus potentially yielding an L-fold decrease in the shuffling-phase delay.

b) Reducing packet overheads: The second ramification from having fewer packets, comes in the form of reduced header overheads that accompany each transmission. Simply put, the fewer the packets, the bigger they are, hence the less the communication load is dominated by header overheads which can further slow down the shuffling phase.

c) Efficient Coded Message Creation by Reducing Unevenness: Another positive ramification from our algorithm is that it can reduce the unevenness between the sizes of the mapped outputs that each packet is mapped into. This unevenness — which is naturally much more accentuated in smaller packets — can cause substantial additional delays because it forces zero padding (we can only combine equal-sized bit streams) which wastes communication resources. Having fewer and thus larger packets, averages out these size variations, thus reducing wasteful zero padding.

References

[1] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Communications of the ACM*, 2008.

[2] O. O'Malley, "Terabyte sort on apache hadoop," *Yahoo, available online at: http://sortbenchmark. org/Yahoo-Hadoop. pdf,(May)*, pp. 1–3, 2008.

[3] K. Shim, "MapReduce algorithms for big data analysis," *Proceedings of the VLDB Endowment*, vol. 5, no. 12, pp. 2016–2017, 2012.

[4] A. Kumar, M. Kiran, and B. Prathap, "Verification and validation of mapreduce program model for parallel k-means algorithm on hadoop cluster," in *Computing, Communications and Networking Technologies (ICCCNT), Fourth International Conference on*, IEEE, 2013.

[5] J. Dean and S. Ghemawat, "Distributed programming with mapreduce," *Beautiful Code. Sebastopol: O'Reilly Media, Inc*, vol. 384, 2007.

[6] J. Dean and S. Ghemawat, "MapReduce: a flexible data processing tool," *Communications of the ACM*, vol. 53, no. 1, pp. 72–77, 2010.

[7] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," *IEEE Transactions on Information Theory*, vol. 64, pp. 1514–1529, March 2018.

[8] Y. Chen, A. Ganapathi, R. Griffith, and R. Katz, "The case for evaluating MapReduce performance using workload suites," in *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2011 IEEE 19th International Symposium on*, 2011.

[9] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, "Coded MapReduce," in *53rd Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, Sept 2015.

[10] S. Li, S. Supittayapornpong, M. A. Maddah-Ali, and S. Avestimehr, "Coded TeraSort," in *IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, May 2017.

[11] S. Li, M. A. Maddah-Ali, Q. Yu, and A. S. Avestimehr, "A fundamental tradeoff between computation and communication in distributed computing," *IEEE Transactions on Information Theory*, Jan 2018.

[12] E. Lampiris and P. Elia, "Adding transmitters dramatically boosts coded-caching gains for finite file sizes." to appear in IEEE Journal on Special Areas of Communications (JSAC), 2018.

[13] S. P. Shariatpanahi, S. A. Motahari, and B. H. Khalaj, "Multi-server coded caching," *IEEE Transactions on Information Theory*, Dec 2016.