This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIT.2017.2756959, IEEE Transactions on Information Theory

A Fundamental Tradeoff between Computation and Communication in Distributed Computing

Songze Li, *Student Member, IEEE,* Mohammad Ali Maddah-Ali, *Member, IEEE,*
Qian Yu, *Student Member, IEEE,* and A. Salman Avestimehr, *Senior Member, IEEE*

***Abstract*—How can we optimally trade extra computing power to reduce the communication load in distributed computing? We answer this question by characterizing a fundamental tradeoff between computation and communication in distributed computing, i.e., the two are *inversely proportional* to each other.**

More specifically, a general distributed computing framework, motivated by commonly used structures like MapReduce, is considered, where the overall computation is decomposed into computing a set of "Map" and "Reduce" functions distributedly across multiple computing nodes. A coded scheme, named "Coded Distributed Computing" (CDC), is proposed to demonstrate that increasing the computation load of the Map functions by a factor of r (i.e., evaluating each function at r *carefully chosen* nodes) can create novel coding opportunities that reduce the communication load by the same factor.

An information-theoretic lower bound on the communication load is also provided, which matches the communication load achieved by the CDC scheme. As a result, the optimal computation-communication tradeoff in distributed computing is exactly characterized.

Finally, the coding techniques of CDC is applied to the Hadoop `TeraSort` benchmark to develop a novel `CodedTeraSort` algorithm, which is empirically demonstrated to speed up the overall job execution by $1.97\times$ - $3.39\times$, for typical settings of interest.

***Index Terms*—Distributed Computing, MapReduce, Computation-Communication Tradeoff, Coded Multicasting, Coded TeraSort**

I. Introduction

We consider a general distributed computing framework, motivated by prevalent structures like MapReduce [4] and Spark [5], in which the overall computation is decomposed into two stages: "Map" and "Reduce". Firstly in the Map stage, distributed computing nodes process parts of the input data locally, generating some intermediate values according to their designed Map functions. Next, they exchange the calculated intermediate values among each other (a.k.a. data shuffling), in order to calculate the final output results distributedly using their designed Reduce functions.

Within this framework, data shuffling often appears to limit the performance of distributed computing applications, including self-join [6], tera-sort [7], and machine learning algorithms [8]. For example, in a Facebook's Hadoop cluster, it is observed that 33% of the overall job execution time is spent on data shuffling [8]. Also as is observed in [9], 70% of the overall job execution time is spent on data shuffling when running a self-join application on an Amazon EC2 cluster [10]. As such motivated, we ask this fundamental question that *if coding can help distributed computing in reducing the load of communication and speeding up the overall computation?* Coding is known to be helpful in coping with the channel uncertainty in telecommunication and also in reducing the storage cost in distributed storage systems and cache networks. In this work, we extend the application of coding to *distributed computing* and propose a framework to substantially reduce the load of data shuffling via coding and some extra computing in the Map phase.

More specifically, we formulate and characterize a fundamental tradeoff relationship between "computation load" in the Map phase and "communication load" in the data shuffling phase, and demonstrate that the two are *inversely proportional* to each other. We propose an optimal coded scheme, named "Coded Distributed Computing" (CDC), which demonstrates that increasing the computation load of the Map phase by a factor of r (i.e., evaluating each Map function at r *carefully chosen* nodes) can create novel coding opportunities in the data shuffling phase that reduce the communication load by the same factor.

To illustrate our main result, consider a distributed computing framework to compute Q arbitrary output functions from N input files, using K distributed computing nodes. As mentioned earlier, the overall computation is performed by computing a set of Map and Reduce functions distributedly across the K nodes. In the Map phase, each input file is processed locally, in one of the nodes, to generate Q intermediate values, each corresponding to one of the Q output functions. Thus, at the end of this phase, QN intermediate values are calculated, which can be split into Q subsets of N intermediate values and each subset is needed to calculate

S. Li, Q. Yu and A.S. Avestimehr are with the Department of Electrical Engineering, University of Southern California, Los Angeles, CA, 90089, USA (e-mail: songzeli@usc.edu; qyu880@usc.edu; avestimehr@ee.usc.edu).

M. A. Maddah-Ali is with Department of Electrical Engineering, Sharif University of Technology, Tehran, 11365, Iran (e-mail: maddah_ali@sharif.edu).

A preliminary part of this work was presented in 53rd Annual Allerton Conference on Communication, Control, and Computing, 2015 [1]. A part of this work was presented in IEEE International Symposium on Information Theory, 2016 [2]. A part of this work was presented in the 6th International Workshop on Parallel and Distributed Computing for Large Scale Machine Learning and Big Data Analytics, 2017 [3].

This work is in part supported by NSF grants CCF-1408639, NETS-1419632, ONR award N000141612189, NSA Award No. H98230-16-C-0255, and a research gift from Intel. This material is based upon work supported by Defense Advanced Research Projects Agency (DARPA) under Contract No. HR001117C0053. The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIT.2017.2756959, IEEE Transactions on Information Theory

one of the output functions. In the Shuffle phase, for every output function to be calculated, all N intermediate values corresponding to that function are transferred to one of the nodes for reduction. Of course, depending on the node that has been chosen to reduce an output function, a part of the intermediate values are already available locally, and do not need to be transferred in the Shuffle phase. This is because that the Map phase has been carried out on the same set of nodes, and the results of mapping done at a node can remain in that node to be used for the Reduce phase. This offers some saving in the load of communication. To reduce the communication load even more, we may map each input file in *more than one* nodes. Apparently, this increases the fraction of intermediate values that are locally available. However, as we will show, there is a better way to exploit this redundancy in computation to reduce the communication load. The main message of this paper is to show that following a particular patten in repeating Map computations along with some coding techniques, we can significantly reduce the load of communication. Perhaps surprisingly, we show that the gain of coding in reducing communication load scales with the size of the network.

To be more precise, we define the *computation load* r, $1 \leq r \leq K$, as the total number of computed Map functions at the nodes, normalized by N. For example, $r = 1$ means that none of the Map functions has been re-computed, and $r = 2$ means that on average each Map function can be computed on two nodes. We also define *communication load* L, $0 \leq L \leq 1$, as the total amount of information exchanged across nodes in the shuffling phase, normalized by the size of QN intermediate values, in order to compute the Q output functions disjointly and uniformly across the K nodes. Based on this formulation, we now ask the following fundamental question:

- *Given a computation load r in the Map phase, what is the minimum communication load $L^*(r)$, using any data shuffling scheme, needed to compute the final output functions?*

We propose Coded Distributed Computing (CDC) that achieves a communication load of $L_{\text{coded}}(r) = \frac{1}{r} \cdot (1 - \frac{r}{K})$ for $r = 1, \ldots, K$, and the lower convex envelop of these points. CDC employs a specific strategy to assign the computations of the Map and Reduce functions across the computing nodes, in order to enable novel coding opportunities for data shuffling. In particular, for a computation load $r \in \{1, \ldots, K\}$, CDC utilizes a carefully designed repetitive mapping of data blocks at r distinct nodes to create coded multicast messages that deliver data *simultaneously* to a subset of $r \geq 1$ nodes. Hence, compared with an uncoded data shuffling scheme, which as we show later achieves a communication load $L_{\text{uncoded}}(r) = 1 - \frac{r}{K}$, CDC is able to reduce the communication load by exactly a factor of the computation load r. Furthermore, the proposed CDC scheme applies to a more general distributed computing framework where every output function is computed by more than one, or particularly $s \in \{1, \ldots, K\}$ nodes, which provides better fault-tolerance in distributed computing.

We numerically compare the computation-communication tradeoffs of CDC and uncoded data shuffling schemes (i.e., $L_{\text{coded}}(r)$ and $L_{\text{uncoded}}(r)$) in Fig. 1. As it is illustrated, in the uncoded scheme that achieves a communication load

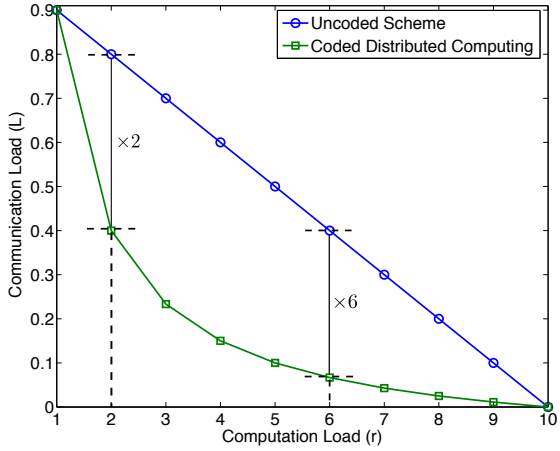


Fig. 1: Comparison of the communication load achieved by Coded Distributed Computing $L_{\text{coded}}(r)$ with that of the uncoded scheme $L_{\text{uncoded}}(r)$, for $Q = 10$ output functions, $N = 2520$ input files and $K = 10$ computing nodes. For $r \in \{1, \ldots, K\}$, CDC is r times better than the uncoded scheme.

$L_{\text{uncoded}}(r) = 1 - \frac{r}{K}$, increasing the computation load r offers only a modest reduction in communication load. In fact for any r, this gain vanishes for large number of nodes K. Consequently, it is not justified to trade computation for communication using uncoded schemes. However, for the coded scheme that achieves a communication load of $L_{\text{coded}}(r) = \frac{1}{r} \cdot (1 - \frac{r}{K})$, increasing the computation load r will significantly reduce the communication load, and this gain does not vanish for large K. For example as illustrated in Fig. 1, when mapping each file at one extra node ($r = 2$), CDC reduces the communication load by 55.6%, while the uncoded scheme only reduces it by 11.1%.

We also prove an information-theoretic lower bound on the minimum communication load $L^*(r)$. To prove the lower bound, we derive a lower bound on the total number of bits communicated by any subset of nodes, using induction on the size of the subset. To derive the lower bound for a particular subset of nodes, we first establish a lower bound on the number of bits needed by one of the nodes to recover the intermediate values it needs to calculate its assigned output functions, and then utilize the bound on the number of bits communicated by the rest of the nodes in that subset, which is given by the inductive argument. The derived lower bound on $L^*(r)$ matches the communication load achieved by the CDC scheme for any computation load $1 \leq r \leq K$. As a result, we *exactly* characterize the optimal tradeoff between computation load and communication load in the following:

$$L^*(r) = L_{\text{coded}}(r) = \frac{1}{r} \cdot (1 - \frac{r}{K}), \; r \in \{1, \ldots, K\}.$$

For general $1 \leq r \leq K$, $L^*(r)$ is the lower convex envelop of the above points $\{(r, L_{\text{coded}}(r)) : r \in \{1, \ldots, K\}\}$. Note that for large K, $\frac{1}{r} \cdot (1 - \frac{r}{K}) \approx \frac{1}{r}$, hence $L^*(r) \approx \frac{1}{r}$. This result reveals a fundamental inversely proportional relationship between computation load and communication load in distributed computing. This also illustrates that the gain of $\frac{1}{r}$ achieved by CDC is optimal and it cannot be improved by any

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIT.2017.2756959, IEEE Transactions on Information Theory

other scheme (since $L_{\text{coded}}(r)$ is an information-theoretic lower bound on $L^*(r)$ that applies to any data shuffling scheme).

Having theoretically characterized the optimal computation-communication tradeoff achieved by the proposed CDC scheme, we also empirically demonstrate the practical impact of this tradeoff. In particular, we apply the coding techniques of CDC to a widely used Hadoop sorting benchmark `TeraSort` [11], developing a novel coded distributed sorting algorithm `CodedTeraSort` [3]. We perform extensive experiments on Amazon EC2 clusters, and observe that for typical settings of interest, `CodedTeraSort` speeds up the overall execution of the conventional `TeraSort` by a factor of $1.97\times$ - $3.39\times$.

Finally, we discuss some future directions to extend the results of this work. In particular, we consider topics including heterogeneous networks with asymmetric tasks, straggling/failing computing nodes, multi-stage computation tasks, multi-layer networks and structured topology, joint storage and computation optimization, and coded edge/fog computing.

Related Works. The problem of characterizing the minimum communication for distributed computing has been previously considered in several settings in both computer science and information theory communities. In [12], a basic computing model is proposed, where two parities have x and y and aim to compute a boolean function $f(x, y)$ by exchanging the minimum number of bits between them. Also, the problem of minimizing the required communication for computing the modulo-two sum of distributed binary sources with symmetric joint distribution was introduced in [13]. Following these two seminal works, a wide range of communication problems in the scope of distributed computing have been studied (see, e.g., [14]–[19]). The key differences distinguishing the setting in this paper from most of the prior ones are 1) We focus on the flow of communication in a general distributed computing framework, motivated by MapReduce, rather than the structures of the functions or the input distributions. 2) We do not impose any constraint on the numbers of output results, input data files and computing nodes (they can be arbitrarily large), 3) We do not assume any special property (e.g. linearity) of the computed functions.

The idea of efficiently creating and exploiting *coded multicasting* was initially proposed in the context of cache networks in [20], [21], and extended in [22], [23], where caches prefetch part of the content in a way to enable coding during the content delivery, minimizing the network traffic. In this paper, we propose a framework to study the tradeoff between computation and communication in distributed computing. We demonstrate that the coded multicasting opportunities exploited in the above caching problems also exist in the data shuffling of distributed computing frameworks, which can be created by a strategy of repeating the computations of the Map functions specified by the Coded Distributed Computing (CDC) scheme.

Finally, in a recent work [24], the authors have proposed methods for utilizing codes to speed up some specific distributed machine learning algorithms. The considered problem in this paper differs from [24] in the following aspects. We propose a general methodology for utilizing coding in data shuffling that can be applied to any distributed computing framework with a MapReduce structure, regardless of the underlying application. In other words, any distributed computing algorithm that fits in the MapReduce framework can benefit from the proposed CDC solution. We also characterize the information-theoretic computation-communication tradeoff in such frameworks. Furthermore, the coding used in [24] is at the application layer (i.e., applying computation on coded data), while in this paper we focus on applying codes directly on the shuffled data.

II. Problem Formulation

In this section, we formulate a general distributed computing framework motivated by MapReduce, and define the function characterizing the tradeoff between computation and communication.

We consider the problem of computing Q arbitrary output functions from N input files using a cluster of K distributed computing nodes (servers), for some positive integers $Q, N, K \in \mathbb{N}$, with $N \geq K$. More specifically, given N input files $w_1, \ldots, w_N \in \mathbb{F}_{2^F}$, for some $F \in \mathbb{N}$, the goal is to compute Q output functions $\phi_1, \ldots, \phi_Q$, where $\phi_q : (\mathbb{F}_{2^F})^N \to \mathbb{F}_{2^B}$, $q \in \{1, \ldots, Q\}$ maps all input files to a length-B binary stream $u_q = \phi_q(w_1, \ldots, w_N) \in \mathbb{F}_{2^B}$, for some $B \in \mathbb{N}$.

Motivated by MapReduce, we assume that as illustrated in Fig. 2 the computation of the output function ϕ_q, $q \in \{1, \ldots, Q\}$ can be decomposed as follows:

$$\phi_q(w_1, \ldots, w_N) = h_q(g_{q,1}(w_1), \ldots, g_{q,N}(w_N)), \quad (1)$$

where

- The "Map" functions $\vec{g}_n = (g_{1,n}, \ldots, g_{Q,n}) : \mathbb{F}_{2^F} \to (\mathbb{F}_{2^T})^Q$, $n \in \{1, \ldots, N\}$ maps the input file w_n into Q length-T *intermediate values* $v_{q,n} = g_{q,n}(w_n) \in \mathbb{F}_{2^T}$, $q \in \{1, \ldots, Q\}$, for some $T \in \mathbb{N}$.[1]
- The "Reduce" functions $h_q : (\mathbb{F}_{2^T})^N \to \mathbb{F}_{2^B}$, $q \in \{1, \ldots, Q\}$ maps the intermediate values of the output function ϕ_q in all input files into the output value $u_q = h_q(v_{q,1}, \ldots, v_{q,N})$.

Remark 1. Note that for every set of output functions $\phi_1, \ldots, \phi_Q$ such a Map-Reduce decomposition exists (e.g., setting $g_{q,n}{}'s$ to identity functions such that $g_{q,n}(w_n) = w_n$ for all $n = 1, \ldots, N$, and h_q to ϕ_q in (1)). However, such a decomposition is not unique, and in the distributed computing literature, there has been quite some work on developing appropriate decompositions of computations like join, sorting and matrix multiplication (see, e.g., [4], [25]), for them to be performed efficiently in a distributed manner. Here we do not impose any constraint on how the Map and Reduce functions

[1]When mapping a file, we compute Q intermediate values in parallel, one for each of the Q output functions. The main reason to do this is that parallel processing can be efficiently performed for applications that fit into the MapReduce framework. In other words, mapping a file according to one function is only marginally more expensive than mapping according to all functions. For example, for the canonical Word Count job, while we are scanning a document to count the number of appearances of one word, we can simultaneously count the numbers of appearances of other words with marginally increased computation cost.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIT.2017.2756959, IEEE Transactions on Information Theory

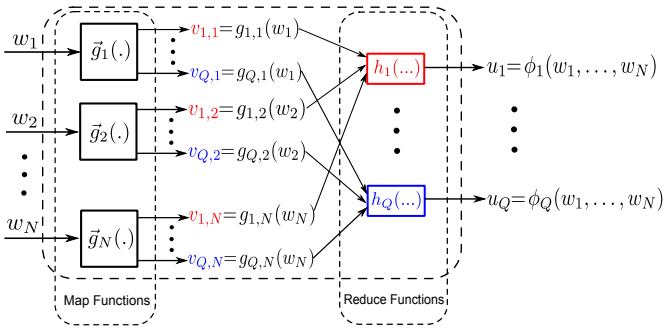


Fig. 2: Illustration of a two-stage distributed computing framework. The overall computation is decomposed into computing a set of Map and Reduce functions.

are chosen (for example, they can be arbitrary linear or non-linear functions). □

The above computation is carried out by K distributed computing nodes, labelled as Node $1,\ldots,$Node K. They are interconnected through a multicast network. Following the above decomposition, the computation proceeds in three phases: *Map*, *Shuffle* and *Reduce*.

Map Phase: Node k, $k \in \{1,\ldots,K\}$ computes the Map functions of a set of files $\mathcal{M}_k$, which are stored on Node k, for some design parameter $\mathcal{M}_k \subseteq \{w_1,\ldots,w_N\}$. For each file w_n in $\mathcal{M}_k$, Node k computes $\vec{g}_n(w_n)=(v_{1,n},\ldots,v_{Q,n})$. We assume that each file is mapped by at least one node, i.e., $\bigcup_{k=1,\ldots,K}\mathcal{M}_k = \{w_1,\ldots,w_N\}$.

Definition 1 (Computation Load)**.** We define the *computation load*, denoted by r, $1 \le r \le K$, as the total number of Map functions computed across the K nodes, normalized by the number of files N, i.e., $r \triangleq \frac{\sum_{k=1}^{K}|\mathcal{M}_k|}{N}$. The computation load r can be interpreted as the average number of nodes that map each file. ◊

Shuffle Phase: Node k, $k \in \{1,\ldots,K\}$ is responsible for computing a subset of output functions, whose indices are denoted by a set $\mathcal{W}_k \subseteq \{1,\ldots,Q\}$. We focus on the case $\frac{Q}{K} \in \mathbb{N}$, and utilize a *symmetric* task assignment across the K nodes to maintain load balance. More precisely, we require 1) $|\mathcal{W}_1| = \cdots = |\mathcal{W}_K| = \frac{Q}{K}$, 2) $\mathcal{W}_j \cap \mathcal{W}_k = \varnothing$ for all $j \neq k$.

Remark 2. Beyond the symmetric task assignment considered in this paper, characterizing the optimal computation-communication tradeoff allowing general asymmetric task assignments is a challenging open problem. As the first step to study this problem, in our follow-up work [26] in which the number of output functions Q is fixed and the computing resources are abundant (e.g., number of computing nodes $K \gg Q$), we have shown that asymmetric task assignments can do better than the symmetric ones, and achieve the optimum run-time performance. □

To compute the output value u_q for some $q \in \mathcal{W}_k$, Node k needs the intermediate values that are *not* computed *locally* in the Map phase, i.e., $\{v_{q,n} : q \in \mathcal{W}_k, w_n \notin \mathcal{M}_k\}$. After Node k, $k \in \{1,\ldots,K\}$ has finished mapping all the files in $\mathcal{M}_k$, the K nodes proceed to exchange the needed intermediate values. In particular, each node k, $k \in \{1,\ldots,K\}$, creates an input symbol $X_k \in \mathbb{F}_{2^{\ell_k}}$, for some $\ell_k \in \mathbb{N}$, as a function of the intermediate values computed locally during the Map phase, i.e., for some encoding function $\psi_k : (\mathbb{F}_{2^T})^{Q|\mathcal{M}_k|} \to \mathbb{F}_{2^{\ell_k}}$ at Node k, we have

$$X_k = \psi_k\left(\{\vec{g}_n : w_n \in \mathcal{M}_k\}\right). \tag{2}$$

Having generated the message X_k, Node k multicasts it to all other nodes.

By the end of the Shuffle phase, each of the K nodes receives $X_1,\ldots,X_K$ free of error.

Definition 2 (Communication Load)**.** We define the *communication load*, denoted by L, $0 \le L \le 1$, as $L \triangleq \frac{\ell_1+\cdots+\ell_K}{QNT}$. That is, L represents the (normalized) total number of bits communicated by the K nodes during the Shuffle phase.[2] ◊

Reduce Phase: Node k, $k \in \{1,\ldots,K\}$, uses the messages $X_1,\ldots,X_K$ communicated in the Shuffle phase, and the local results from the Map phase $\{\vec{g}_n : w_n \in \mathcal{M}_k\}$ to construct inputs to the corresponding Reduce functions of $\mathcal{W}_k$, i.e., for each $q \in \mathcal{W}_k$ and some decoding function $\chi_k^q : \mathbb{F}_{2^{\ell_1}} \times \cdots \times \mathbb{F}_{2^{\ell_K}} \times (\mathbb{F}_{2^T})^{Q|\mathcal{M}_k|} \to (\mathbb{F}_{2^T})^N$, Node k computes

$$(v_{q,1},\ldots,v_{q,N}) = \chi_k^q\left(X_1,\ldots,X_K,\{\vec{g}_n : w_n \in \mathcal{M}_k\}\right). \tag{3}$$

Finally, Node k, $k \in \{1,\ldots,K\}$, computes the Reduce function $u_q = h_q(v_{q,1}\ldots v_{q,N})$ for all $q \in \mathcal{W}_k$.

We say that a computation-communication pair $(r, L) \in \mathbb{R}^2$ is *feasible* if for any $\delta > 0$ and sufficiently large N, there exist $\mathcal{M}_1,\ldots,\mathcal{M}_K, \mathcal{W}_1,\ldots,\mathcal{W}_K$, a set of encoding functions $\{\psi_k\}_{k=1}^K$, and a set of decoding functions $\{\chi_k^q : q \in \mathcal{W}_k\}_{k=1}^K$ that achieve a computation-communication pair $(\tilde{r}, \tilde{L}) \in \mathbb{Q}^2$ such that $|r-\tilde{r}| \le \delta$, $|L-\tilde{L}| \le \delta$, and Node k can successfully compute all the output functions whose indices are in $\mathcal{W}_k$, for all $k \in \{1,\ldots,K\}$.

Definition 3. We define the *computation-communication function* of the distributed computing framework

$$L^*(r) \triangleq \inf\{L : (r, L) \text{ is feasible}\}. \tag{4}$$

$L^*(r)$ characterizes the optimal tradeoff between computation and communication in this framework. ◊

Example (Uncoded Scheme). In the Shuffle phase of a simple "uncoded" scheme, each node receives the needed intermediate values sent uncodedly by some other nodes. Since a total of QN intermediate values are needed across the K nodes and $rN \cdot \frac{Q}{K} = \frac{rQN}{K}$ of them are already available after the Map phase, the communication load achieved by the uncoded scheme

$$L_{\text{uncoded}}(r) = 1 - r/K. \tag{5}$$

Remark 3. After the Map phase, each node knows the intermediate values of all Q output functions in the files it

[2]For notational convenience, we define all variables in binary extension fields. However, one can consider arbitrary field sizes. For example, we can consider all intermediate values $v_{q,n}$, $q = 1,\ldots,Q$, $n = 1,\ldots,N$, to be in the field $\mathbb{F}_{p^T}$, for some prime number p and positive integer T, and the symbol communicated by Node k (i.e., X_k), to be in the field $\mathbb{F}_{s^{\ell_k}}$ for some prime number s and positive integer ℓ_k, for all $k = 1,\ldots,K$. In this case, the communication load can be defined as $L \triangleq \frac{(\ell_1+\cdots+\ell_K)\log s}{QNT\log p}$.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIT.2017.2756959, IEEE Transactions on Information Theory

has mapped. Therefore, for a fixed file assignment and any symmetric assignment of the Reduce functions, specified by $\mathcal{W}_1, \ldots, \mathcal{W}_K$, we can satisfy the data requirements using the same data shuffling scheme up to relabelling the Reduce functions. In other words, the communication load is independent of the assignment of the Reduce functions. □

In this paper, we also consider a generalization of the above framework, which we call "cascaded distributed computing framework", where after the Map phase, each Reduce function is computed by more than one, or particularly s nodes, for some $s \in \{1, \ldots, K\}$. This generalized model is motivated by the fact that many distributed computing jobs require multiple rounds of Map and Reduce computations, where the Reduce results of the previous round serve as the inputs to the Map functions of the next round. Computing each Reduce function at more than one node admits *data redundancy* for the subsequent Map-function computations, which can help to improve the fault-tolerance and reduce the communication load of the next-round data shuffling. We focus on the case $\frac{Q}{\binom{K}{s}} \in \mathbb{N}$, and enforce a symmetric assignment of the Reduce tasks to maintain load balance. Particularly, we require that every subset of s nodes compute a disjoint subset of $\frac{Q}{\binom{K}{s}}$ Reduce functions.

The feasible computation-communication triple $(r, s, L) \in \mathbb{R} \times \mathbb{N} \times \mathbb{R}$ is defined similar as before. We define the computation-communication function of the cascaded distributed computing framework

$$L^*(r, s) \triangleq \inf\{L : (r, s, L) \text{ is feasible}\}. \tag{6}$$

III. Main Results

Theorem 1. *The computation-communication function of the distributed computing framework, $L^*(r)$ is given by*

$$L^*(r) = L_{\text{coded}}(r) \triangleq \tfrac{1}{r} \cdot (1 - \tfrac{r}{K}), \quad r \in \{1, \ldots, K\}, \tag{7}$$

for sufficiently large T. For general $1 \le r \le K$, $L^*(r)$ is the lower convex envelop of the above points $\{(r, \frac{1}{r} \cdot (1 - \frac{r}{K})) : r \in \{1, \ldots, K\}\}$.

We prove the achievability of Theorem 1 by proposing a *coded* scheme, named Coded Distributed Computing, in Section V. We demonstrate that no other scheme can achieve a communication load smaller than the lower convex envelop of the points $\{(r, \frac{1}{r} \cdot (1 - \frac{r}{K})) : r \in \{1, \ldots, K\}\}$ by proving the converse in Section VI.

Remark 4. Theorem 1 exactly characterizes the optimal trade-off between the computation load and the communication load in the considered distributed computing framework. □

Remark 5. For $r \in \{1, \ldots, K\}$, the communication load achieved in Theorem 1 is less than that of the uncoded scheme in (5) by a multiplicative factor of r, which equals the computation load and can grow unboundedly as the number of nodes K increases if e.g. $r = \Theta(K)$. As illustrated in Fig. 1 in Section I, while the communication load of the uncoded scheme decreases linearly as the computation load increases, $L_{\text{coded}}(r)$ achieved in Theorem 1 is inversely proportional to the computation load. □

Remark 6. While increasing the computation load r causes a longer Map phase, the coded achievable scheme of Theorem 1 maximizes the reduction of the communication load using the extra computations. Therefore, Theorem 1 provides an analytical framework to optimally trading the computation power in the Map phase for more bandwidth in the Shuffle phase, which helps to minimize the overall execution time of applications whose performances are limited by data shuffling. □

Theorem 2. *The computation-communication function of the cascaded distributed computing framework, $L^*(r, s)$, for $r \in \{1, \ldots, K\}$, is characterized by*

$$L^*(r, s) = L_{\text{coded}}(r, s) \triangleq \sum_{\ell=\max\{r+1,s\}}^{\min\{r+s,K\}} \frac{\ell \binom{K}{\ell} \binom{\ell-2}{r-1} \binom{r}{\ell-s}}{r \binom{K}{r} \binom{K}{s}}, \tag{8}$$

for some $s \in \{1, \ldots, K\}$ and sufficiently large T. For general $1 \le r \le K$, $L^*(r, s)$ is the lower convex envelop of the above points $\{(r, L_{\text{coded}}(r, s)) : r \in \{1, \ldots, K\}\}$.

We present the Coded Distributed Computing scheme that achieves the computation-communication function in Theorem 2 in Section V, and the converse of Theorem 2 in Section VII.

Remark 7. A preliminary part of this result, in particular the achievability for the special case of $s = 1$, or the achievable scheme of Theorem 1 was presented in [1]. We note that when $s = 1$, Theorem 2 provides the same result as in Theorem 1, i.e., $L^*(r, 1) = \frac{1}{r} \cdot (1 - \frac{r}{K})$, for $r \in \{1, \ldots, K\}$. □

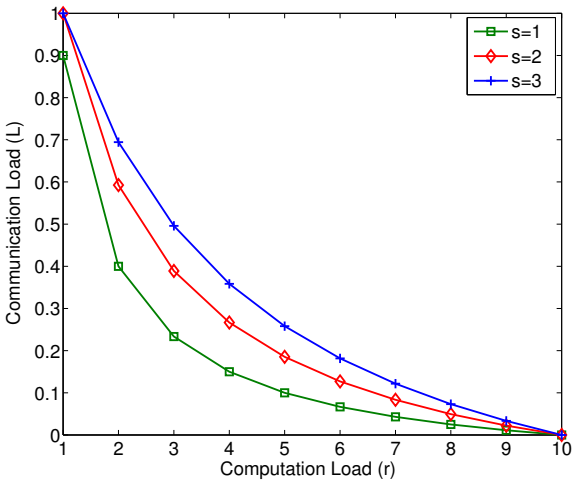


Fig. 3: Minimum communication load $L^*(r, s) = L_{\text{coded}}(r, s)$ in Theorem 2, for $Q = 360$ output functions, $N = 2520$ input files and $K = 10$ computing nodes.

Remark 8. For any fixed $s \in \{1, \ldots, K\}$ (number of nodes that compute each Reduce function), as illustrated in Fig. 3, the communication load achieved in Theorem 2 outperforms the linear relationship between computation and communication, i.e., it is superlinear with respect to the computation load r. □

Before we proceed to describe the general achievability scheme for the cascaded distributed computing framework

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIT.2017.2756959, IEEE Transactions on Information Theory

(also the distributed computing framework as a special case of $s = 1$), we first illustrate the key ideas of the proposed Coded Distributed Computing scheme by presenting two examples in the next section, for the cases of $s = 1$ and $s > 1$ respectively.

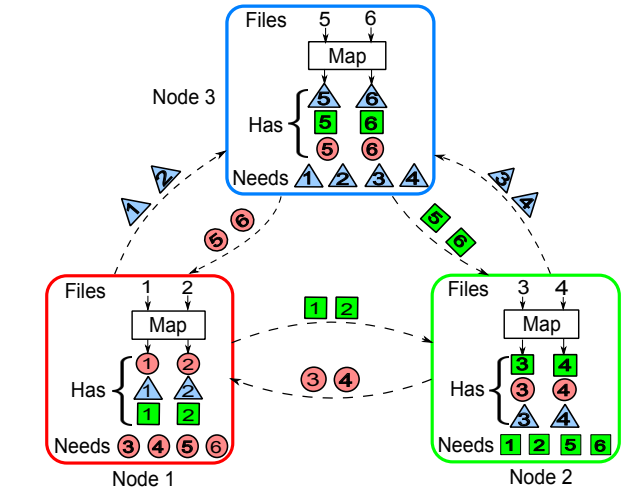
IV. Illustrative Examples: Coded Distributed Computing

In this section, we present two illustrative examples of the proposed achievable scheme for Theorem 1 and Theorem 2, which we call Coded Distributed Computing (CDC), for the cases of $s = 1$ (Theorem 1) and $s > 1$ (Theorem 2) respectively.

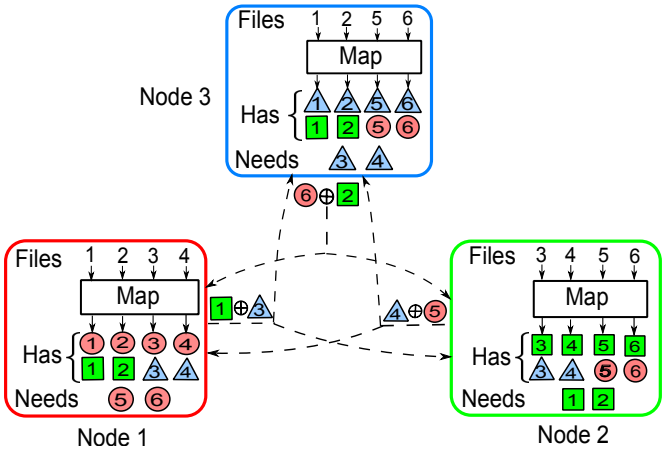
Example 1 (CDC for $s = 1$)**.** We consider a MapReduce-type problem in Fig. 4 for distributed computing of $Q = 3$ output functions, represented by red/circle, green/square, and blue/triangle respectively, from $N = 6$ input files, using $K = 3$ computing nodes. Nodes 1, 2, and 3 are respectively responsible for final reduction of red/circle, green/square, and blue/triangle output functions. Let us first consider the case where no redundancy is imposed on the computations, i.e., each file is mapped once and computation load $r = 1$. As shown in Fig. 4(a), Node k maps File $2k - 1$ and File $2k$ for $k = 1, 2, 3$. In this case, each node maps 2 input files locally, computing all three intermediate values needed for the three output functions from each mapped file. In Fig. 4, we represent, for example, the intermediate value of the red/circle function in File n using a red circle labelled by n, for all $n = 1, \ldots, 6$. Similar representations follow for the green/square and the blue/triangle functions. After the Map phase, each node obtains 2 out of 6 required intermediate values to reduce the output function it is responsible for (e.g., Node 1 knows the red circles in File 1 and File 2). Hence, each node needs 4 intermediate values from the other nodes, yielding a communication load of $\frac{4\times3}{3\times6} = \frac{2}{3}$.

Now, we demonstrate how the proposed CDC scheme trades the computation load to slash the communication load via in-network coding. As shown in Fig. 4(b), we double the computation load such that each file is now mapped on two nodes ($r = 2$). It is apparent that since more local computations are performed, each node now only requires 2 other intermediate values, and an uncoded shuffling scheme would achieve a communication load of $\frac{2\times3}{3\times6} = \frac{1}{3}$. However, we can do much better with coding. As shown in Fig. 4(b), instead of unicasting individual intermediate values, every node multicasts a bit-wise XOR, denoted by $\oplus$, of 2 locally computed intermediate values to the other two nodes, simultaneously satisfying their data demands. For example, knowing the blue/triangle in File 3, Node 2 can cancel it from the coded packet sent by Node 1, recovering the needed green/square in File 1. Therefore, this coding incurs a communication load of $\frac{3}{3\times6} = \frac{1}{6}$, achieving a $2\times$ gain from the uncoded shuffling. □

From the above example, we see that for the case of $s = 1$, i.e., each of the Q output functions is computed on one node and the computations of the Reduce functions are symmetrically distributed across nodes, the proposed CDC scheme only requires performing bit-wise XOR as the encoding and decoding operations. However, for the case of $s > 1$, as we will show in the following example, the proposed CDC scheme requires computing linear combinations of the intermediate values during the encoding process.



(a) Uncoded Distributed Computing Scheme.



(b) Coded Distributed Computing Scheme.

Fig. 4: Illustrations of the conventional uncoded distributed computing scheme with computation load $r = 1$, and the proposed Coded Distributed Computing scheme with computation load $r = 2$, for computing $Q = 3$ functions from $N = 6$ inputs on $K = 3$ nodes.

Example 2 (CDC for $s > 1$)**.** In this example, we consider a job of computing $Q = 6$ output functions from $N = 6$ input files, using $K = 4$ nodes. We focus on the case where the computation load $r = 2$, and each Reduce function is computed by $s = 2$ nodes. In the Map phase, each file is mapped by $r = 2$ nodes. As shown in Fig. 5, the sets of the files mapped by the 4 nodes are $\mathcal{M}_1 = \{w_1, w_2, w_3\}$, $\mathcal{M}_2 = \{w_1, w_4, w_5\}$, $\mathcal{M}_3 = \{w_2, w_4, w_6\}$, and $\mathcal{M}_4 = \{w_3, w_5, w_6\}$. After the Map phase, Node k, $k \in \{1, 2, 3, 4\}$, knows the intermediate values of all $Q = 6$ output functions in the files in $\mathcal{M}_k$, i.e., $\{v_{q,n} : q \in \{1, \ldots, 6\}, w_n \in \mathcal{M}_k\}$. In the Reduce phase, we assign the computations of the Reduce functions in a symmetric manner such that every subset of $s = 2$ nodes compute a common Reduce function. More

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIT.2017.2756959, IEEE Transactions on Information Theory

specifically as shown in Fig. 5, the sets of indices of the Reduce functions computed by the 4 nodes are $\mathcal{W}_1 = \{1, 2, 3\}$, $\mathcal{W}_2 = \{1, 4, 5\}$, $\mathcal{W}_3 = \{2, 4, 6\}$, and $\mathcal{W}_4 = \{3, 5, 6\}$. Therefore, for example, Node 1 still needs the intermediate values $\{v_{q,n} : q \in \{1, 2, 3\}, n \in \{4, 5, 6\}\}$ through data shuffling to compute its assigned Reduce functions h_1, h_2, h_3.

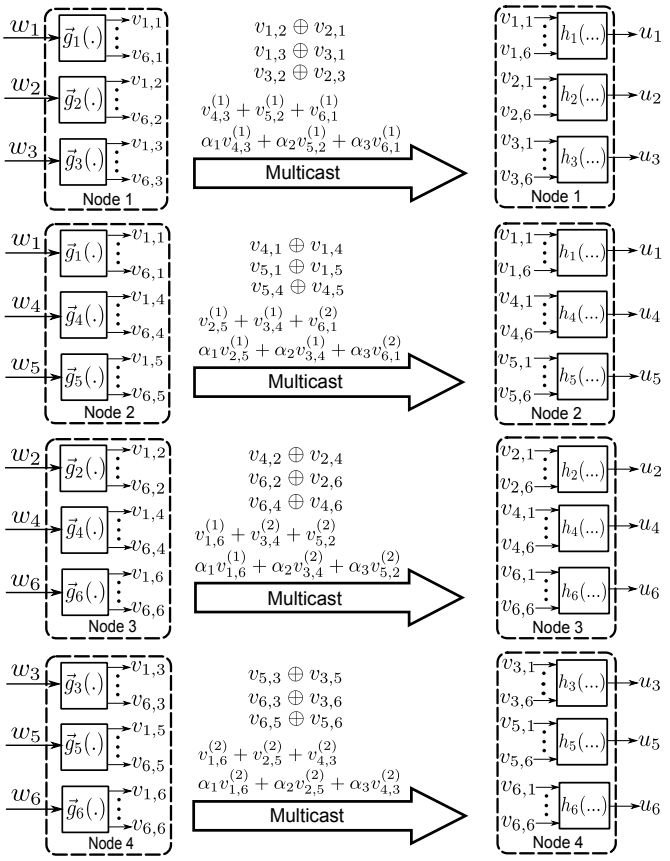


Fig. 5: Illustration of the CDC scheme to compute $Q = 6$ output functions from $N = 6$ input files distributedly at $K = 4$ computing nodes. Each file is mapped by $r = 2$ nodes and each output function is computed by $s = 2$ nodes. After the Map phase, every node knows 6 intermediate values, one for each output function, in every file it has mapped. The Shuffle phase proceeds in two rounds. In the first round, each node multicasts bit-wise XOR of intermediate values to subsets of two nodes. In the second round, each node splits an intermediate value $v_{q,n}$ evenly into two segments $v_{q,n} = (v_{q,n}^{(1)}, v_{q,n}^{(2)})$, and multicasts two linear combinations of the segments that are cosntructed using coefficients α_1, α_2, and α_3 to the other three nodes.

The data shuffling process consists of two rounds of communication over the multicast network. In the first round, intermediate values are communicated within each subset of 3 nodes. In the second round, intermediate values are communicated within the set of all 4 nodes. In what follows, we describe these two rounds of communication respectively.

Round 1: Subsets of 3 *nodes.* We first consider the subset $\{1, 2, 3\}$. During the data shuffling, each node whose index is in $\{1, 2, 3\}$ multicasts a bit-wise XOR of two locally computed intermediate values to the other two nodes:

- Node 1 multicasts $v_{1,2} \oplus v_{2,1}$ to Node 2 and Node 3,
- Node 2 multicasts $v_{4,1} \oplus v_{1,4}$ to Node 1 and Node 3,
- Node 3 mulicasts $v_{4,2} \oplus v_{2,4}$ to Node 1 and Node 2,

Since Node 2 knows $v_{2,1}$ and Node 3 knows $v_{1,2}$ locally, they can respectively decode $v_{1,2}$ and $v_{2,1}$ from the coded message $v_{1,2} \oplus v_{2,1}$.

We employ the similar coded shuffling scheme on the other 3 subsets of 3 nodes. After the first round of shuffling,

- Node 1 recovers $(v_{1,4}, v_{1,5})$, $(v_{2,4}, v_{2,6})$ and $(v_{3,5}, v_{3,6})$,
- Node 2 recovers $(v_{1,2}, v_{1,3})$, $(v_{4,2}, v_{4,6})$ and $(v_{5,3}, v_{5,6})$,
- Node 3 recovers $(v_{2,1}, v_{2,3})$, $(v_{4,1}, v_{4,5})$ and $(v_{6,3}, v_{6,5})$,
- Node 4 recovers $(v_{3,1}, v_{3,2})$, $(v_{5,1}, v_{5,4})$ and $(v_{6,2}, v_{6,4})$.

Round 2: All 4 *nodes.* We first split each of the intermediate values $v_{6,1}$, $v_{5,2}$, $v_{4,3}$, $v_{3,4}$, $v_{2,5}$, and $v_{1,6}$ into two equal-sized segments each containing $T/2$ bits, which are denoted by $v_{q,n}^{(1)}$ and $v_{q,n}^{(2)}$ for an intermediate value $v_{q,n}$. Then, for some coefficients $\alpha_1, \alpha_2, \alpha_3 \in \mathbb{F}_{2^{\frac{T}{2}}}$, Node 1 multicasts the following two linear combinations of three locally computed segments to the other three nodes.

$$v_{4,3}^{(1)} + v_{5,2}^{(1)} + v_{6,1}^{(1)}, \tag{9}$$

$$\alpha_1 v_{4,3}^{(1)} + \alpha_2 v_{5,2}^{(1)} + \alpha_3 v_{6,1}^{(1)}. \tag{10}$$

Similarly, as shown in Fig. 5, each of Node 2, Node 3, and Node 4 multicasts two linear combinations of three locally computed segments to the other three nodes, using the same coefficients α_1, α_2, and α_3.

Having received the above two linear combinations, each of Node 2, Node 3, and Node 4 first subtracts out one segment available locally from the combinations, or more specifically, $v_{6,1}^{(1)}$ for Node 2, $v_{5,2}^{(1)}$ for Node 3, and $v_{4,3}^{(1)}$ for Node 4. After the subtraction, each of these three nodes recovers the required segments from the two linear combinations. More specifically, Node 2 recovers $v_{4,3}^{(1)}$ and $v_{5,2}^{(1)}$, Node 3 recovers $v_{4,3}^{(1)}$ and $v_{6,1}^{(1)}$, and Node 4 recovers $v_{5,2}^{(1)}$ and $v_{6,1}^{(1)}$. It is not difficult to see that the above decoding process is guaranteed to be successful if α_1, α_2, and α_3 are all distinct from each other, which requires the field size $2^{\frac{T}{2}} \geq 3$ (e.g., $T = 4$). Following the similar procedure, each node recovers the required segments from the linear combinations multicast by the other three nodes. More specifically, after the second round of data shuffling,

- Node 1 recovers $v_{1,6}$, $v_{2,5}$ and $v_{3,4}$,
- Node 2 recovers $v_{1,6}$, $v_{4,3}$ and $v_{5,2}$,
- Node 3 recovers $v_{2,5}$, $v_{4,3}$ and $v_{6,1}$,
- Node 4 recovers $v_{3,4}$, $v_{5,2}$ and $v_{6,1}$.

We finally note that in the second round of data shuffling, each linear combination multicast by a node is simultaneously useful for the rest of the three nodes. □

V. General Achievable Scheme: Coded Distributed Computing

In this section, we formally prove the upper bounds in Theorem 1 and 2 by presenting and analyzing the Coded Distributed Computing (CDC) scheme. We focus on the more general case considered in Theorem 2 with $s \geq 1$, and the scheme for Theorem 1 simply follows by setting $s = 1$.

We first consider the integer-valued computation load $r \in \{1, \ldots, K\}$, and then generalize the CDC scheme for any $1 \leq r \leq K$. When $r = K$, every node can map all the input files and compute all the output functions locally,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIT.2017.2756959, IEEE Transactions on Information Theory

thus no communication is needed and $L^*(K,s)=0$ for all $s\in\{1,\ldots,K\}$. In what follows, we focus on the case where $r<K$.

We consider sufficiently large number of input files N, and $\binom{K}{r}(\eta_1-1)<N\leq\binom{K}{r}\eta_1$, for some $\eta_1\in\mathbb{N}$. We first inject $\binom{K}{r}\eta_1-N$ empty files into the system to obtain a total of $\bar{N}=\binom{K}{r}\eta_1$ files, which is now a multiple of of $\binom{K}{r}$. We note that $\lim_{N\to\infty}\frac{\bar{N}}{N}=1$. Next, we proceed to present the achievable scheme for a system with $\bar{N}$ input files $w_1,\ldots,w_{\bar{N}}$.

A. Map Phase Design

In the Map phase the $\bar{N}$ input files are evenly partitioned into $\binom{K}{r}$ disjoint batches of size η_1, each corresponding to a subset $\mathcal{T}\subset\{1,\ldots,K\}$ of size r, i.e.,

$$\{w_1,\ldots,w_{\bar{N}}\}=\bigcup_{\mathcal{T}\subset\{1,\ldots,K\},|\mathcal{T}|=r}\mathcal{B}_{\mathcal{T}},\tag{11}$$

where $\mathcal{B}_{\mathcal{T}}$ denotes the batch of η_1 files corresponding to the subset $\mathcal{T}$.

Given this partition, Node k, $k\in\{1,\ldots,K\}$, computes the Map functions of the files in $\mathcal{B}_{\mathcal{T}}$ if $k\in\mathcal{T}$. Or equivalently, $\mathcal{B}_{\mathcal{T}}\subseteq\mathcal{M}_k$ if $k\in\mathcal{T}$. Since each node is in $\binom{K-1}{r-1}$ subsets of size r, each node computes $\binom{K-1}{r-1}\eta_1=\frac{r\bar{N}}{K}$ Map functions, i.e., $|\mathcal{M}_k|=\frac{r\bar{N}}{K}$ for all $k\in\{1,\ldots,K\}$. After the Map phase, Node k, $k\in\{1,\ldots,K\}$, knows the intermediate values of all Q output functions in the files in $\mathcal{M}_k$, i.e., $\{v_{q,n}:q\in\{1,\ldots,Q\},w_n\in\mathcal{M}_k\}$.

B. Coded Data Shuffling

We recall that we focus on the case where the number of the output functions Q satisfies $\frac{Q}{\binom{K}{s}}\in\mathbb{N}$, and enforce a symmetric assignment of the Reduce functions such that every subset of s nodes reduce $\frac{Q}{\binom{K}{s}}$ functions. Specifically, $Q=\binom{K}{s}\eta_2$ for some $\eta_2\in\mathbb{N}$, and the computations of the Reduce functions are assigned symmetrically across the K nodes as follows. Firstly the Q Reduce functions are evenly partitioned into $\binom{K}{s}$ disjoint batches of size η_2, each corresponding to a unique subset $\mathcal{P}$ of s nodes, i.e.,

$$\{1,\ldots,Q\}=\bigcup_{\mathcal{P}\subseteq\{1,\ldots,K\},|\mathcal{P}|=s}\mathcal{D}_{\mathcal{P}},\tag{12}$$

where $\mathcal{D}_{\mathcal{P}}$ denotes the indices of the batch of η_2 Reduce functions corresponding to the subset $\mathcal{P}$.

Given this partition, Node k, $k\in\{1,\ldots,K\}$, computes the Reduce functions whose indices are in $\mathcal{D}_{\mathcal{P}}$ if $k\in\mathcal{P}$. Or equivalently, $\mathcal{D}_{\mathcal{P}}\subseteq\mathcal{W}_k$ if $k\in\mathcal{P}$. As a result, each node computes $\binom{K-1}{s-1}\eta_2=\frac{sQ}{K}$ Reduce functions, i.e., $|\mathcal{W}_k|=\frac{sQ}{K}$ for all $k\in\{1,\ldots,K\}$.

For a subset $\mathcal{S}$ of $\{1,\ldots,K\}$ and $\mathcal{S}_1\subset\mathcal{S}$ with $|\mathcal{S}_1|=r$, we denote the set of intermediate values needed by *all* nodes in $\mathcal{S}\backslash\mathcal{S}_1$, *no* node outside $\mathcal{S}$, and known *exclusively* by nodes in $\mathcal{S}_1$ as $\mathcal{V}_{\mathcal{S}_1}^{\mathcal{S}\backslash\mathcal{S}_1}$. More formally:

$$\begin{aligned}\mathcal{V}_{\mathcal{S}_1}^{\mathcal{S}\backslash\mathcal{S}_1}\triangleq\{v_{q,n}:&q\in\bigcap_{k\in\mathcal{S}\backslash\mathcal{S}_1}\mathcal{W}_k,q\notin\bigcup_{k\notin\mathcal{S}}\mathcal{W}_k,\\&w_n\in\bigcap_{k\in\mathcal{S}_1}\mathcal{M}_k,w_n\notin\bigcup_{k\notin\mathcal{S}_1}\mathcal{M}_k\}.\end{aligned}\tag{13}$$

We observe that the set $\mathcal{V}_{\mathcal{S}_1}^{\mathcal{S}\backslash\mathcal{S}_1}$ defined above contains intermediate values of $\binom{r}{|\mathcal{S}|-s}\eta_2$ output functions. This is because that the output functions whose intermediate values are included in $\mathcal{V}_{\mathcal{S}_1}^{\mathcal{S}\backslash\mathcal{S}_1}$ should be computed *exclusively* by the nodes in $\mathcal{S}\backslash\mathcal{S}_1$ and a subset of $s-(|\mathcal{S}|-r)$ nodes in $\mathcal{S}_1$. Therefore, $\mathcal{V}_{\mathcal{S}_1}^{\mathcal{S}\backslash\mathcal{S}_1}$ contains the intermediate values of a total of $\binom{r}{s-(|\mathcal{S}|-r)}\eta_2=\binom{r}{|\mathcal{S}|-s}\eta_2$ output functions. Since every subset of r nodes map a unique batch of η_1 files, $\mathcal{V}_{\mathcal{S}_1}^{\mathcal{S}\backslash\mathcal{S}_1}$ contains $|\mathcal{V}_{\mathcal{S}_1}^{\mathcal{S}\backslash\mathcal{S}_1}|=\binom{r}{|\mathcal{S}|-s}\eta_1\eta_2$ intermediate values.

Next, we first concatenate all intermediate values in $\mathcal{V}_{\mathcal{S}_1}^{\mathcal{S}\backslash\mathcal{S}_1}$ to construct a symbol $U_{\mathcal{S}_1}^{\mathcal{S}\backslash\mathcal{S}_1}\in\mathbb{F}_{2^{\binom{r}{|\mathcal{S}|-s}\eta_1\eta_2 T}}$. Then for $\mathcal{S}_1=\{\sigma_1,\ldots,\sigma_r\}$, we arbitrarily and evenly split $U_{\mathcal{S}_1}^{\mathcal{S}\backslash\mathcal{S}_1}$ into r segments, each containing $\binom{r}{|\mathcal{S}|-s}\frac{\eta_1\eta_2 T}{r}$ bits, i.e.,

$$U_{\mathcal{S}_1}^{\mathcal{S}\backslash\mathcal{S}_1}=\left(U_{\mathcal{S}_1,\sigma_1}^{\mathcal{S}\backslash\mathcal{S}_1},U_{\mathcal{S}_1,\sigma_2}^{\mathcal{S}\backslash\mathcal{S}_1},\ldots,U_{\mathcal{S}_1,\sigma_r}^{\mathcal{S}\backslash\mathcal{S}_1}\right),\tag{14}$$

where $U_{\mathcal{S}_1,\sigma_i}^{\mathcal{S}\backslash\mathcal{S}_1}\in\mathbb{F}_{2^{\binom{r}{|\mathcal{S}|-s}\frac{\eta_1\eta_2 T}{r}}}$ denotes the segment associated with Node $\sigma_i\in\mathcal{S}_1$.

For each $k\in\mathcal{S}$, there are a total of $\binom{|\mathcal{S}|-1}{r-1}$ subsets of $\mathcal{S}$ with size r that contain the element k. We index these subsets as $\mathcal{S}_{(k)}[1],\mathcal{S}_{(k)}[2]\ldots,\mathcal{S}_{(k)}[\binom{|\mathcal{S}|-1}{r-1}]$. Within a subset $\mathcal{S}_{(k)}[i]$, the segment associated with Node k is $U_{\mathcal{S}_{(k)}[i],k}^{\mathcal{S}\backslash\mathcal{S}_{(k)}[i]}$, for all $i=1,\ldots,\binom{|\mathcal{S}|-1}{r-1}$. We note that each segment $U_{\mathcal{S}_{(k)}[i],k}^{\mathcal{S}\backslash\mathcal{S}_{(k)}[i]}$, $i=1,\ldots,\binom{|\mathcal{S}|-1}{r-1}$, is known by all nodes whose indices are in $\mathcal{S}_{(k)}[i]$, and needed by all nodes whose indices are in $\mathcal{S}\backslash\mathcal{S}_{(k)}[i]$.

1) Encoding: The shuffling scheme of CDC consists of multiple rounds, each corresponding to all subsets of the K nodes with a particular size. Within each subset, each node multicasts *linear combinations* of the segments that are associated with it to the other nodes in the subset. More specifically, for each subset $\mathcal{S}\subseteq\{1,\ldots,K\}$ of size $\max\{r+1,s\}\leq|\mathcal{S}|\leq\min\{r+s,K\}$, we define $n_1\triangleq\binom{|\mathcal{S}|-1}{r-1}$ and $n_2\triangleq\binom{|\mathcal{S}|-2}{r-1}$. Then for each $k\in\mathcal{S}$, Node k computes n_2 message symbols, denoted by $X_k^{\mathcal{S}}[1],X_k^{\mathcal{S}}[2],\ldots,X_k^{\mathcal{S}}[n_2]$ as follows. For some coefficients $\alpha_1,\ldots,\alpha_{n_1}$ where $\alpha_i\in\mathbb{F}_{2^{\binom{r}{|\mathcal{S}|-s}\frac{\eta_1\eta_2 T}{r}}}$ for all $i=1,\ldots,n_1$, Node k computes

$$\begin{aligned}X_k^{\mathcal{S}}[1]&=U_{\mathcal{S}_{(k)}[1],k}^{\mathcal{S}\backslash\mathcal{S}_{(k)}[1]}+U_{\mathcal{S}_{(k)}[2],k}^{\mathcal{S}\backslash\mathcal{S}_{(k)}[2]}+\cdots+U_{\mathcal{S}_{(k)}[n_1],k}^{\mathcal{S}\backslash\mathcal{S}_{(k)}[n_1]},\\X_k^{\mathcal{S}}[2]&=\alpha_1U_{\mathcal{S}_{(k)}[1],k}^{\mathcal{S}\backslash\mathcal{S}_{(k)}[1]}+\alpha_2U_{\mathcal{S}_{(k)}[2],k}^{\mathcal{S}\backslash\mathcal{S}_{(k)}[2]}+\cdots+\alpha_{n_1}U_{\mathcal{S}_{(k)}[n_1],k}^{\mathcal{S}\backslash\mathcal{S}_{(k)}[n_1]},\\&\vdots\\X_k^{\mathcal{S}}[n_2]&=\alpha_1^{n_2-1}U_{\mathcal{S}_{(k)}[1],k}^{\mathcal{S}\backslash\mathcal{S}_{(k)}[1]}+\alpha_2^{n_2-1}U_{\mathcal{S}_{(k)}[2],k}^{\mathcal{S}\backslash\mathcal{S}_{(k)}[2]}\\&\quad+\cdots+\alpha_{n_1}^{n_2-1}U_{\mathcal{S}_{(k)}[n_1],k}^{\mathcal{S}\backslash\mathcal{S}_{(k)}[n_1]},\end{aligned}\tag{15}$$

or equivalently,

$$\begin{bmatrix}X_k^{\mathcal{S}}[1]\\X_k^{\mathcal{S}}[2]\\\vdots\\X_k^{\mathcal{S}}[n_2]\end{bmatrix}=\underbrace{\begin{bmatrix}1&1&\cdots&1\\\alpha_1&\alpha_2&\cdots&\alpha_{n_1}\\\vdots&\vdots&\ddots&\vdots\\\alpha_1^{n_2-1}&\alpha_2^{n_2-1}&\cdots&\alpha_{n_1}^{n_2-1}\end{bmatrix}}_{\mathbf{A}^{\mathcal{S}}}\begin{bmatrix}U_{\mathcal{S}_{(k)}[1],k}^{\mathcal{S}\backslash\mathcal{S}_{(k)}[1]}\\U_{\mathcal{S}_{(k)}[2],k}^{\mathcal{S}\backslash\mathcal{S}_{(k)}[2]}\\\vdots\\U_{\mathcal{S}_{(k)}[n_1],k}^{\mathcal{S}\backslash\mathcal{S}_{(k)}[n_1]}\end{bmatrix}.\tag{16}$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIT.2017.2756959, IEEE Transactions on Information Theory

We note that the above encoding process is the same at all nodes whose indices are in $\mathcal{S}$, i.e., each of them multiplies the same matrix $\mathbf{A}^{\mathcal{S}}$ in (16) with the segments associated with it.

Having generated the above message symbols, Node k multicasts them to the other nodes whose indices are in $\mathcal{S}$.

Remark 9. When $s = 1$, i.e., every output function is computed by one node, the above shuffling scheme only takes one round for all subsets $\mathcal{S}$ of size $|\mathcal{S}| = r+1$. Instead of multicasting linear combinations, every node in $\mathcal{S}$ can simply multicast the bit-wise XOR of its associated segments to the other r nodes in $\mathcal{S}$. □

2) Decoding: For $j \in \mathcal{S}$ and $j \neq k$, there are a total of $\binom{|\mathcal{S}|-2}{r-2}$ subsets of $\mathcal{S}$ that have size r and simultaneously contain j and k. Hence, among all n_1 segments $U^{\mathcal{S}\backslash\mathcal{S}_{(k)}[1]}_{\mathcal{S}_{(k)}[1],k}, U^{\mathcal{S}\backslash\mathcal{S}_{(k)}[2]}_{\mathcal{S}_{(k)}[2],k}, \ldots, U^{\mathcal{S}\backslash\mathcal{S}_{(k)}[n_1]}_{\mathcal{S}_{(k)}[n_1],k}$ associated with Node k, $\binom{|\mathcal{S}|-2}{r-2}$ of them are already known at Node j, and the rest of $n_1 - \binom{|\mathcal{S}|-2}{r-2} = \binom{|\mathcal{S}|-1}{r-1} - \binom{|\mathcal{S}|-2}{r-2} = \binom{|\mathcal{S}|-2}{r-1} = n_2$ segments are needed by Node j. We denote the indices of the subsets that contain the element k but not the element j as $b^1_{jk}, b^2_{jk}, \ldots, b^{n_2}_{jk}$, such that $1 \leq b^1_{jk} < b^2_{jk} < \cdots < b^{n_2}_{jk} \leq n_1$, and $j \notin \mathcal{S}_{(k)}[b^i_{jk}]$ for all $i = 1, 2, \ldots, n_2$.

After receiving the symbols $X^{\mathcal{S}}_k[1], X^{\mathcal{S}}_k[2], \ldots, X^{\mathcal{S}}_k[n_2]$ from Node k, Node j first removes the locally known segments from the linear combinations to generate n_2 symbols $Y^{\mathcal{S}}_{jk}[1], Y^{\mathcal{S}}_{jk}[2], \ldots, Y^{\mathcal{S}}_{jk}[n_2]$, such that

$$\begin{bmatrix} Y^{\mathcal{S}}_{jk}[1] \\ Y^{\mathcal{S}}_{jk}[2] \\ \vdots \\ Y^{\mathcal{S}}_{jk}[n_2] \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 1 & \cdots & 1 \\ \alpha_{b^1_{jk}} & \alpha_{b^2_{jk}} & \cdots & \alpha_{b^{n_2}_{jk}} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha^{n_2-1}_{b^1_{jk}} & \alpha^{n_2-1}_{b^2_{jk}} & \cdots & \alpha^{n_2-1}_{b^{n_2}_{jk}} \end{bmatrix}}_{\mathbf{B}^{\mathcal{S}}_{jk}} \begin{bmatrix} U^{\mathcal{S}\backslash\mathcal{S}_{(k)}[b^1_{jk}]}_{\mathcal{S}_{(k)}[b^1_{jk}],k} \\ U^{\mathcal{S}\backslash\mathcal{S}_{(k)}[b^2_{jk}]}_{\mathcal{S}_{(k)}[b^2_{jk}],k} \\ \vdots \\ U^{\mathcal{S}\backslash\mathcal{S}_{(k)}[b^{n_2}_{jk}]}_{\mathcal{S}_{(k)}[b^{n_2}_{jk}],k} \end{bmatrix}, \tag{17}$$

where $\mathbf{B}^{\mathcal{S}}_{jk} \in \mathbb{F}^{n_2 \times n_2}_{2^{\binom{r}{|\mathcal{S}|-s}\frac{\eta_1\eta_2 T}{r}}}$ is a square sub-matrix of $\mathbf{A}^{\mathcal{S}}$ in (16) that contains the columns with indices $b^1_{jk}, b^2_{jk}, \ldots, b^{n_2}_{jk}$ of $\mathbf{A}^{\mathcal{S}}_k$.

Node j can decode the desired segments from Node k if the matrix $\mathbf{B}^{\mathcal{S}}_{jk}$ is invertible. We note that $\mathbf{B}^{\mathcal{S}}_{jk}$ is a Vandermonde matrix, and it is invertible if $\alpha_{b^1_{jk}}, \alpha_{b^2_{jk}}, \ldots, \alpha_{b^{n_2}_{jk}}$ are all distinct. This holds for all $j \in \mathcal{S}\backslash\{k\}$ if there exist n_1 distinct coefficients in $\mathbb{F}_{2^{\binom{r}{|\mathcal{S}|-s}\frac{\eta_1\eta_2 T}{r}}}$, which requires $2^{\binom{r}{|\mathcal{S}|-s}\frac{\eta_1\eta_2 T}{r}} \geq n_1 = \binom{|\mathcal{S}|-1}{r-1}$, or equivalently $T \geq \frac{r\log\binom{|\mathcal{S}|-1}{r-1}}{\binom{r}{|\mathcal{S}|-s}\eta_1\eta_2}$. Finally, the proposed coded shuffling scheme can successfully deliver all the required intermediate values within all subsets $\mathcal{S}$ with $\max\{r+1, s\} \leq |\mathcal{S}| \leq \min\{r+s, K\}$, if T is sufficiently large, i.e.,

$$T \geq \max_{\max\{r+1,s\} \leq |\mathcal{S}| \leq \min\{r+s,K\}} \frac{r\log\binom{|\mathcal{S}|-1}{r-1}}{\binom{r}{|\mathcal{S}|-s}\eta_1\eta_2}. \tag{18}$$

C. Correctness of CDC

We demonstrate the correctness of the above shuffling scheme by showing that after the Shuffle phase, each node can decode all of the required intermediate values to compute its assigned Reduce functions. We use Node 1 as an example, and similar arguments apply to all other nodes. WLOG we assume that the Reduce function h_1 is to be computed by Node 1. Node 1 will need a total of $\binom{K-1}{r}\eta_1$ distinct intermediate values of h_1 from other nodes (it already knows $\frac{r\bar{N}}{K} = \bar{N} - \binom{K-1}{r}\eta_1$ intermediate values of h_1 by mapping the files in $\mathcal{M}_1$). By the assignment of the Reduce functions, there exits a subset $\mathcal{S}_2$ of size s containing Node 1 such that all nodes in $\mathcal{S}_2$ need to compute h_1. Then, during the data shuffling process within each subset $\mathcal{S}$ containing $\mathcal{S}_2$ (note that by the definition of $\mathcal{V}^{\mathcal{S}\backslash\mathcal{S}_1}_{\mathcal{S}_1}$ in (13), the intermediate values of h_1 will *not* be communicated to Node 1 if $\mathcal{S}_2 \nsubseteq \mathcal{S}$, and this is because that some node outside $\mathcal{S}$ also wants to compute h_1), there are $\binom{s-1}{|\mathcal{S}|-r-1}$ subsets $\mathcal{S}_1$ of $\mathcal{S}$ with size $|\mathcal{S}_1| = r$ such that $1 \notin \mathcal{S}_1$ and $\mathcal{S}\backslash\mathcal{S}_1 \subseteq \mathcal{S}_2$, and thus Node 1 decodes $\binom{s-1}{|\mathcal{S}|-r-1}\eta_1$ distinct intermediate values of h_1. Therefore, the total number of distinct intermediate values of h_1 Node 1 decodes over the entire Shuffle phase is

$$\sum_{\ell=\max\{r+1,s\}}^{\min\{r+s,K\}} \binom{s-1}{\ell-r-1}\binom{K-s}{\ell-s}\eta_1 = \binom{K-1}{r}\eta_1, \tag{19}$$

which matches the required number of intermediate values for h_1. This is also true for all the other Reduce functions assigned to Node 1.

D. Communication Load

In the above shuffling scheme, for each subset $\mathcal{S} \subseteq \{1, \ldots, K\}$ of size $\max\{r+1, s\} \leq |\mathcal{S}| \leq \min\{r+s, K\}$, each Node $k \in \mathcal{S}$ communicates $n_2 = \binom{|\mathcal{S}|-2}{r-1}$ message symbols. Each of these symbols contains $\binom{r}{|\mathcal{S}|-s}\frac{\eta_1\eta_2 T}{r}$ bits. Hence, all nodes whose indices are in $\mathcal{S}$ communicate a total of $|\mathcal{S}|\binom{|\mathcal{S}|-2}{r-1}\binom{r}{|\mathcal{S}|-s}\frac{\eta_1\eta_2 T}{r}$ bits. The overall communication load achieved by the proposed CDC scheme is

$$\begin{aligned} L_{\text{coded}}(r,s) &= \lim_{N\to\infty} \sum_{\ell=\max\{r+1,s\}}^{\min\{r+s,K\}} \frac{\binom{K}{\ell}\frac{\ell}{r}\binom{\ell-2}{r-1}\binom{r}{\ell-s}\eta_1\eta_2 T}{QNT} \\ &= \lim_{N\to\infty} \sum_{\ell=\max\{r+1,s\}}^{\min\{r+s,K\}} \frac{\ell\binom{K}{\ell}\binom{\ell-2}{r-1}\binom{r}{\ell-s}\bar{N}}{r\binom{K}{r}\binom{K}{s}N} \\ &= \sum_{\ell=\max\{r+1,s\}}^{\min\{r+s,K\}} \frac{\ell\binom{K}{\ell}\binom{\ell-2}{r-1}\binom{r}{\ell-s}}{r\binom{K}{r}\binom{K}{s}}. \end{aligned} \tag{20}$$

E. Non-Integer Valued Computation Load

For non-integer valued computation load $r \geq 1$, we generalize the CDC scheme as follows. We first expand the computation load $r = \alpha r_1 + (1-\alpha)r_2$ as a convex combination of $r_1 \triangleq \lfloor r \rfloor$ and $r_2 \triangleq \lceil r \rceil$, for some $0 \leq \alpha \leq 1$. Then we partition the set of $\bar{N}$ input files $\{w_1, \ldots, w_{\bar{N}}\}$ into two disjoint subsets $\mathcal{I}_1$ and $\mathcal{I}_2$ of sizes $|\mathcal{I}_1| = \alpha\bar{N}$ and $|\mathcal{I}_2| = (1-\alpha)\bar{N}$. We next apply the CDC scheme described above respectively to the files in $\mathcal{I}_1$ with a computation load r_1 and the files in $\mathcal{I}_2$ with a computation load r_2, to compute

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIT.2017.2756959, IEEE Transactions on Information Theory

each of the Q output functions at the same set of s nodes. This results in a communication load of

$$\lim_{N\to\infty} \frac{Q\alpha\bar{N}L_{\text{coded}}(r_1,s)T + Q(1-\alpha)\bar{N}L_{\text{coded}}(r_2,s)T}{QNT} = \alpha L_{\text{coded}}(r_1,s) + (1-\alpha)L_{\text{coded}}(r_2,s), \quad (21)$$

where $L_{\text{coded}}(r,s)$ is the communication load achieved by CDC in (20) for integer-valued $r,s \in \{1,\ldots,K\}$.

Using this generalized CDC scheme, for any two integer-valued computation loads r_1 and r_2, the points on the line segment connecting $(r_1, L_{\text{coded}}(r_1,s))$ and $(r_2, L_{\text{coded}}(r_2,s))$ are achievable. Therefore, for general $1 \le r \le K$, the *lower convex envelop* of the achievable points $\{(r, L_{\text{coded}}(r,s)) : r \in \{1,\ldots,K\}\}$ is achievable. This proves the upper bound on the computation-communication function in Theorem 2 (also the achievability part of Theorem 1 by setting $s=1$).

Remark 10. The ideas of efficiently creating and exploiting coded multicasting opportunities have been introduced in caching problems [20]–[22]. In this section, we illustrated how coding opportunities can be utilized in distributed computing to slash the load of communicating intermediate values, by designing a particular assignment of extra computations across distributed computing nodes. We note that the calculated intermediate values in the Map phase mimics the locally stored cache contents in caching problems, providing the "side information" to enable coding in the following Shuffle phase (or content delivery).

For the case of $s=1$ where no two nodes are interested in computing a common Reduce function, the coded data shuffling of CDC is similar to a coded transmission strategy in wireless D2D networks proposed in [22], where the side information enabling coded multicasting are pre-fetched in a specific repetitive manner in the caches of wireless nodes (in CDC such information is obtained by computing the Map functions locally). When s is larger than 1, i.e., every Reduce function needs to be computed at multiple nodes, our CDC scheme creates novel coding opportunities that exploit both the redundancy of the Map computations and the commonality of the data requests for Reduce functions across nodes, further reducing the communication load. □

Remark 11. Generally speaking, we can view the Shuffle phase of the considered distributed computing framework as an instance of the index coding problem [27], [28], in which a central server aims to design a broadcast message (code) with minimum length to simultaneously satisfy the requests of all the clients, given the clients' side information stored in their local caches. Note that while a randomized linear network coding approach (see e.g., [29]–[31]) is sufficient to implement any multicast communication where messages are intended by all receivers, it is generally sub-optimal for index coding problems where every client requests different messages. Although the index coding problem is still open in general, for the considered distributed computing scenario where we are given the flexibility of designing Map computation (thus the flexibility of designing side information), we prove in the next two sections *tight* lower bounds on the minimum communication loads for the cases $s=1$ and $s>1$ respectively, demonstrating the optimality of the proposed CDC scheme. □

VI. Converse of Theorem 1

In this section, we prove the lower bound on $L^*(r)$ in Theorem 1.

For $k \in \{1,\ldots,K\}$, we denote the set of indices of the files mapped by Node k as $\mathcal{M}_k$, and the set of indices of the Reduce functions computed by Node k as $\mathcal{W}_k$. As the first step, we consider the communication load for a given file assignment $\mathcal{M} \triangleq (\mathcal{M}_1, \mathcal{M}_2 \ldots, \mathcal{M}_K)$ in the Map phase. We denote the minimum communication load under the file assignment $\mathcal{M}$ by $L^*_{\mathcal{M}}$.

We denote the number of files that are mapped at j nodes under a file assignment $\mathcal{M}$, as $a^j_{\mathcal{M}}$, for all $j \in \{1,\ldots,K\}$:

$$a^j_{\mathcal{M}} = \sum_{\mathcal{J}\subseteq\{1,\ldots,K\}:|\mathcal{J}|=j} |(\bigcap_{k\in\mathcal{J}} \mathcal{M}_k)\backslash(\bigcup_{i\notin\mathcal{J}} \mathcal{M}_i)|. \quad (22)$$

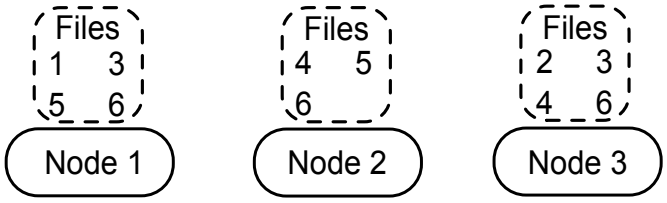


Fig. 6: A file assignment for $N=6$ files and $K=3$ nodes.

For example, for the particular file assignment in Fig. 6, i.e., $\mathcal{M} = (\{1,3,5,6\},\{4,5,6\},\{2,3,4,6\})$, $a^1_{\mathcal{M}} = 2$ since File 1 and File 2 are mapped on a single node (i.e., Node 1 and Node 3 respectively). Similarly, we have $a^2_{\mathcal{M}} = 3$ (Files 3, 4, and 5), and $a^3_{\mathcal{M}} = 1$ (File 6).

For a particular file assignment $\mathcal{M}$, we present a lower bound on $L^*_{\mathcal{M}}$ in the following lemma.

Lemma 1. $L^*_{\mathcal{M}} \ge \sum_{j=1}^{K} \frac{a^j_{\mathcal{M}}}{N} \cdot \frac{K-j}{Kj}$.

Next, we first demonstrate the converse of Theorem 1 using Lemma 1, and then give the proof of Lemma 1.

Converse Proof of Theorem 1. It is clear that the minimum communication load $L^*(r)$ is lower bounded by the minimum value of $L^*_{\mathcal{M}}$ over all possible file assignments which admit a computation load of r:

$$L^*(r) \ge \inf_{\mathcal{M}:|\mathcal{M}_1|+\cdots+|\mathcal{M}_K|=rN} L^*_{\mathcal{M}}. \quad (23)$$

Then by Lemma 1, we have

$$L^*(r) \ge \inf_{\mathcal{M}:|\mathcal{M}_1|+\cdots+|\mathcal{M}_K|=rN} \sum_{j=1}^{K} \frac{a^j_{\mathcal{M}}}{N} \cdot \frac{K-j}{Kj}. \quad (24)$$

For every file assignment $\mathcal{M}$ such that $|\mathcal{M}_1|+\cdots+|\mathcal{M}_K| = rN$, $\{a^j_{\mathcal{M}}\}_{j=1}^K$ satisfy

$$a^j_{\mathcal{M}} \ge 0,\ j \in \{1,...,K\}, \quad (25)$$

$$\sum_{j=1}^{K} a^j_{\mathcal{M}} = N, \quad (26)$$

$$\sum_{j=1}^{K} j a^j_{\mathcal{M}} = rN. \quad (27)$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIT.2017.2756959, IEEE Transactions on Information Theory

Then since the function $\frac{K-j}{Kj}$ in (24) is convex in j, and by (26) $\sum_{j=1}^{K} \frac{a_{\mathcal{M}}^j}{N} = 1$, (24) becomes

$$L^*(r) \geq \inf_{\mathcal{M}:|\mathcal{M}_1|+\cdots+|\mathcal{M}_K|=rN} \frac{K - \sum_{j=1}^{K} j \frac{a_{\mathcal{M}}^j}{N}}{K \sum_{j=1}^{K} j \frac{a_{\mathcal{M}}^j}{N}} \overset{(a)}{=} \frac{K-r}{Kr}, \tag{28}$$

where (a) is due to the requirement imposed by the computation load in (27).

The lower bound on $L^*(r)$ in (28) holds for general $1 \leq r \leq K$. We can further improve the lower bound for non-integer valued r as follows. For a particular $r \notin \mathbb{N}$, we first find the line $p+qj$ as a function of $1 \leq j \leq K$ connecting the two points $(\lfloor r \rfloor, \frac{K-\lfloor r \rfloor}{K\lfloor r \rfloor})$ and $(\lceil r \rceil, \frac{K-\lceil r \rceil}{K\lceil r \rceil})$. More specifically, we find $p, q \in \mathbb{R}$ such that

$$p + qj|_{j=\lfloor r \rfloor} = \frac{K - \lfloor r \rfloor}{K \lfloor r \rfloor}, \tag{29}$$

$$p + qj|_{j=\lceil r \rceil} = \frac{K - \lceil r \rceil}{K \lceil r \rceil}. \tag{30}$$

Then by the convexity of the function $\frac{K-j}{Kj}$ in j, we have for integer-valued $j = 1, \ldots, K$,

$$\frac{K-j}{Kj} \geq p + qj, \quad j = 1, \ldots, K. \tag{31}$$

Then (24) reduces to

$$L^*(r) \geq \inf_{\mathcal{M}:|\mathcal{M}_1|+\cdots+|\mathcal{M}_K|=rN} \sum_{j=1}^{K} \frac{a_{\mathcal{M}}^j}{N} \cdot (p + qj) \tag{32}$$

$$= \inf_{\mathcal{M}:|\mathcal{M}_1|+\cdots+|\mathcal{M}_K|=rN} \sum_{j=1}^{K} \frac{a_{\mathcal{M}}^j}{N} \cdot p + \sum_{j=1}^{K} \frac{j a_{\mathcal{M}}^j}{N} \cdot q \tag{33}$$

$$\overset{(b)}{=} p + qr, \tag{34}$$

where (b) is due to the constraints on $\{a_{\mathcal{M}}^j\}_{j=1}^K$ in (26) and (27).

Therefore, $L^*(r)$ is lower bounded by the lower convex envelop of the points $\{(r, \frac{K-r}{Kr}) : r \in \{1, ..., K\}\}$. This completes the proof of the converse part of Theorem 1. ■

Remark 12. Although the model proposed in this paper only allows each node sending messages independently, we can show that even if the data shuffling process can be carried out in multiple rounds and dependency between messages are allowed, the lower bound on $L^*(r)$ remains the same. □

We devote the rest of this section to the proof of Lemma 1. To prove Lemma 1, we develop a lower bound on the number of bits communicated by any subset of nodes, by induction on the size of the subset. In particular, for a subset of computing nodes, we first characterize a lower bound on the minimum number of bits required by a particular node in the subset, which is given by a cut-set bound separating this node and all the other nodes in the subset. Then, we combine this bound with the lower bound on the number of bits communicated by the rest of the nodes in the subset, which is given by the inductive argument.

Proof of Lemma 1. For $q \in \{1, ..., Q\}$, $n \in \{1, ..., N\}$, we let $V_{q,n}$ be i.i.d. random variables uniformly distributed on $\mathbb{F}_{2^T}$. We let the intermediate values $v_{q,n}$ be the realizations of $V_{q,n}$. For some $\mathcal{Q} \subseteq \{1, \ldots, Q\}$ and $\mathcal{N} \subseteq \{1, \ldots, N\}$, we define

$$V_{\mathcal{Q},\mathcal{N}} \triangleq \{V_{q,n} : q \in \mathcal{Q}, n \in \mathcal{N}\}. \tag{35}$$

Since each message X_k is generated as a function of the intermediate values that are computed at Node k, the following equation holds for all $k \in \{1, ..., K\}$.

$$H(X_k | V_{:,\mathcal{M}_k}) = 0, \tag{36}$$

where we use ":" to denote the set of all possible indices.

The validity of the shuffling scheme requires that for all $k \in \{1, ..., K\}$, the following equation holds :

$$H(V_{\mathcal{W}_k,:} | X_:, V_{:,\mathcal{M}_k}) = 0. \tag{37}$$

For a subset $\mathcal{S} \subseteq \{1, ..., K\}$, we define

$$Y_{\mathcal{S}} \triangleq (V_{\mathcal{W}_{\mathcal{S}},:}, V_{:,\mathcal{M}_{\mathcal{S}}}), \tag{38}$$

which contains all the intermediate values required by the nodes in $\mathcal{S}$ and all the intermediate values known locally by the nodes in $\mathcal{S}$ after the Map phase.

For any subset $\mathcal{S} \subseteq \{1, \ldots, K\}$ and a file assignment $\mathcal{M}$, we denote the number of files that are *exclusively* mapped by j nodes in $\mathcal{S}$ as $a_{\mathcal{M}}^{j,\mathcal{S}}$:

$$a_{\mathcal{M}}^{j,\mathcal{S}} \triangleq \sum_{\mathcal{J} \subseteq \mathcal{S}:|\mathcal{J}|=j} |(\bigcap_{k \in \mathcal{J}} \mathcal{M}_k) \backslash (\bigcup_{i \notin \mathcal{J}} \mathcal{M}_i)|, \tag{39}$$

and the message symbols communicated by the nodes whose indices are in $\mathcal{S}$ as

$$X_{\mathcal{S}} = \{X_k : k \in \mathcal{S}\}. \tag{40}$$

Then we prove the following claim.

Claim 1. For any subset $\mathcal{S} \subseteq \{1, ..., K\}$, we have

$$H(X_{\mathcal{S}} | Y_{\mathcal{S}^c}) \geq T \sum_{j=1}^{|\mathcal{S}|} a_{\mathcal{M}}^{j,\mathcal{S}} \frac{Q}{K} \cdot \frac{|\mathcal{S}| - j}{j}, \tag{41}$$

where $\mathcal{S}^c \triangleq \{1, \ldots, K\} \backslash \mathcal{S}$ denotes the complement of $\mathcal{S}$. □

We prove Claim 1 by induction.

a. If $\mathcal{S} = \{k\}$ for any $k \in \{1, \ldots, K\}$, obviously

$$H(X_k | Y_{\{1,...,K\}\backslash\{k\}}) \geq 0 = T a_{\mathcal{M}}^{1,\{k\}} \frac{Q}{K} \cdot \frac{1-1}{1}. \tag{42}$$

b. Suppose the statement is true for all subsets of size S_0.

For any $\mathcal{S} \subseteq \{1, ..., K\}$ of size $|\mathcal{S}| = S_0 + 1$ and any $k \in \mathcal{S}$, we have

$$H(X_{\mathcal{S}} | Y_{\mathcal{S}^c})$$

$$= \frac{1}{|\mathcal{S}|} \sum_{k \in \mathcal{S}} H(X_{\mathcal{S}}, X_k | Y_{\mathcal{S}^c}) \tag{43}$$

$$= \frac{1}{|\mathcal{S}|} \sum_{k \in \mathcal{S}} (H(X_{\mathcal{S}} | X_k, Y_{\mathcal{S}^c}) + H(X_k | Y_{\mathcal{S}^c})) \tag{44}$$

$$\geq \frac{1}{|\mathcal{S}|} \sum_{k \in \mathcal{S}} H(X_{\mathcal{S}} | X_k, Y_{\mathcal{S}^c}) + \frac{1}{|\mathcal{S}|} H(X_{\mathcal{S}} | Y_{\mathcal{S}^c}). \tag{45}$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIT.2017.2756959, IEEE Transactions on Information Theory

From (45), we have

$$H(X_{\mathcal{S}}|Y_{\mathcal{S}^c}) \geq \frac{1}{|\mathcal{S}|-1}\sum_{k\in\mathcal{S}} H(X_{\mathcal{S}}|X_k, Y_{\mathcal{S}^c}) \tag{46}$$

$$\geq \frac{1}{S_0}\sum_{k\in\mathcal{S}} H(X_{\mathcal{S}}|X_k, V_{:,\mathcal{M}_k}, Y_{\mathcal{S}^c}) \tag{47}$$

$$= \frac{1}{S_0}\sum_{k\in\mathcal{S}} H(X_{\mathcal{S}}|V_{:,\mathcal{M}_k}, Y_{\mathcal{S}^c}). \tag{48}$$

For each $k \in \mathcal{S}$, we have the following subset version of (36) and (37).

$$H(X_k|V_{:,\mathcal{M}_k}, Y_{\mathcal{S}^c}) = 0, \tag{49}$$

$$H(V_{\mathcal{W}_k,:}|X_{\mathcal{S}}, V_{:,\mathcal{M}_k}, Y_{\mathcal{S}^c}) = 0. \tag{50}$$

Consequently,

$$H(X_{\mathcal{S}}, V_{\mathcal{W}_k,:}|V_{:,\mathcal{M}_k}, Y_{\mathcal{S}^c}) = H(X_{\mathcal{S}}|V_{:,\mathcal{M}_k}, Y_{\mathcal{S}^c}) \tag{51}$$

$$= H(V_{\mathcal{W}_k,:}|V_{:,\mathcal{M}_k}, Y_{\mathcal{S}^c}) + H(X_{\mathcal{S}}|V_{\mathcal{W}_k,:}, V_{:,\mathcal{M}_k}, Y_{\mathcal{S}^c}). \tag{52}$$

The first term on the RHS of (52) can be lower bounded as follows.

$$H(V_{\mathcal{W}_k,:}|V_{:,\mathcal{M}_k}, Y_{\mathcal{S}^c}) = H(V_{\mathcal{W}_k,:}|V_{:,\mathcal{M}_k}, V_{\mathcal{W}_{\mathcal{S}^c},:}, V_{:,\mathcal{M}_{\mathcal{S}^c}})$$

$$\overset{(a)}{=} H(V_{\mathcal{W}_k,:}|V_{:,\mathcal{M}_k}, V_{:,\mathcal{M}_{\mathcal{S}^c}}) \tag{53}$$

$$\overset{(b)}{=} H(V_{\mathcal{W}_k,:}|V_{\mathcal{W}_k,\mathcal{M}_k\cup\mathcal{M}_{\mathcal{S}^c}}) \tag{54}$$

$$\overset{(c)}{=} \sum_{q\in\mathcal{W}_k} H(V_{\{q\},:}|V_{\{q\},\mathcal{M}_k\cup\mathcal{M}_{\mathcal{S}^c}}) \tag{55}$$

$$\overset{(d)}{=} \frac{Q}{K}T\sum_{j=0}^{S_0} a_{\mathcal{M}}^{j,\mathcal{S}\backslash\{k\}} \tag{56}$$

$$\geq \frac{Q}{K}T\sum_{j=1}^{S_0} a_{\mathcal{M}}^{j,\mathcal{S}\backslash\{k\}}, \tag{57}$$

where (a) is due to the independence of intermediate values and the fact that $\mathcal{W}_k \cap \mathcal{W}_{\mathcal{S}^c} = \varnothing$ (different nodes calculate different output functions), (b) and (c) are due to the independence of intermediate values, and (d) is due to the independence of the intermediate values and the fact that $|\mathcal{W}_k| = \frac{Q}{K}$.

The second term on the RHS of (52) can be lower bounded by the induction assumption:

$$H(X_{\mathcal{S}}|V_{\mathcal{W}_k,:}, V_{:,\mathcal{M}_k}, Y_{\mathcal{S}^c}) = H(X_{\mathcal{S}\backslash\{k\}}|Y_{(\mathcal{S}\backslash\{k\})^c}) \tag{58}$$

$$\geq T\sum_{j=1}^{S_0} a_{\mathcal{M}}^{j,\mathcal{S}\backslash\{k\}}\frac{Q}{K}\cdot\frac{S_0-j}{j}. \tag{59}$$

Thus by (48), (52), (57) and (59), we have

$$H(X_{\mathcal{S}}|Y_{\mathcal{S}^c}) \geq \frac{1}{S_0}\sum_{k\in\mathcal{S}} H(X_{\mathcal{S}}|V_{:,\mathcal{M}_k}, Y_{\mathcal{S}^c}) \tag{60}$$

$$= \frac{1}{S_0}\sum_{k\in\mathcal{S}}\Big(H(V_{\mathcal{W}_k,:}|V_{:,\mathcal{M}_k}, Y_{\mathcal{S}^c}) + H(X_{\mathcal{S}}|V_{\mathcal{W}_k,:}, V_{:,\mathcal{M}_k}, Y_{\mathcal{S}^c})\Big) \tag{61}$$

$$\geq \frac{1}{S_0}\sum_{k\in\mathcal{S}}\Big(T\sum_{j=1}^{S_0} a_{\mathcal{M}}^{j,\mathcal{S}\backslash\{k\}}\frac{Q}{K} + T\sum_{j=1}^{S_0} a_{\mathcal{M}}^{j,\mathcal{S}\backslash\{k\}}\frac{Q}{K}\cdot\frac{S_0-j}{j}\Big) \tag{62}$$

$$= \frac{T}{S_0}\sum_{k\in\mathcal{S}}\sum_{j=1}^{S_0} a_{\mathcal{M}}^{j,\mathcal{S}\backslash\{k\}}\frac{Q}{K}\cdot\frac{S_0}{j} \tag{63}$$

$$= T\sum_{j=1}^{S_0}\frac{Q}{K}\cdot\frac{1}{j}\sum_{k\in\mathcal{S}} a_{\mathcal{M}}^{j,\mathcal{S}\backslash\{k\}}. \tag{64}$$

By the definition of $a_{\mathcal{M}}^{j,\mathcal{S}}$, we have the following equations.

$$\sum_{k\in\mathcal{S}} a_{\mathcal{M}}^{j,\mathcal{S}\backslash\{k\}}$$

$$= \sum_{k\in\mathcal{S}}\sum_{n=1}^{N}\mathbb{1}(\text{file } n \text{ is only mapped by some nodes in } \mathcal{S}\backslash\{k\}) \times \mathbb{1}(\text{file } n \text{ is mapped by } j \text{ nodes}) \tag{65}$$

$$= \sum_{n=1}^{N}\mathbb{1}(\text{file } n \text{ is only mapped by } j \text{ nodes in } \mathcal{S}) \times \sum_{k\in\mathcal{S}}\mathbb{1}(\text{file } n \text{ is not mapped by Node } k) \tag{66}$$

$$= \sum_{n=1}^{N}\mathbb{1}(\text{file } n \text{ is only mapped by } j \text{ nodes in } \mathcal{S})(|\mathcal{S}|-j) \tag{67}$$

$$= a_{\mathcal{M}}^{j,\mathcal{S}}(S_0+1-j). \tag{68}$$

Applying (68) to (64) yields

$$H(X_{\mathcal{S}}|Y_{\mathcal{S}^c}) \geq T\sum_{j=1}^{S_0} a_{\mathcal{M}}^{j,\mathcal{S}}\frac{Q}{K}\cdot\frac{S_0+1-j}{j} \tag{69}$$

$$= T\sum_{j=1}^{S_0+1} a_{\mathcal{M}}^{j,\mathcal{S}}\frac{Q}{K}\cdot\frac{S_0+1-j}{j}. \tag{70}$$

c. Thus for all subsets $\mathcal{S} \subseteq \{1,...,K\}$, the following equation holds:

$$H(X_{\mathcal{S}}|Y_{\mathcal{S}^c}) \geq T\sum_{j=1}^{|\mathcal{S}|} a_{\mathcal{M}}^{j,\mathcal{S}}\frac{Q}{K}\cdot\frac{|\mathcal{S}|-j}{j}, \tag{71}$$

which proves Claim 1.

Then by Claim 1, let $\mathcal{S} = \{1,...,K\}$ be the set of all K nodes,

$$L_{\mathcal{M}}^* \geq \frac{H(X_{\mathcal{S}}|Y_{\mathcal{S}^c})}{QNT} \geq \sum_{j=1}^{K}\frac{a_{\mathcal{M}}^{j}}{N}\cdot\frac{K-j}{Kj}. \tag{72}$$

This completes the proof of Lemma 1. ■

VII. Converse of Theorem 2

In this section, we prove the lower bound on $L^*(r,s)$ in Theorem 2, which generalizes the converse result of Theorem 1 for the case $s > 1$. Since the lower bound on $L^*(r,1)$ in Theorem 2 exactly matches the lower bound on $L^*(r)$ in

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIT.2017.2756959, IEEE Transactions on Information Theory

Theorem 1, we focus on the case $s > 1$ (i.e., each Reduce function is calculated by 2 or more nodes) throughout this section.

We denote the minimum communication load under a particular file assignment $\mathcal{M}$ as $L^*_{\mathcal{M}}(s)$, and we present a lower bound on $L^*_{\mathcal{M}}(s)$ in the following lemma.

Lemma 2. $L^*_{\mathcal{M}}(s) \geq \sum_{j=1}^{K} \frac{a^j_{\mathcal{M}}}{N} \sum_{\ell=\max\{j,s\}}^{\min\{j+s,K\}} \frac{\binom{K-j}{\ell-j}\binom{j}{\ell-s}}{\binom{K}{s}} \cdot \frac{\ell-j}{\ell-1}$, *where $a^j_{\mathcal{M}}$ is defined in (22).*

In the rest of this section, we first prove the converse part of Theorem 2 by showing $L^*(r,s) \geq \sum_{\ell=\max\{r,s\}}^{\min\{r+s,K\}} \frac{\binom{K-r}{\ell-r}\binom{r}{\ell-s}}{\binom{K}{s}} \cdot \frac{\ell-r}{\ell-1}$, and then give the proof of Lemma 2.

Converse Proof of Theorem 2. The minimum communication load $L^*(r,s)$ is lower bounded by the minimum value of $L^*_{\mathcal{M}}(s)$ over all possible file assignments having a computation load of r:

$$L^*(r,s) \geq \inf_{\mathcal{M}:|\mathcal{M}_1|+\cdots+|\mathcal{M}_K|=rN} L^*_{\mathcal{M}}(s). \tag{73}$$

For every file assignment $\mathcal{M}$ such that $|\mathcal{M}_1|+\cdots+|\mathcal{M}_K| = rN$, $\{a^j_{\mathcal{M}}\}_{j=1}^K$ satisfy the same conditions as the case of $s=1$ in (25), (26) and (27).

For a general computation load $1 \leq r \leq K$, and the function $L_{\text{coded}}(r,s) = \sum_{\ell=\max\{r+1,s\}}^{\min\{r+s,K\}} \frac{\ell\binom{K}{\ell}\binom{\ell-2}{r-1}\binom{r}{\ell-s}}{r\binom{K}{r}\binom{K}{s}} = \sum_{\ell=\max\{r,s\}}^{\min\{r+s,K\}} \frac{\binom{K-r}{\ell-r}\binom{r}{\ell-s}}{\binom{K}{s}} \cdot \frac{\ell-r}{\ell-1}$ as defined in (20), we first find the line $p+qj$ as a function of $1 \leq j \leq K$ connecting the two points $(\lfloor r \rfloor, L_{\text{coded}}(\lfloor r \rfloor, s))$ and $(\lceil r \rceil, L_{\text{coded}}(\lceil r \rceil, s))$. More specifically, we find $p, q \in \mathbb{R}$ such that

$$p + qj|_{j=\lfloor r \rfloor} = L_{\text{coded}}(\lfloor r \rfloor, s), \tag{74}$$

$$p + qj|_{j=\lceil r \rceil} = L_{\text{coded}}(\lceil r \rceil, s). \tag{75}$$

Then by the convexity of the function $L_{\text{coded}}(j,s)$ in j, we have for integer-valued $j = 1, \ldots, K$,

$$L_{\text{coded}}(j,s) = \sum_{\ell=\max\{j,s\}}^{\min\{j+s,K\}} \frac{\binom{K-j}{\ell-j}\binom{j}{\ell-s}}{\binom{K}{s}} \cdot \frac{\ell-j}{\ell-1} \geq p + qj. \tag{76}$$

Next, we first apply Lemma 2 to (73), then by (76), we have

$$L^*(r,s) \geq \inf_{\mathcal{M}:|\mathcal{M}_1|+\cdots+|\mathcal{M}_K|=rN} \sum_{j=1}^{K} \frac{a^j_{\mathcal{M}}}{N} \cdot (p+qj) \tag{77}$$

$$= \inf_{\mathcal{M}:|\mathcal{M}_1|+\cdots+|\mathcal{M}_K|=rN} \sum_{j=1}^{K} \frac{a^j_{\mathcal{M}}}{N} \cdot p + \sum_{j=1}^{K} \frac{j a^j_{\mathcal{M}}}{N} \cdot q \tag{78}$$

$$\overset{(a)}{=} p + qr, \tag{79}$$

where (a) is due to the constraints on $\{a^j_{\mathcal{M}}\}_{j=1}^K$ in (26) and (27).

Therefore, $L^*(r,s)$ is lower bounded by the lower convex envelop of the points $\{(r, L_{\text{coded}}(r,s)) : r \in \{1, ..., K\}\}$. This completes the proof of the converse part of Theorem 2. ■

The proof of lemma 2 follows the same steps of the proof of Lemma 1, where a lower bound on the number of bits communicated by any subset of nodes, for the case of $s > 1$, is established by induction.

Proof of Lemma 2. We first prove the following claim.

Claim 2. For any subset $\mathcal{S} \subseteq \{1, ..., K\}$, we have

$$H(X_{\mathcal{S}}|Y_{\mathcal{S}^c}) \geq QT \sum_{j=1}^{|\mathcal{S}|} a^{j,\mathcal{S}}_{\mathcal{M}} \sum_{\ell=\max\{j,s\}}^{\min\{j+s,|\mathcal{S}|\}} \frac{\binom{|\mathcal{S}|-j}{\ell-j}\binom{j}{\ell-s}}{\binom{K}{s}} \cdot \frac{\ell-j}{\ell-1}, \tag{80}$$

where $a^{j,\mathcal{S}}_{\mathcal{M}}$ is defined in (39). □

We prove Claim 2 by induction.

a. If $\mathcal{S} = \{k\}$ for any $k \in \{1, \ldots, K\}$, obviously

$$H(X_k|Y_{\{1,\ldots,K\}\setminus\{k\}}) \geq 0 = QT a^{1,\{k\}}_{\mathcal{M}} \sum_{\ell=s}^{1} \frac{\binom{0}{\ell-1}\binom{1}{\ell-s}}{\binom{K}{s}}. \tag{81}$$

b. Suppose the statement is true for all subsets of size S_0. For any $\mathcal{S} \subseteq \{1, ..., K\}$ of size $|\mathcal{S}| = S_0 + 1$, and all $k \in \mathcal{S}$, we have as derived in (61):

$$H(X_{\mathcal{S}}|Y_{\mathcal{S}^c}) \geq \frac{1}{S_0} \sum_{k\in\mathcal{S}} \Big(H(X_{\mathcal{S}}|V_{\mathcal{W}_k,:}, V_{:,\mathcal{M}_k}, Y_{\mathcal{S}^c}) + H(V_{\mathcal{W}_k,:}|V_{:,\mathcal{M}_k}, Y_{\mathcal{S}^c}) \Big), \tag{82}$$

where $Y_{\mathcal{S}^c} = (V_{\mathcal{W}_{\mathcal{S}^c},:}, V_{:,\mathcal{M}_{\mathcal{S}^c}})$.

The first term on the RHS of (82) is lower bounded by the induction assumption:

$$H(X_{\mathcal{S}}|V_{\mathcal{W}_k,:}, V_{:,\mathcal{M}_k}, Y_{\mathcal{S}^c}) = H(X_{\mathcal{S}\setminus\{k\}}|Y_{(\mathcal{S}\setminus\{k\})^c}) \tag{83}$$

$$\geq QT \sum_{j=1}^{S_0} a^{j,\mathcal{S}\setminus\{k\}}_{\mathcal{M}} \sum_{\ell=\max\{j,s\}}^{\min\{j+s,S_0\}} \frac{\binom{S_0-j}{\ell-j}\binom{j}{\ell-s}}{\binom{K}{s}} \cdot \frac{\ell-j}{\ell-1}. \tag{84}$$

The second term on the RHS of (82) can be calculated based on the independence of intermediate values:

$$H(V_{\mathcal{W}_k,:}|V_{:,\mathcal{M}_k}, Y_{\mathcal{S}^c})$$

$$= H(V_{\mathcal{W}_k,:}|V_{:,\mathcal{M}_k}, V_{\mathcal{W}_{\mathcal{S}^c},:}, V_{:,\mathcal{M}_{\mathcal{S}^c}}) \tag{85}$$

$$\overset{(a)}{=} H(V_{\mathcal{W}_k,:}|V_{\mathcal{W}_k,\mathcal{M}_k\cup\mathcal{M}_{\mathcal{S}^c}}, V_{\mathcal{W}_{\mathcal{S}^c},:}) \tag{86}$$

$$\overset{(b)}{=} \sum_{q\in\mathcal{W}_k} H(V_{\{q\},:}|V_{\{q\},\mathcal{M}_k\cup\mathcal{M}_{\mathcal{S}^c}}, V_{\mathcal{W}_{\mathcal{S}^c},:}) \tag{87}$$

$$\overset{(c)}{=} \frac{Q}{\binom{K}{s}} \binom{|\mathcal{S}|-1}{s-1} T \sum_{j=0}^{S_0} a^{j,\mathcal{S}\setminus\{k\}}_{\mathcal{M}} \tag{88}$$

$$\geq \frac{Q}{\binom{K}{s}} \binom{|\mathcal{S}|-1}{s-1} T \sum_{j=1}^{S_0} a^{j,\mathcal{S}\setminus\{k\}}_{\mathcal{M}}, \tag{89}$$

where (a) and (b) are due to the independence of the intermediate values, and (c) is due to the uniform distribution of the output functions such that each node in $\mathcal{S}$ calculates $\frac{Q}{\binom{K}{s}} \cdot \binom{|\mathcal{S}|-1}{s-1}$ output functions computed exclusively by s nodes in $\mathcal{S}$.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIT.2017.2756959, IEEE Transactions on Information Theory

Thus by (82), (84), and (89), we have

$$\begin{aligned}
&H(X_{\mathcal{S}}|Y_{\mathcal{S}^c})\\
\geq&\frac{QT}{S_0}\sum_{k\in\mathcal{S}}\sum_{j=1}^{S_0}a_{\mathcal{M}}^{j,\mathcal{S}\backslash\{k\}}\Bigg(\sum_{\ell=\max\{j,s\}}^{\min\{j+s,S_0\}}\frac{\binom{S_0-j}{\ell-j}\binom{j}{\ell-s}}{\binom{K}{s}}\cdot\frac{\ell-j}{\ell-1}\\
&+\frac{\binom{S_0}{s-1}}{\binom{K}{s}}\Bigg) &(90)\\
=&\frac{QT}{S_0}\sum_{j=1}^{S_0}\Bigg(\sum_{\ell=\max\{j,s\}}^{\min\{j+s,S_0\}}\frac{\binom{S_0-j}{\ell-j}\binom{j}{\ell-s}}{\binom{K}{s}}\cdot\frac{\ell-j}{\ell-1}+\frac{\binom{S_0}{s-1}}{\binom{K}{s}}\Bigg)\\
&\cdot\sum_{k\in\mathcal{S}}a_{\mathcal{M}}^{j,\mathcal{S}\backslash\{k\}} &(91)\\
=&QT\cdot\frac{S_0+1-j}{S_0}\sum_{j=1}^{S_0}\Bigg(\sum_{\ell=\max\{j,s\}}^{\min\{j+s,S_0\}}\frac{\binom{S_0-j}{\ell-j}\binom{j}{\ell-s}}{\binom{K}{s}}\cdot\frac{\ell-j}{\ell-1}\\
&+\frac{\binom{S_0}{s-1}}{\binom{K}{s}}\Bigg)a_{\mathcal{M}}^{j,\mathcal{S}} &(92)\\
=&QT\sum_{j=1}^{S_0+1}\frac{S_0+1-j}{S_0}\Bigg(\sum_{\ell=\max\{j,s\}}^{\min\{j+s,S_0\}}\frac{\binom{S_0-j}{\ell-j}\binom{j}{\ell-s}}{\binom{K}{s}}\cdot\frac{\ell-j}{\ell-1}\\
&+\frac{\binom{S_0}{s-1}}{\binom{K}{s}}\Bigg)a_{\mathcal{M}}^{j,\mathcal{S}}. &(93)
\end{aligned}$$

For each $j\in\{1,\ldots,S_0+1\}$ in (93), we have

$$\begin{aligned}
&\frac{S_0+1-j}{S_0}\Bigg(\sum_{\ell=\max\{j,s\}}^{\min\{j+s,S_0\}}\frac{\binom{S_0-j}{\ell-j}\binom{j}{\ell-s}}{\binom{K}{s}}\cdot\frac{\ell-j}{\ell-1}+\frac{\binom{S_0}{s-1}}{\binom{K}{s}}\Bigg)\\
=&\frac{S_0+1-j}{S_0\binom{K}{s}}\Bigg(\sum_{\ell=\max\{j,s\}}^{\min\{j+s,S_0\}}\binom{S_0-j}{\ell-j}\binom{j}{\ell-s}\frac{\ell-j}{\ell-1}\\
&+\sum_{\ell=\max\{j+1,s\}}^{\min\{j+s,S_0+1\}}\binom{S_0-j}{\ell-j-1}\binom{j}{\ell-s}\Bigg) &(94)\\
=&\frac{S_0+1-j}{S_0\binom{K}{s}}\Bigg(\sum_{\ell=\max\{j,s\}}^{\min\{j+s,S_0+1\}}\binom{S_0-j}{\ell-j}\binom{j}{\ell-s}\frac{\ell-j}{\ell-1}\\
&+\sum_{\ell=\max\{j,s\}}^{\min\{j+s,S_0+1\}}\binom{S_0-j}{\ell-j-1}\binom{j}{\ell-s}\Bigg) &(95)\\
=&\frac{1}{\binom{K}{s}}\sum_{\ell=\max\{j,s\}}^{\min\{j+s,S_0+1\}}\binom{S_0+1-j}{\ell-j}\binom{j}{\ell-s}\\
&\left(\frac{S_0-\ell+1}{S_0}\cdot\frac{\ell-j}{\ell-1}+\frac{\ell-j}{S_0}\right) &(96)\\
=&\frac{1}{\binom{K}{s}}\sum_{\ell=\max\{j,s\}}^{\min\{j+s,S_0+1\}}\binom{S_0+1-j}{\ell-j}\binom{j}{\ell-s}\frac{\ell-j}{\ell-1}. &(97)
\end{aligned}$$

Applying (97) into (93) yields

$$\begin{aligned}
&H(X_{\mathcal{S}}|Y_{\mathcal{S}^c})\\
\geq&QT\sum_{j=1}^{S_0+1}a_{\mathcal{M}}^{j,\mathcal{S}}\sum_{\ell=\max\{j,s\}}^{\min\{j+s,S_0+1\}}\frac{\binom{S_0+1-j}{\ell-j}\binom{j}{\ell-s}}{\binom{K}{s}}\cdot\frac{\ell-j}{\ell-1} &(98)\\
=&QT\sum_{j=1}^{|\mathcal{S}|}a_{\mathcal{M}}^{j,\mathcal{S}}\sum_{\ell=\max\{j+s,|\mathcal{S}|\}}^{\min\{j+s,|\mathcal{S}|\}}\frac{\binom{|\mathcal{S}|-j}{\ell-j}\binom{j}{\ell-s}}{\binom{K}{s}}\cdot\frac{\ell-j}{\ell-1}. &(99)
\end{aligned}$$

Since (99) holds for all subsets $\mathcal{S}$ of size $|\mathcal{S}|=S_0+1$, we have proven Claim 2.

Then by Claim 2, let $\mathcal{S}=\{1,...,K\}$ be the set of all K nodes,

$$\begin{aligned}
L_{\mathcal{M}}^*(s)\geq&\frac{H(X_{\mathcal{S}}|Y_{\mathcal{S}^c})}{QNT}\\
\geq&\sum_{j=1}^{K}\frac{a_{\mathcal{M}}^{j}}{N}\sum_{\ell=\max\{j,s\}}^{\min\{j+s,K\}}\frac{\binom{K-j}{\ell-j}\binom{j}{\ell-s}}{\binom{K}{s}}\cdot\frac{\ell-j}{\ell-1}. &(100)
\end{aligned}$$

This completes the proof of Lemma 2. ■

VIII. Implementation and Empirical Evaluation of Coded Distributed Computing

In this section, we demonstrate the impact of the proposed Coded Distributed Computing (CDC) scheme on balancing the time spent on task execution and the time spent on data movement, in order to speed up practical distributed computing applications. In particular, let us consider a MapReduce-type application for which the total execution time is roughly composed of the time spent executing the Map tasks, denoted by T_{map}, the time spent shuffling intermediate values, denoted by T_{shuffle}, and the time spent executing the Reduce tasks, denoted by T_{reduce}, i.e.,

$$T_{\text{total, MR}}\approx T_{\text{map}}+T_{\text{shuffle}}+T_{\text{reduce}}. \tag{101}$$

Using CDC, we can leverage $r\times$ more computations in the Map phase, in order to reduce the communication load by the same multiplicative factor. Hence, ignoring the coding overheads, CDC promises an approximate total execution time of

$$T_{\text{total, CDC}}\approx rT_{\text{map}}+\tfrac{1}{r}T_{\text{shuffle}}+T_{\text{reduce}}. \tag{102}$$

To minimize the above execution time, one would choose $r^*=\left\lfloor\sqrt{\frac{T_{\text{shuffle}}}{T_{\text{map}}}}\right\rfloor$ or $\left\lceil\sqrt{\frac{T_{\text{shuffle}}}{T_{\text{map}}}}\right\rceil$, resulting in the minimum execution time of

$$T_{\text{total, CDC}}^*\approx 2\sqrt{T_{\text{shuffle}}T_{\text{map}}}+T_{\text{reduce}}. \tag{103}$$

For example, in an application that T_{shuffle} is $10\times$ - $100\times$ larger than $T_{\text{map}}+T_{\text{reduce}}$, by comparing from (101) and (103), we note that CDC can reduce the execution time by approximately $1.5\times$ - $5\times$.

In the rest of this section, we empirically demonstrate the performance gain of applying CDC to `TeraSort` [11], which is a commonly used Hadoop benchmark for distributed sorting terabytes of data [32]. In particular, we first incorporate the coding ideas in CDC into `TeraSort` to develop a novel coded distributed sorting algorithm, named `CodedTeraSort`, which imposes *structured* redundancy in the input data, in order to enable in-network coding opportunities that overcome the data shuffling bottleneck of `TeraSort`. Then, we evaluate the performance of `CodedTeraSort` on Amazon EC2 clusters, and observe a $1.97\times$ - $3.39\times$ speedup, compared with `TeraSort`, for typical settings of interest.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIT.2017.2756959, IEEE Transactions on Information Theory

A. TeraSort

`TeraSort` [32] is a conventional algorithm for distributed sorting of a large amount of data. The input data that is to be sorted is in the format of key-value (KV) pairs, meaning that each input KV pair consists of a key and a value. For example, the domain of the keys can be 10-byte integers, and the domain of the values can be arbitrary strings. `TeraSort` sorts the input data according to their keys, e.g., sorting integers.

1) Algorithm Description: Let us consider implementing `TeraSort` over K distributed computing nodes, which consists of 5 stages: File Placement, Key Domain Partitioning, Map Phase, Shuffle Phase, and Reduce Phase. In File Placement, all input KV pairs are split into K disjoint files, and each file is placed on one of the K nodes. In Key Domain Partitioning, the domain of the keys is split into K partitions, and each node will be responsible for sorting the KV pairs whose keys fall into one of the partitions. In Map Phase, each node hashes each KV pair in its locally stored file into one of the K partitions, according to its key. In Shuffle Phase, the KV pairs in the same partition are transferred to the node that is responsible for sorting that partition. In Reduce Stage, each node locally sorts KV pairs belonging to its assigned partition. We illustrate the `TeraSort` algorithm using a simple example shown in Fig. 7.

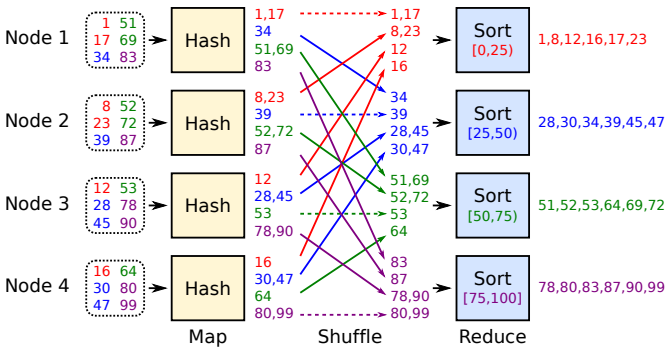


Fig. 7: Illustration of `TeraSort` algorithm with $K = 4$ nodes and key domain partitions $[0, 25), [25, 50), [50, 75), [75, 100]$. A dotted box represents an input file. An input file is hashed into 4 groups of KV pairs, one for each partition. For each of the 4 partitions, the KV pairs belonging to that partition computed on all 4 nodes are fetched to a corresponding node, which sorts all KV pairs in that partition locally.

2) Performance Evaluation: To understand the performance of `TeraSort`, we performed an experiment on Amazon EC2 to sort 12GB of data by running `TeraSort` on 16 instances.[3] The breakdown of the total execution time is shown in Table I.

TABLE I: Performance of `TeraSort` sorting 12GB data with $K = 16$ instances and 100 Mbps network speed

Map (sec.)	Pack (sec.)	Shuffle (sec.)	Unpack (sec.)	Reduce (sec.)	Total (sec.)
1.86	2.35	945.72	0.85	10.47	961.25

We observe from Table I that for a conventional `TeraSort` execution, 98.4% of the total execution time was spent in data shuffling, which is $508.5\times$ of the time spent in the Map phase. Given the fact that data shuffling dominates the job execution time, the principle of optimally trading computation for communication of the proposed CDC scheme can be applied to significantly improve the performance of `TeraSort`. For example, when executing the same sorting job using a coded version of `TeraSort` with a computation load of $r = 10$, according to (102), we could theoretically save the total execution time by approximately $8\times$. This motivates us to develop a novel coded distributed sorting algorithm, named `CodedTeraSort`, which is briefly described in the next sub-section.

[3]We note that EC2 uses virtual machines, and each instance may not be hosted by a dedicated physical machine.

B. Coded TeraSort

We develop the `CodedTeraSort` algorithm by applying the proposed CDC scheme for the case of $s = 1$ (see Example 1 in Section IV for an illustration) to the above described `TeraSort` algorithm. `CodedTeraSort` exploits redundant computations on the input files in the Map phase, creating in-network coding opportunities to significantly slash the load of data shuffling. In particular, the execution of `CodedTeraSort` consists of following 6 stages of operations. Here we give high-lever descriptions of these operations, and we refer the interested readers to [3] for more detailed descriptions.

1) *Structured Redundant File Placement*. The entire input KV pairs are split into many small files, each of which is repeatedly placed on $1 \leq r \leq K$ nodes (i.e., a computation load of r), according to the particular pattern specified by the CDC scheme.
2) *Map*. Each node applies the hashing operation as in `TeraSort` on each of its assigned files.
3) *Encoding to Create Coded Packets*. Each node generates coded multicast packets from local results computed in Map phase, according to the encoding process of the CDC scheme.
4) *Multicast Shuffling*. Each node multicasts each of its generated coded packet to a specific set of r other nodes.
5) *Decoding*. Each node locally decodes the required KV pairs from the received coded packets.
6) *Reduce*. Each node locally sorts the KV pairs within its assigned partition as in the Reduce phase of `TeraSort`.

C. Empirical Evaluations

We imperially demonstrate the performance gain of `CodedTeraSort` through experiments on Amazon EC2 clusters. In this sub-section, we first present some choices we have made for the implementation. Then, we discuss the experiment results.

1) Implementation Choices: We first describe the following common implementation choices that we have made for both `TeraSort` and `CodedTeraSort` algorithms.

Data Format: All input KV pairs are generated from `TeraGen` [11] in the standard Hadoop package. Each input KV pair consists of a 10-byte key and a 90-byte value. A key is a 10-byte unsigned integer, and the value is an arbitrary

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIT.2017.2756959, IEEE Transactions on Information Theory

string of 90 bytes. The KV pairs are sorted based on their keys, using the standard integer ordering.

Library: We implement both `TeraSort` and `CodedTeraSort` algorithms in `C++`, and use Open MPI library [33] for communications between EC2 instances.

System Architecture: We employ a system architecture that consists of a coordinator node and K worker nodes, for some $K \in \mathbb{N}$. Each node is run as an EC2 instance. The coordinator node is responsible for creating the key partitions and placing the input files on the local disks of the worker nodes. The worker nodes are responsible for distributedly executing the stages of the sorting algorithms.

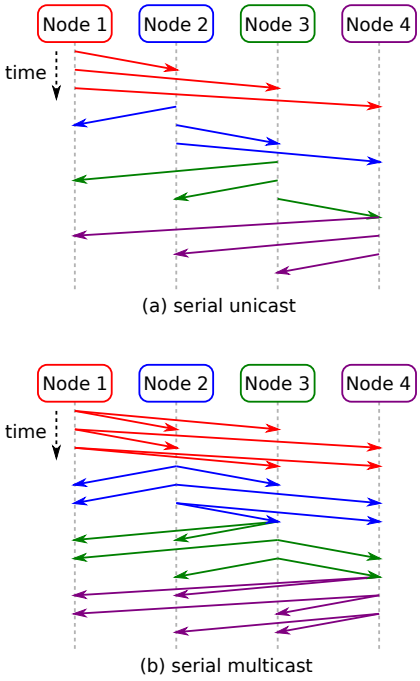


Fig. 8: (a) Serial unicast in the Shuffle phase of `TeraSort`; a solid arrow represents a unicast. (b) Serial multicast in the Multicast Shuffle phase of `CodedTeraSort`; a group of solid arrows starting at the same node represents a multicast.

In the `TeraSort` implementation, each node sequentially steps through Map, Pack, Shuffle, Unpack, and Reduce stages. The Pack stage serializes each intermediate value to a continuous memory array to ensure that a single TCP flow is created for each intermediate value (which may contain multiple KV pairs) when `MPI_Send` is called[4]. The Unpack stage deserializes the received data to a list of KV pairs. In the Shuffle stage, intermediate values are unicast serially, meaning that there is only one sender node and one receiver node at any time instance. Specifically, as illustrated in Fig. 8(a), Node 1 starts to unicast to Nodes 2, 3, and 4 back-to-back. After Node 1 finishes, Node 2 unicasts back-to-back to Nodes 1, 3, and 4. This continues until Node 4 finishes.

In the `CodedTeraSort` implementation, each node sequentially steps through CodeGen, Map, Encode, Multicast Shuffling, Decode, and Reduce stages. In the CodeGen (or code generation) stage, firstly, each node generates all file indices, as subsets of r nodes. Then each node uses `MPI_Comm_split` to initialize $\binom{K}{r+1}$ multicast groups each containing $r+1$ nodes on Open MPI, such that multicast communications will be performed within each of these groups. The serialization and deserialization are implemented respectively in the Encode and the Decode stages. In Multicast Shuffling, `MPI_Bcast` is called to multicast a coded packet in a serial manner, so only one node multicasts one of its encoded packets at any time instance. Specifically, as illustrated in Fig. 8(b), Node 1 multicasts to the other 2 nodes in each multicast group Node 1 is in. For example, Node 1 first multicasts to Node 2 and 3 in the multicast group $\{1,2,3\}$. After Node 1 finishes, Node 2 starts multicasting in the same manner. This process continues until Node 4 finishes.

2) Experiment Results: We evaluate the run-time performance of `TeraSort` and `CodedTeraSort`, for different combinations of the number of workers K and the computation load $1 \leq r \leq K$. All experiments are repeated 5 times, and the average values are recorded.

In Table II and Table III, we list the breakdowns of the average execution times to sort 12 GB of input data using $K = 16$ workers and $K = 20$ workers respectively. Here we limit the incoming and outgoing traffic rates of each instance to 100 Mbps. This is to alleviate the effects of the bursty behaviors of the transmission rates in the beginning of some TCP sessions, given the particular size of the data to be sorted. We observe an overall $1.97\times$ - $3.39\times$ speedup of `CodedTeraSort` as compared with `TeraSort`. From the experiment results we make the following observations:

- For `CodedTeraSort`, the time spent in the CodeGen stage is proportional to $\binom{K}{r+1}$, which is the number of multicast groups.
- The Map time of `CodedTeraSort` is approximately r times higher than that of `TeraSort`. This is because that each node hashes r times more KV pairs than that in `TeraSort`. Specifically, the ratios of the `CodedTeraSort`'s Map time to the `TeraSort`'s Map time from Table II are $6.03/1.86 \approx 3.2$ and $10.84/1.86 \approx 5.8$, and from Table III are $4.68/1.47 \approx 3.2$ and $8.59/1.47 \approx 5.8$.
- While `CodedTeraSort` theoretically promises a factor of more than $r\times$ reduction in shuffling time, the actual gains observed in the experiments are slightly less than r. For example, for the experiment with $K = 16$ nodes and $r = 3$, as shown in Table II, the speedup of the Shuffle stage is $945.72/412.22 \approx 2.3 < 3$. This phenomenon is caused by the following two factors. 1) Open MPI's multicast API (`MPI_Bcast`) has an inherent overhead per a multicast group, for instance, a multicast tree is constructed before multicasting to a set of nodes. 2) Using the `MPI_Bcast` API, the time of multicasting a packet to r nodes is higher than that of unicasting the same packet to a single node. In fact, as measured in [24], the multicasting time increases logarithmically with r.

Further, we observe the following trends from both tables:

The impact of computation load r: As r increases, the shuffling time reduces by approximately r times. However, the Map execution time increases linearly with r, and more

[4]Creating a TCP flow per KV pair leads to inefficiency from overhead and convergence issue.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIT.2017.2756959, IEEE Transactions on Information Theory

TABLE II: Sorting 12 GB data with $K = 16$ worker instances and 100 Mbps network speed

	CodeGen (sec.)	Map (sec.)	Pack/Encode (sec.)	Shuffle (sec.)	Unpack/Decode (sec.)	Reduce (sec.)	Total Time (sec.)	Speedup
`TeraSort`:	–	1.86	2.35	945.72	0.85	10.47	961.25	
`CodedTeraSort`: $r = 3$	6.06	6.03	5.79	412.22	2.41	13.05	445.56	2.16×
`CodedTeraSort`: $r = 5$	23.47	10.84	8.10	222.83	3.69	14.40	283.33	3.39×

TABLE III: Sorting 12 GB data with $K = 20$ worker instances and 100 Mbps network speed

	CodeGen (sec.)	Map (sec.)	Pack/Encode (sec.)	Shuffle (sec.)	Unpack/Decode (sec.)	Reduce (sec.)	Total Time (sec.)	Speedup
`TeraSort`:	–	1.47	2.00	960.07	0.62	8.29	972.45	
`CodedTeraSort`: $r = 3$	19.32	4.68	4.89	453.37	1.87	9.73	493.86	1.97×
`CodedTeraSort`: $r = 5$	140.91	8.59	7.51	269.42	3.70	10.97	441.10	2.20×

importantly the CodeGen time increases exponentially with r as $\binom{K}{r+1}$. Hence, for small values of r ($r < 6$) we observe overall reduction in execution time, and the speedup increases. However, as we further increase r, the CodeGen time will dominate the execution time, and the speedup decreases. Hence, in our evaluations, we have limited r to be at most 5.[5]

The impact of worker number K: As K increases, the speedup decreases. This is due to the following two reasons. 1) The number of multicast groups, i.e., $\binom{K}{r+1}$, grows exponentially with K, resulting in a longer execution time of the CodeGen process. 2) When more nodes participate in the computation, for a fixed r, less amount of KV pairs are hashed at each node locally in the Map phase, resulting in less locally available intermediate values and a higher communication load. Hence, given more worker nodes, one would preferably use larger computation load to achieve a better run-time performance.

IX. Concluding Remarks and Future Directions

We introduced a scalable distributed computing framework motivated by MapReduce, which is suited for arbitrary types of output functions. We formulated and exactly characterized an information-theoretic tradeoff between computation load and communication load within this framework. In particular, we proposed Coded Distributed Computing (CDC), a coded scheme that reduces the communication load by a factor that can grow with the network size, illustrating the role of coding in speeding up distributed computing jobs. We also proved a tight information-theoretic lower bound on the minimum communication load, using any data shuffling scheme, which exactly matches the communication load achieved by CDC. This result reveals a fundamental relationship between computation and communication in distributed computing–the two are inversely proportional to each other. Moreover, we applied the proposed CDC scheme to the conventional `TeraSort` algorithm to develop a novel distributed sorting algorithm, named `CodedTeraSort`, and empirically demonstrated the performance gain of `CodedTeraSort` through extensive experiments on Amazon EC2 clusters.

Finally, we discuss some follow-up research directions of this work.

[5]The redundancy parameter r is also limited by the total storage available at the nodes. Since for a choice of redundancy parameter r, each piece of input KV pairs should be stored at r nodes, we can not increase r beyond $\frac{\text{total available storage at the worker nodes}}{\text{input size}}$.

Heterogeneous Networks with Asymmetric Tasks. It is common to have computing nodes with heterogeneous storage, processing and communication capacities within computer clusters (e.g., Amazon EC2 clusters composed of heterogeneous computing instances). In addition, processing different parts of the dataset can generate intermediate results with different sizes (e.g., performing data analytics on highly-clustered graphs). For computing over heterogeneous nodes, one solution is to break the more powerful nodes into multiple smaller virtual nodes that have homogeneous capability, and then apply the proposed CDC scheme for the homogeneous setting. When intermediate results have different sizes, the proposed coding scheme still applies, but the coding operations are not symmetric as in the case of homogeneous intermediate results (e.g., one may now need to compute the XOR of two data segments with different sizes). Alternatively, we can employ a low-complexity greedy approach, in which we assign the Map tasks to maximize the number of multicasting opportunities that simultaneously deliver useful information to the largest possible number of nodes. Some preliminary studies along this direction have been conducted to obtain the solutions for some special cases (see, e.g., [34], [35]). Nevertheless, systematically characterizing the optimal resource allocation strategies and coding schemes for general heterogeneous networks with asymmetric tasks remains an interesting open problem.

Straggling/Failing Computing Nodes. Other than the communication bottleneck, the effect of straggling servers also severely degrades the run-time performance of distributed computing applications (see e.g., [36]). Recently in [24], Maximum-Distance-Separable (MDS) codes were utilized to encode linear computation tasks, providing robustness to a certain number of stragglers. Following the results in [24], coded computing strategies have been proposed to efficiently deal with the stragglers for various computation tasks and network settings (see, e.g., [37]–[40]). In [41], we have superimposed the proposed CDC scheme on top of the MDS codes, developing a unified coding framework for distributed computing with straggling servers. This framework achieves a flexible tradeoff between computation latency in the Map phase and communication load in the Shuffle phase, which has the CDC scheme (or minimum bandwidth code) and the MDS code (or minimum latency code) as the two end points. Nevertheless, designing resource allocation strategies and coding techniques to optimize the run-time performance over distributed computing clusters with stragglers is a challenging

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIT.2017.2756959, IEEE Transactions on Information Theory

open problem.

Multi-Stage Computation Tasks. Unlike simple computation tasks like Grep, Join and Sort, many distributed computing applications contain multiples stages of MapReduce computations. Examples of these applications include machine learning algorithms [42], SQL queries for databases [43], [44], and scientific analytics [45]. One can express the computation logic of a multi-stage application as a directed acyclic graph (DAG) [46], in which each vertex represents a logical step of data transformation, and each edge represents the dataflow across processing vertices. In order to speed up multi-stage computation tasks using codes, while one straightforward approach is to apply the proposed CDC scheme for the cascaded distributed computing framework (see Theorem 2) to compute each stage locally, we expect to achieve a higher reduction in bandwidth consumption and response time by globally designing codes for the entire task graph and accounting for interactions between consecutive stages. A preliminary exploration along this direction was recently presented in [47].

Multi-Layer Networks and Structured Topology. So far we have only considered a single-layer topology of the distributed computing nodes, in which each node can multicast to an arbitrary number of other nodes at the same cost as unicasting to a single node. However, in practical data center networks, nodes can be connected through multiple switches at different layers with different capacities, forming a hierarchical multi-root tree topology (e.g., fat-tree topology [48]). In this case, we need to generalize our communication model to include more structured topologies, and develop coded shuffling strategies that account for (1) path lengths of shuffled data (2) congestion at links higher up in the topology; and (3) different link capacities and multicast-costs at different layers of network topology. We have made preliminary progress in [49] for a star topology (motivated by wireless edge computing), where nodes are connected via only one access point (or switch layer).

Joint Storage and Computation Optimization. We have so far assumed that we can design the placement of the input files to create coding opportunities during the computation process. However, in practical file storage systems, data blocks are often stored without prior knowledge about the computations that will be performed on them, and moving the data across the nodes before the computation is often too costly. In this case, even without the capability of designing the data placement as exactly specified by the CDC scheme, one can still take advantage of the inherent data redundancy (e.g., GFS [50] and HDFS [51] by default place replicas of each data block on 3 distributed nodes) to create coded multicast opportunities, significantly reducing the communication load.

We plot in Fig. 9 the average communication load achieved by a coded shuffling scheme similar to the one presented in Section V-B (with the modification that each node zero-pads its associated data segments to the length of the longest one before coding), when each input file is placed and mapped at r out of K nodes chosen *uniformly at random*, and compare it with the communication load achieved by CDC where the input files are placed based on the Map phase design in Section V-A. As demonstrated in Fig. 9, without requiring the files to be placed as exactly described by the CDC scheme, one can still exploit the data redundancy to achieve a communication load that is superlinear with respect to the computation load. Therefore, the coded data shuffling scheme of CDC can effectively reduce the communication loads of computation jobs on general data storage systems. This behavior that a random data placement achieves close-to-optimum performance has also been reported in [49] for a decentralized wireless distributed computing platform, and in [21] for a decentralized caching system.

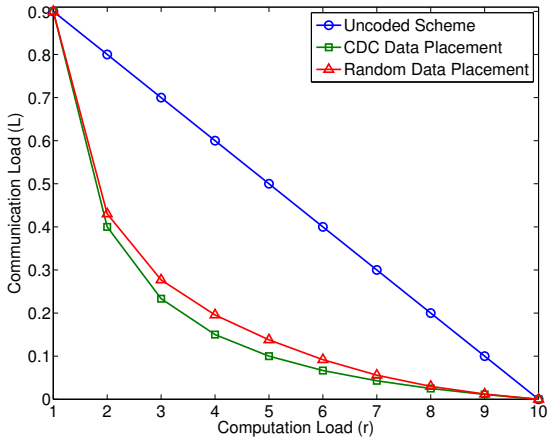


Fig. 9: Comparison of the average communication load by placing and mapping every input file randomly at r out of $K = 10$ nodes with the communication load achieved by placing the files specified by the CDC scheme. Here we compute $Q = 10$ output functions from 2520 input files using $K = 10$ distributed computing nodes.

Coded Edge/Fog Computing. In the emerging mobile Edge/Fog computing paradigm (see, e.g., [52], [53]), abundant computation resources scattered across the network edge (e.g., smartphones, tablets and smart cars) are harvested to perform data-intensive computations collaboratively. In this scenario, coding opportunities are widely available by injecting redundant storage and computations into the edge network. We envision codes to play a transformational role in Edge/Fog computing for leveraging such redundancy to substantially reduce the bandwidth consumption and the latency of computing. For an edge computing scenario where the mobile users upload the tasks to the edge nodes, and retrieve the computed results from the edge nodes, we have designed coded computing architectures in [54], [55], in which coded computations that are aware of the underlying physical-layer communication are performed at the edge nodes, achieving the minimum load of computation and the maximum spectral efficiency simultaneously. In [49], we have formulated a wireless distributed computing framework, in which a cluster of mobile users collaborate via an access point to simultaneously meet their computational needs. For this wireless computing platform, we exploited the coding techniques of CDC to achieve a scalable design such that the platform can accommodate an unlimited number of mobile users with a constant amount of bandwidth consumption. Also in a recent magazine paper [56], we have demonstrated the opportunities of utilizing coding to improve the performance of Edge/Fog computing applications (e.g., navigation services and recommendation systems).

REFERENCES

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

[2] ——, "Fundamental tradeoff between computation and communication in distributed computing," *IEEE International Symposium on Information Theory*, July 2016.

[3] S. Li, S. Supittayapornpong, M. A. Maddah-Ali, and A. S. Avestimehr, "Coded terasort," *6th International Workshop on Parallel and Distributed Computing for Large Scale Machine Learning and Big Data Analytics*, 2017.

[4] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Sixth USENIX Symposium on Operating System Design and Implementation*, Dec. 2004.

[5] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: cluster computing with working sets," in *Proceedings of the 2nd USENIX HotCloud*, vol. 10, June 2010, p. 10.

[6] F. Ahmad, S. T. Chakradhar, A. Raghunathan, and T. Vijaykumar, "Tarazu: optimizing MapReduce on heterogeneous clusters," in *ACM SIGARCH Computer Architecture News*, vol. 40, no. 1, Mar. 2012, pp. 61–74.

[7] Y. Guo, J. Rao, and X. Zhou, "iShuffle: Improving Hadoop performance with shuffle-on-write," in *Proceedings of the 10th International Conference on Autonomic Computing*, June 2013, pp. 107–117.

[8] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica, "Managing data transfers in computer clusters with orchestra," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 98–109, Aug. 2011.

[9] Z. Zhang, L. Cherkasova, and B. T. Loo, "Performance modeling of MapReduce jobs in heterogeneous cloud environments," in *IEEE Sixth International Conference on Cloud Computing*, June 2013, pp. 839–846.

[10] Amazon.com, "Amazon Elastic Compute Cloud (Amazon EC2)," https://aws.amazon.com/ec2/.

[11] "Hadoop TeraSort," https://hadoop.apache.org/docs/r2.7.1/api/org/apache/hadoop/examples/terasort/package-summary.html.

[12] A. C.-C. Yao, "Some complexity questions related to distributive computing (preliminary report)," in *Proceedings of the eleventh annual ACM symposium on Theory of computing*, Apr. 1979, pp. 209–213.

[13] J. Korner and K. Marton, "How to encode the modulo-two sum of binary sources," *IEEE Transactions on Information Theory*, vol. 25, no. 2, pp. 219–221, Mar. 1979.

[14] A. Orlitsky and A. El Gamal, "Average and randomized communication complexity," *IEEE Transactions on Information Theory*, vol. 36, no. 1, pp. 3–16, Jan. 1990.

[15] K. Becker and U. Wille, "Communication complexity of group key distribution," in *Proceedings of the 5th ACM conference on Computer and communications security*, Nov. 1998, pp. 1–6.

[16] E. Kushilevitz and N. Nisan, *Communication Complexity*. Cambridge University Press, 2006.

[17] A. Orlitsky and J. Roche, "Coding for computing," *IEEE Transactions on Information Theory*, vol. 47, no. 3, pp. 903–917, Mar. 2001.

[18] B. Nazer and M. Gastpar, "Computation over multiple-access channels," *IEEE Transactions on Information Theory*, vol. 53, no. 10, pp. 3498–3516, Oct. 2007.

[19] A. Ramamoorthy and M. Langberg, "Communicating the sum of sources over a network," *IEEE Journal on Selected Areas in Communications*, vol. 31, no. 4, pp. 655–665, Apr. 2013.

[20] M. A. Maddah-Ali and U. Niesen, "Fundamental limits of caching," *IEEE Transactions on Information Theory*, vol. 60, no. 5, pp. 2856–2867, Mar. 2014.

[21] ——, "Decentralized coded caching attains order-optimal memory-rate tradeoff," *IEEE/ACM Transactions on Networking*, Apr. 2014.

[22] M. Ji, G. Caire, and A. F. Molisch, "Fundamental limits of caching in wireless D2D networks," *IEEE Transactions on Information Theory*, vol. 62, no. 2, pp. 849–869, Feb. 2016.

[23] N. Karamchandani, U. Niesen, M. A. Maddah-Ali, and S. Diggavi, "Hierarchical coded caching," *IEEE International Symposium on Information Theory*, pp. 2142–2146, June 2014.

[24] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," *IEEE Transactions on Information Theory*, 2017.

[25] A. Rajaraman and J. D. Ullman, *Mining of massive datasets*. Cambridge University Press, 2011.

[26] Q. Yu, S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, "How to optimally allocate resources for coded distributed computing?" *IEEE International Conference on Communications (ICC)*, pp. 1–7, May 2017.

[27] Y. Birk and T. Kol, "Coding on demand by an informed source (ISCOD) for efficient broadcast of different supplemental data to caching clients," *IEEE Transactions on Information Theory*, vol. 52, no. 6, pp. 2825–2830, June 2006.

[28] Z. Bar-Yossef, Y. Birk, T. Jayram, and T. Kol, "Index coding with side information," *IEEE Transactions on Information Theory*, vol. 57, no. 3, pp. 1479–1494, Mar. 2011.

[29] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung, "Network information flow," *IEEE Transactions on Information Theory*, vol. 46, no. 4, pp. 1204–1216, July 2000.

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

[31] T. Ho, R. Koetter, M. Medard, D. R. Karger, and M. Effros, "The benefits of coding over routing in a randomized setting," *IEEE International Symposium on Information Theory*, pp. 442–, June 2003.

[32] O. O'Malley, "Terabyte sort on Apache Hadoop," Yahoo, Tech. Rep., May 2008, http://sortbenchmark.org/YahooHadoop.pdf.

[33] "Open MPI: Open source high performance computing," https://www.open-mpi.org/.

[34] A. Reisizadeh, S. Prakash, R. Pedarsani, and S. Avestimehr, "Coded computation over heterogeneous clusters," in *IEEE International Symposium on Information Theory*, 2017, pp. 2408–2412.

[35] M. Kiamari, C. Wang, and A. S. Avestimehr, "On heterogeneous coded distributed computing," *e-print arXiv:1709.00196*, 2017, to appear in IEEE GLOBECOM 2017.

[36] M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz, and I. Stoica, "Improving MapReduce performance in heterogeneous environments," *OSDI*, vol. 8, no. 4, p. 7, Dec. 2008.

[37] S. Dutta, V. Cadambe, and P. Grover, "Short-dot: Computing large linear transforms distributedly using coded short dot products," in *Advances In Neural Information Processing Systems (NIPS)*, 2016, pp. 2100–2108.

[38] K. Lee, C. Suh, and K. Ramchandran, "High-dimensional coded matrix multiplication," *IEEE International Symposium on Information Theory*, pp. 2418–2422, 2017.

[39] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Polynomial codes: an optimal design for high-dimensional coded matrix multiplication," *to appear in Advances In Neural Information Processing Systems (NIPS)*, 2017.

[40] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient coding: Avoiding stragglers in distributed learning," in *Proceedings of the 34th International Conference on Machine Learning (ICML)*, Aug. 2017, pp. 3368–3376.

[41] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, "A unified coding framework for distributed computing with straggling servers," *IEEE NetCod*, 2016.

[42] C. Chu, S. K. Kim, Y.-A. Lin, Y. Yu, G. Bradski, A. Y. Ng, and K. Olukotun, "Map-Reduce for machine learning on multicore," *Advances in neural information processing systems*, vol. 19, 2007.

[43] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: distributed data-parallel programs from sequential building blocks," in *ACM SIGOPS Operating Systems Review*, vol. 41, no. 3, June 2007, pp. 59–72.

[44] A. Abouzeid, K. Bajda-Pawlikowski, D. Abadi, A. Silberschatz, and A. Rasin, "HadoopDB: an architectural hybrid of MapReduce and DBMS technologies for analytical workloads," *Proceedings of the VLDB Endowment*, vol. 2, no. 1, pp. 922–933, Aug. 2009.

[45] J. Ekanayake, T. Gunarathne, G. Fox, A. S. Balkir, C. Poulain, N. Araujo, and R. Barga, "DryadLINQ for scientific analyses," in *Fifth IEEE International Conference on e-Science*, 2009, pp. 329–336.

[46] B. Saha, H. Shah, S. Seth, G. Vijayaraghavan, A. Murthy, and C. Curino, "Apache Tez: A unifying framework for modeling and building data processing applications," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, May 2015, pp. 1357–1369.

[47] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, "Coded distributed computing: Straggling servers and multistage dataflows," *54th Allerton Conference on Communication, Control, and Computing*, Sept. 2016.

[48] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4, pp. 63–74, Oct. 2008.

[49] S. Li, Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "A scalable framework for wireless distributed computing," *IEEE/ACM Transactions on Networking*, pp. 1–12, 2017, a shorter version is in IEEE GLOBECOM 2016.

[50] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," in *ACM SIGOPS operating systems review*, vol. 37, no. 5, Dec. 2003, pp. 29–43.
[51] "Apache Hadoop Distributed File System," https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html.
[52] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, ser. MCC '12. ACM, Aug. 2012, pp. 13–16.
[53] M. Chiang and T. Zhang, "Fog and IoT: An overview of research opportunities," *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 854–864, Dec. 2016.
[54] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, "Communication-aware computing for edge processing," *IEEE International Symposium on Information Theory*, pp. 2885–2889, 2017.
[55] ——, "Architectures for coded mobile edge computing," *to appear in Fog World Congress*, 2017.
[56] ——, "Coding for distributed fog computing," *IEEE Communications Magazine*, vol. 55, no. 4, pp. 34–40, Apr. 2017.

BIOGRAPHIES

Songze Li (S'09) received his B.S. in Electrical Engineering from Polytechnic Institute of New York University (now NYU Tandon School of Engineering) in 2011, and M.S. in Electrical Engineering from University of Southern California in 2016. He is currently pursuing his Ph.D. as a Research Assistant at the Electrical Engineering Department, University of Southern California. His research interest is network information theory and its applications including improving parallel/distributed computing using codes, and interference management in wireless networks.

Songze is a Qualcomm Innovation Fellowship finalist in 2017, and received the USC Viterbi School of Engineering Doctoral Fellowship in 2011.

Mohammad Ali Maddah-Ali (S'03-M'08) received the B.Sc. degree from Isfahan University of Technology, and the M.A.Sc. degree from the University of Tehran, both in electrical engineering. From 2002 to 2007, he was with the Coding and Signal Transmission Laboratory (CST Lab), Department of Electrical and Computer Engineering, University of Waterloo, Canada, working toward the Ph.D. degree. From 2007 to 2008, he worked at the Wireless Technology Laboratories, Nortel Networks, Ottawa, ON, Canada. From 2008 to 2010, he was a post-doctoral fellow in the Department of Electrical Engineering and Computer Sciences at the University of California at Berkeley. Then, he joined Bell Labs, Holmdel, NJ, as a communication research scientist. Recently, he started working at Sharif University of Technology, as a faculty member.

Dr. Maddah-Ali is a recipient of NSERC Postdoctoral Fellowship in 2007, a best paper award from IEEE International Conference on Communications (ICC) in 2014, the IEEE Communications Society and IEEE Information Theory Society Joint Paper Award in 2015, and the IEEE Information Theory Society Joint Paper Award in 2016.

Qian Yu (S'16) is pursuing his Ph.D. degree in Electrical Engineering at University of Southern California (USC), Viterbi School of Engineering. He received his M.Eng. degree in Electrical Engineering and B.S. degree in EECS and Physics, both from Massachusetts Institute of Technology (MIT). His interests span information theory, distributed computing, and many other problems math-related.

Qian received the IEEE Jack Keil Wolf ISIT Student Paper Award in 2017. He is a Qualcomm Innovation Fellowship finalist in 2017, and received the Annenberg Graduate Fellowship in 2015. He received Honorable Mention in the William Lowell Putnam Mathematical Competition in 2013.

A. Salman Avestimehr (SM) is an Associate Professor at the Electrical Engineering Department of University of Southern California. He received his Ph.D. in 2008 and M.S. degree in 2005 in Electrical Engineering and Computer Science, both from the University of California, Berkeley. Prior to that, he obtained his B.S. in Electrical Engineering from Sharif University of Technology in 2003. His research interests include information theory, the theory of communications, and their applications to distributed computing and data analytics.

Dr. Avestimehr has received a number of awards, including the Communications Society and Information Theory Society Joint Paper Award, the Presidential Early Career Award for Scientists and Engineers (PECASE) for "pushing the frontiers of information theory through its extension to complex wireless information networks", the Young Investigator Program (YIP) award from the U. S. Air Force Office of Scientific Research, the National Science Foundation CAREER award, and the David J. Sakrison Memorial Prize. He is currently an Associate Editor for the IEEE Transactions on Information Theory.