Distributed Gradient Descent with Coding and Partial Recovery

Emre Ozfatura, Sennur Ulukus, and Deniz Gündüz

***Abstract*—Coded computation techniques provide robustness against *straggling* workers in distributed computing. However, in addition to increasing the decoding complexity, most of the existing schemes ignore the computations carried out by straggling workers; and they are typically designed to recover the full gradient, and thus, cannot provide a balance between the accuracy of the gradient estimate and the per-iteration completion time. In this paper, we introduce a novel approach, called *coded computation with partial recovery (CCPR)*, which benefits from the advantages of both coded and uncoded computation schemes, and reduces both the computation time and the decoding complexity by allowing a trade-off between the accuracy of the gradients estimate at each iteration and the computation time.**

***Index Terms*—Coded computation, distributed computation, maximum distance separable (MDS) code, linear codes, rateless codes, straggler mitigation**

I. Introduction

One of the key enablers of efficient machine learning solutions is the availability of large datasets. However, the ever growing size of the datasets and the complexity of the models trained on them lead also to an increase in the computational complexity and storage requirements of the algorithms employed. As a consequence, carrying out these demanding computation tasks within reasonable time frames becomes unmanageable within the resources of a single machine. Distributed computation framework has been designed to remedy the limitations and speed up the computation for massive machine learning algorithms by hatnessing the computation and memory resources of multiple machines, referred to as *workers*.

One of the most often used strategy for distributed computation is the *parameter server(PS)* type implementation, where a central PS divides the main computational task into several subtasks and assign them to workers. Then the workers execute their assigned tasks and convey the results to the PS, which combines them to obtain the result of the main computation task. In principle, such a distributed computation framework should achieve a speed-up factor proportional to the number of workers employed. However, in real implementations, the overall computation time is constrained by the slowest, so-called *straggling worker(s)* due to synchronised updates. Moreover, as the number of employed workers increase, communication between them and the PS becomes more complex, and introduces additional delays, which can aggravate the straggler problem.

To remedy the delays due to straggling workers various straggler-tolerant distributed computation schemes have been introduced recently, which build upon the idea of assigning redundant computations/subtasks to workers, to let faster workers compensate for the stragglers [1]–[41].

A. Preliminaries

In many machine learning problems, the principal computational task boils down to a matrix-vector multiplication. Consider, for example, the minimization of the loss function that is empirical mean squared error in linear regression:

$$L(\boldsymbol{\theta}) \triangleq \frac{1}{2N}\sum_{i=1}^{N}(y_i - \mathbf{x}_i^T\boldsymbol{\theta})^2, \tag{1}$$

where $\mathbf{x}_1,\dots,\mathbf{x}_N \in \mathbb{R}^d$ are the data points with corresponding labels $y_1,\dots,y_N \in \mathbb{R}$, and $\boldsymbol{\theta} \in \mathbb{R}^d$ is the parameter vector. The optimal parameter vector can be obtained iteratively by the gradient descent (GD) method, in which the parameter vector is updated iteratively as follows:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta_t \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_t), \tag{2}$$

where η_t is the learning rate at the tth iteration. Gradient of the loss function in (1) can be written as

$$\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta_t}) = \mathbf{X}^T\mathbf{X}\boldsymbol{\theta}_t - \mathbf{X}^T\mathbf{y}, \tag{3}$$

where $\mathbf{X} = [\mathbf{x}_1,\dots,\mathbf{x}_N]^T$ and $\mathbf{y} = [y_1,\dots,y_N]^T$. In the gradient expression, only $\boldsymbol{\theta}_t$ changes over iterations; hence, the key computational task at each iteration is the matrix-vector multiplication $\mathbf{W}\boldsymbol{\theta}_t$, where $\mathbf{W} \triangleq \mathbf{X}^T\mathbf{X} \in \mathbb{R}^{d\times d}$.

Now assume that K worker machines are available, whose processing power can be harnessed to speed up GD. Accordingly, the execution of $\mathbf{W}\boldsymbol{\theta}_t$ can be distributed across these K *workers* by simply dividing $\mathbf{W}$ into K disjoint submatrices and assigning each submatrix to one of the workers. However, under this naive approach, the computation time will now be limited by the *straggling* worker(s). The main challenge in this setup arises because the straggling behaviour (due either to the computation speed of the workers or the delays in communication) varies over time, and its realization at each iteration is not known in advance. Although a statistical knowledge of the computation and communication latency for each worker can be acquired over time, and used for a more efficient allocation of computation tasks (e.g. as in [34], [36]) as well as the coding scheme employed, for the sake of

This paper was presented in part at the 2019 IEEE International Conference on Acoustics, Speech and Signal Processing in Brighton, UK

Emre Ozfatura and Deniz Gündüz are with Information Processing and Communications Lab, Department of Electrical and Electronic Engineering, Imperial College London Email: {m.ozfatura, d.gunduz} @imperial.ac.uk.

Sennur Ulukus is with Department of Electrical and Computer Engineering, University of Maryland.

This work was supported in part by the Marie Sklodowska-Curie Action SCAVENGE (grant agreement no. 675891), and by the European Research Council (ERC) Starting Grant BEACON (grant agreement no. 677854).

simplicity we assume homogeneous workers in this work.

Coded computation has been introduced to tolerate stragglers by encoding the data before it is distributed among the workers, to achieve certain redundancy [15]–[31]. One well-known method to introduce redundancy in a matrix-vector multiplication task is to utilize maximum distance separable (MDS) codes to encode data in advance and execute computations on coded data [15]. To elucidate the MDS-coded computation (MCC), consider the matrix-vector multiplication task $\mathbf{W}\boldsymbol{\theta}_t$ $\mathbf{W} \in \mathbb{R}^{d\times d}$. We can divide $\mathbf{W}$ into $\bar{K}$ disjoint submatrices, $\mathbf{W}_1, \dots, \mathbf{W}_{\bar{K}} \in \mathbb{R}^{\bar{d}\times d}$, $\bar{d} = d/\bar{K}$, which are then encoded with a $(\bar{K}, K)$ MDS code. Each coded submatrix is assigned to a different worker, which multiplies its assigned coded submatrix with $\boldsymbol{\theta}_t$ with its assigned coded submatrix, and returns the result to the PS. The PS can recover $\mathbf{W}\boldsymbol{\theta}_t$ from the results of any $\bar{K}$ workers.Note that, up to $K - \bar{K}$ stragglers can be tolerated with this MDS coding strategy at the expense of increasing the *computation load* of each worker by $r = K/\bar{K}$ [15]; that is, each worker is assigned r times more computations compared to the naive approach of equally dividing all the required computations among the workers.

B. Related works

1) Straggler avoidance schemes: Straggler-aware distributed learning frameworks are not limited to coded computation. Alternatively the PS can assign certain computational tasks, specifically the computation of partial gradients corresponding to certain subsets of data, to multiple servers, and once a worker completes the computations assigned to it, these partial gradients are encoded and sent to the PS, where these coded results are aggregated to obtain the full gradient [8]–[14]. Which we refer such schemes as *coded communication* (see Section V for further details) as the workers carry out computations on uncoded data, but convey their results to the PS in a coded manner.

It is also possible to mitigate the delay induced by straggling workers without employing any encoding scheme [1]–[6], [42]. For a more comprehensive overview and comparison of different straggler avoidance approaches we the readers to refer [43].

2) Computation-communication latency trade-off: Conventional straggler-aware designs, particularly those employing a coding strategy, limit the number of messages sent by each worker at each iteration to one. Under this limitation, straggler-aware schemes suffer from two drawbacks: *over-computation* and *under-utilization* [43]. To overcome these obstacles one can allow each worker to send multiple messages to the PS at each iteration, which we refer to as *multi-message communication (MMC)* [1], [3], [13], [16], [19], [23], [24], [38]. However, MMC may introduce additional delays due to the communication overhead. Hence, with MMC the objective is to find an optimal operating point that balances the computation and communication latencies [43]. One recent approach on coded computation with MMC is to utilize rateless codes [23] due to its advantages of better utilizing the computational resources and low decoding complexity at the PS. However, in practice, rateless codes reach the target coding rates only if the number of coded messages are sufficiently large, which may not be desired in distributed computation framework since it leads to a congestion at the PS. Hence, the design of a code structure for distributed computation that can reduce the computation time without inducing a overwhelming communication overhead is an open challenge that we address in this paper.

C. Contributions

We have highlighted the trade-off between the computation and communication latencies; however in the context of GD, we can introduce "update accuracy" as the third dimension to this trade-off. The most common iterative optimization framework for large scale learning problems is *stochastic gradient descent (SGD)*, which uses an estimate of the gradient, in (3) at each iteration, evaluated on a random subset of the dataset. Hence, by changing the size of the sampled dataset it is possible to seek a balance between the accuracy of the gradient estimate and the computation time.

On the other hand, a vast majority of the coded computation schemes in the literature are designed for full gradient recovery. Nevertheless, a simple uncoded computation scheme with MMC [3], [19], can exploit partial computations performed by straggling workers, while also providing the PS a certain flexibility to terminate an iteration when a sufficient number of computations are received. Accordingly, our goal here is to design a coded computing framework that can efficiently benefit from redundant computations with the flexibility of partial gradient computations. To this end, we introduce a novel hybrid scheme, called *coded computation with partial recovery*, bringing together the advantages of uncoded computation, such as low decoding complexity and partial gradient updates, with those of coded computation, such as reduced per-iteration completion time and reduced communication load. Yet another advantage of the proposed straggler mitigating code structure is that due to induced degree limitation on the codewords, data encoding can be also done in a decentralized manner. This local encoding provides two more key advantages; first it is possible to change codewords over time based on straggler observations [Globecomm], second it provides a more generic scheme in a sense that it is not limited to coded computation scenario where the computation is mostly a linear operation.

To the best of our knowledge, the partial recovery approach was first introduced in our preliminary study [26], and in this work, we extend our study and provide a more comprehensive analysis. Our contributions in this paper can be summarized as follows:

- We first provide a general framework,and highlight certain design principles to efficiently employ partial recovery in a coded computation scenario, particularly with MMC.
- Based on design principles, we introduce a general code structure, called *random circularly shifted (RCS) codes*, for distributed coded SGD.
- Through numerical experiments, we show that RCSC codes outperform existing schemes, and also present the trade-offs between the update accuracy, communication latency and computation time achieved by RCS codes.

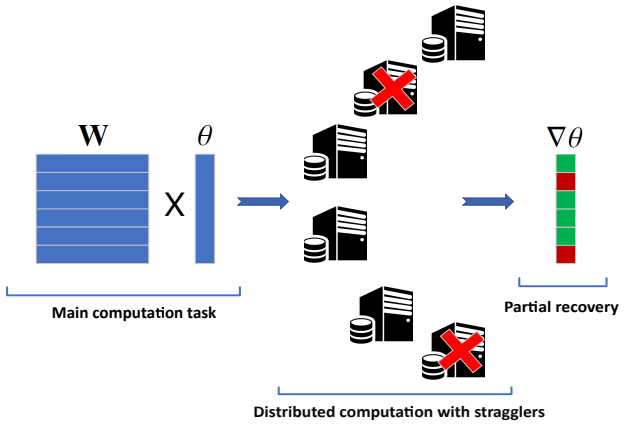

Fig. 1: Illustration of partial recovery in a naive distributed computation scenario with 6 workers, 2 of which are stragglers.

Cumulative computation type	MCC $n_m(\mathbf{N}_i)$	UC-MMC $n_u(\mathbf{N}_i)$	CCPR $n_c(\mathbf{N}_i)$
$\mathbf{N}_1: N_2=4, N_1=0, N_0=0$	1	1	1
$\mathbf{N}_2: N_2=3, N_1=1, N_0=0$	4	4	4
$\mathbf{N}_3: N_2=3, N_1=0, N_0=1$	4	4	4
$\mathbf{N}_4: N_2=2, N_1=2, N_0=0$	6	6	6
$\mathbf{N}_5: N_2=2, N_1=1, N_0=0$	12	8	12
$\mathbf{N}_6: N_2=2, N_1=0, N_0=2$	6	2	6
$\mathbf{N}_7: N_2=1, N_1=3, N_0=0$	0	4	4
$\mathbf{N}_8: N_2=1, N_1=2, N_0=1$	0	4	8
$\mathbf{N}_9: N_2=0, N_1=4, N_0=1$	0	1	1

TABLE I: Number of score vectors for each cumulative computation that can result in full gradient computation with $K=4$ and $r=2$.

- Finally, we show how RCS codes can be employed for coded communication scenario as well, and we also highlight the possible future extensions to enhance the convergence performance of partial recovery approach further.

II. Coded computation with partial recovery

In conventional coded computation schemes, the PS waits until a sufficient number of computations are gathered from the workers to recover the full gradient. In contrast, the partial recovery strategy does not necessarily aim at recovering the full gradient, bu instead terminates an iteration when the received computations allow a sufficiently accurate gradient estimate for a model update. We will explain below how the accuracy of the computation can be evaluated.

If one considers the linear regression problem, where the main task is a matrix vector multiplication, $\mathbf{W}\boldsymbol{\theta}$, missing computations lead to erasures in the gradient vector as illustrated in Fig. 1. We remark again that partial recovery can be achieved trivially with uncoded computation as in Fig. 1; however, coded computation approach provides the PS a certain control over the number of erasures. Hence, the main notion behind the coded computation with partial recovery scheme is to allow erasures in a controlled way without increasing the significantly.

For the encoding part, we utilize a general linear code structure. Matrix $\mathbf{W}$ is initially divided into K disjoint submatricies $\mathbf{W}_1,\dots,\mathbf{W}_K \in \mathbb{R}^{d/K\times d}$. Then, r coded submatricies, $\tilde{\mathbf{W}}_{i,1},\dots,\tilde{\mathbf{W}}_{i,r}$, are assigned to each worker i for computation, where each coded matrix $\tilde{\mathbf{W}}_{i,j}$ is a linear combination of K submatricies, i.e.,

$$\tilde{\mathbf{W}}_{i,j} = \sum_{k\in[K]} \alpha^{(i)}_{j,k}\mathbf{W}_k. \tag{4}$$

Following the initial encoding phase, at each iteration t, the ith worker performs the computations $\tilde{\mathbf{W}}_{i,1}\theta_t,\dots,\tilde{\mathbf{W}}_{i,r}\theta_t$ in the given order and sends the result as soon as the corresponding computation is completed. We remark that, under multi-message communication scenario, the order of the assigned coded computations also affects the iteration time, therefore we use computation assignment matrix $\mathbf{C}$ to represent a coded computation strategy i.e,

$$\mathbf{C} = \begin{bmatrix} \tilde{\mathbf{W}}_{1,1} & \tilde{\mathbf{W}}_{1,2} & \dots & \tilde{\mathbf{W}}_{1,K} \\ \tilde{\mathbf{W}}_{2,1} & \tilde{\mathbf{W}}_{2,2} & \dots & \tilde{\mathbf{W}}_{2,K} \\ \vdots & \vdots & \dots & \vdots \\ \tilde{\mathbf{W}}_{r,1} & \tilde{\mathbf{W}}_{r,2} & \dots & \tilde{\mathbf{W}}_{r,K} \end{bmatrix}.$$

When the partial recovery is allowed, PS waits until $(1-q)\times 100$ percent of the gradient entries are successfully recovered. We call the parameter q as the tolerance rate, which is a design parameter.

In the scope of this paper, our aim is to highlight certain design principles to form coded submatrices $\tilde{\mathbf{W}}_{i,1},\dots,\tilde{\mathbf{W}}_{i,r}$, for each user i, in order to allow partial recovery with reduced iteration time. Before introducing these design principles and the proposed strategy for the encoding phase, we present a simple example to show how coded computation with partial recovery can improve upon other schemes, such as MDS coding or uncoded computation with MMC (UC-MMC).

A. Motivating example

Consider $K=4$ workers and assume that $\mathbf{W}$ is divided into 4 submatrices $\mathbf{W}_1,\dots,\mathbf{W}_4$. Let us first consider two known distributed computation schemes, namely UC-MM [3], [19] and MDS-coded computation (MCC) [15].

Recall, $\mathbf{C}$, shows the assigned computation tasks to each worker with their execution order. More specifically, $\mathbf{C}(i,j)$ denotes the ith computation task to be executed by the jth worker. In MDS-coded computation, linearly independent coded computation tasks are distributed to the workers as follows:

$$\mathbf{C}_{MDS} = \begin{bmatrix} \mathbf{W}_1+\mathbf{W}_3 & \mathbf{W}_1+2\mathbf{W}_3 & \mathbf{W}_1+4\mathbf{W}_3 & \mathbf{W}_1+8\mathbf{W}_3 \\ \mathbf{W}_2+\mathbf{W}_4 & \mathbf{W}_2+2\mathbf{W}_4 & \mathbf{W}_2+4\mathbf{W}_4 & \mathbf{W}_2+8\mathbf{W}_4 \end{bmatrix}.$$

Each worker sends the results of its computations only after all of them are completed, e.g., first worker sends the concatenation of $[(\mathbf{W}_1+\mathbf{W}_3)\boldsymbol{\theta}_t \quad (\mathbf{W}_2+\mathbf{W}_4)\boldsymbol{\theta}_t]$ after completing both computations; therefore, any permutations of each column vector would result in the same performance. $\mathbf{C}_{MDS}$ corresponds to a $(2,4)$ MDS code; hence, the PS can recover the full gradient from the results of any two workers. In the UC-MMC scheme with cyclic shifted computation assignment [19], computation scheduling matrix is given by

$$\mathbf{C}_{UC-MMC} = \begin{bmatrix} \mathbf{W}_1 & \mathbf{W}_2 & \mathbf{W}_3 & \mathbf{W}_4 \\ \mathbf{W}_2 & \mathbf{W}_3 & \mathbf{W}_4 & \mathbf{W}_1 \end{bmatrix},$$

and each worker sends the results of its computations sequentially, as soon as each of them is completed. This helps to reduce the per-iteration completion time with an increase in the communication load [3], [19]. With UC-MMC, full gradient can be recovered even if each worker performs only one computation, which is faster if the workers have similar speeds.

The computation scheduling matrix of the proposed CCPR is given by

$$\mathbf{C}_{CCPR} = \begin{bmatrix} \mathbf{W}_1 & \mathbf{W}_2 & \mathbf{W}_3 & \mathbf{W}_4 \\ \mathbf{W}_3 + \mathbf{W}_4 & \mathbf{W}_1 + \mathbf{W}_3 & \mathbf{W}_2 + \mathbf{W}_4 & \mathbf{W}_1 + \mathbf{W}_2 \end{bmatrix}.$$

B. Full gradient performance

Consider a particular iteration, and let N_s denote the number of workers that have completed exactly s computations by time t, $s = 0, \dots, r$. We define $\mathbf{N} \triangleq (N_r, \dots, N_0)$ as the *cumulative computation type*. Additionally, we introduce the K-dimensional *score vector* $\mathbf{S} = [s_1, \dots, s_K]$, where s_i denotes the number of computations completed and communicated by the ith worker. We note that, due to the homogeneous worker assumption, the probability of experiencing any score vector with the same cumulative computation type is the same. Therefore, what is important for the overall computation time statistics is the number of score vectors corresponding to each computation type.

Let $n_m(\mathbf{N}), n_u(\mathbf{N})$ and $n_c(\mathbf{N})$ denote the number of distinct score vectors with the cumulative computation type $\mathbf{N}$ that allow the recovery of full gradient for the MDS-coded, UC-MMC, and CCPR schemes, respectively. For instance, given cumulative computation type $\mathbf{N} = (1, 2, 0)$, MDS-coded scheme can not recover the full gradient, however UC-MMC can recover the full gradient for four possible $\mathbf{S}$; $\mathbf{S} = [2, 0, 1, 1]$, $\mathbf{S} = [1, 2, 0, 1]$, $\mathbf{S} = [1, 1, 2, 0]$ and $\mathbf{S} = [0, 1, 1, 2]$, hence $n_u(\mathbf{N}) = 4$. Finally, in CCPR scheme, there are in total 8 different $\mathbf{S}$ achieving full recovery; $\mathbf{S} = [2, 1, 0, 1]$, $\mathbf{S} = [2, 1, 1, 0]$, $\mathbf{S} = [1, 2, 1, 0]$, $\mathbf{S} = [0, 2, 1, 1]$ $\mathbf{S} = [1, 0, 2, 1]$, $\mathbf{S} = [0, 1, 2, 1]$, $\mathbf{S} = [1, 1, 0, 2]$ and $\mathbf{S} = [1, 0, 1, 2]$. These values are listed in Table I for the cumulative computation types that can in full gradient recovery for at least one of the schemes. Particularly striking are the last three rows that correspond to cases with very few computations completed, i.e., when at most one worker completes all its assigned tasks. In these cases, CCPR is much more likely to allow full gradient computation; and hence, the computation deadline can be reduced significantly while still recovering the full gradient. For a more explicit comparison of the completion time statistics, we can analyze the probability of each type under a specific computation time statistics. Then, the probability of cumulative computation type $\mathbf{N}(t)$ at time t is given by $\Pr(\mathbf{N}(t)) = \prod_{s=0}^{r} P_s(t)^{N_s}$. Let T denote the full gradient recovery time. Accordingly, for the example above, $\Pr(T < t)$ for CCPR is given by $\sum_{i=1}^{9} n_c(\mathbf{N}_i) \cdot \Pr(\mathbf{N}_i(t))$, where the types $\mathbf{N}_i$ and corresponding $n_c(\mathbf{N}_i)$, $i = 1, \dots, 9$, are listed in Table I. $\Pr(T < t)$ for MCC and UC-MMC can be written similarly by replacing $n_c(\mathbf{N}_i)$ with $n_m(\mathbf{N}_i)$ and $n_u(\mathbf{N}_i)$, respectively. It is now clear that CCPR has the highest $\Pr(T < t)$ for any t; and hence, the minimum average per-iteration completion time $E[T]$. In the next subsection, we will highlight the partial recovery property of CCPR.

Cumulative computation type	MCC $n_m(i)$	UC-MM $n_u(i)$	CCPR $n_c(i)$
$\mathbf{N}_1 : N_2 = 4, N_1 = 0, N_0 = 0$	1	1	1
$\mathbf{N}_2 : N_2 = 3, N_1 = 1, N_0 = 0$	4	4	4
$\mathbf{N}_3 : N_2 = 3, N_1 = 0, N_0 = 1$	4	4	4
$\mathbf{N}_4 : N_2 = 2, N_1 = 2, N_0 = 0$	6	6	6
$\mathbf{N}_5 : N_2 = 2, N_1 = 1, N_0 = 1$	12	12	12
$\mathbf{N}_6 : N_2 = 2, N_1 = 0, N_0 = 2$	6	6	6
$\mathbf{N}_7 : N_2 = 1, N_1 = 3, N_0 = 0$	0	4	4
$\mathbf{N}_8 : N_2 = 1, N_1 = 2, N_0 = 1$	0	12	12
$\mathbf{N}_9 : N_2 = 1, N_1 = 1, N_0 = 2$	0	8	8
$\mathbf{N}_{10} : N_2 = 0, N_1 = 4, N_0 = 0$	0	1	1
$\mathbf{N}_{11} : N_2 = 0, N_1 = 3, N_0 = 1$	0	4	4

TABLE II: Number of score vectors for each cumulative computation type that can result in the recovery of at least 3 out of 4 computations with $K = 4$ and $r = 2$.

C. Partial gradient performance

GD methods can be effective even with less accurate model updates. Indeed, SGD is employed in many large-scale machine learning applications to speed up the convergence. Similarly, in a distributed implementation of GD, sufficient accuracy may be achieved by updating the parameter vector using a subset of the partial computations assigned to the workers at each iteration, assuming that the straggling workers vary over iterations. Hence, based on the learning problem required update accuracy at each iteration can be adjusted.

However, as we have stated earlier, most of the coding schemes target full gradient recovery, For each cumulative computation type the number of score vectors that can result in the recovery of at least 3 out of 4 computations are given in Table II. Note that when the recovery of 75% of the gradient indices are sufficient to complete an iteration UC-MMC and CCPR have the same average completion time statistics. Hence, CCPR can provide a lower average per-iteration completion time for full gradient computation compared to UC-MMC, while achieving the same performance when partial gradients are allowed.

III. Design Principles of CCPR

For the encoding of the assigned computations we use a similar strategy to *rateless codes*, particularly to LT codes [44]. Before, introducing the design principles of CCPR, we want to briefly explain the LT code structure and underline the required modification for our problem setup. Lets consider a set of symbols $\mathcal{S}$ (in our setup these symbols corresponds to submatricies $\mathbf{W}_i$) to be used to form codewords (coded computation) and transmitted to a destination node which wants to recover all symbols. In the encoding phase, a codeword is formed by choosing d symbols randomly from $\mathcal{S}$ and bit-wise XORing them, where d, which simply defines the degree of the codeword, comes from a distribution $P(d)$. In the decoding part, each codeword is decomposed by using recovered symbols, that is if a codeword contains a previously recovered symbol than the symbol is subtracted from the codeword to obtain a new codeword with smaller degree. In overall the objective is to recover $m = |\mathcal{S}|$ symbols from

$m(1+\epsilon)$ codewords with as possible as small ϵ which reflects the overhead.

We remark that codewords with smaller degrees can be decomposed faster; however, having many codewords with smaller degrees increases the probability of linear dependence among codewords. Hence, degree distribution, as known a soliton distribution, plays an important role in the performance of LT codes, and the main challenge is to find the optimal degree distribution $P(d)$. It has been show that for a carefully chosen $P(d)$, ϵ goes to zero as $m \to \infty$. From the distributed computation perspective, use of LT codes in distributed computation scenario have the following drawbacks.

First, LT code structure is designed under the assumption of a receiving large number of coded messages; however, in a distributed computation scenario the number of messages transmitted should be limited due to congestion. Therefore, for small m the overhead ϵ might be high. Hence, when m is small, random codeword generation by choosing the message degrees randomly may not be practical and a more systematic codeword generation might perform better.

Second, soliton distribution of the LT codes, $P(d)$ are designed for the full recovery. However, in our partial coded computation scenario we want to provide a certain flexibility to PS so that an iteration can be terminated when $m(1-q)$ symbols, for some predefined tolerance rate q, are recovered. Although, the partial recovery of the rateless codes has been studied in the literature [45], we note that partial recovery for coded computation requires a tailored approach since the computational tasks, each of which corresponds to a distinct codeword, are executed sequentially; thus, erasure probabilities of codewords, due to straggling behavior of workers, are neither identical nor independent. Therefore, codewords must be designed taking into account their execution orders to prevent overlaps and to minimize the average completion time.

Hence, the main notion behind the CCPR scheme, is to utilize rateless code structure in a more systematic way such that instead of using a random degree d from soliton distribution, the degree of the each codeword is chosen carefully based on its computation order while also taking into account the partial recovery.

A. Computation order and degree limitation

We want to emphasize that, codewords with lower degrees can be recovered faster but as the number of received codewords increases at the PS the codewords with lower degrees become less and less informative, therefore we want later codewords to be more informative while we want first arriving ones to be easily recoverable. Hence, we induce the following design criterias; first, for the first row of the computation assignment matrix, we consider uncoded computation, second for a particular worker and given computation orders $i, j < r$, the degree of the codeword with order i can not be higher than the one with order j.

B. Uniformity imposed encoding

As highlighted before, coded messages with lower degrees may result in duplicate recoveries, wasting the computation resources. To this end, under degree limitation, the main design issue is how to form the coded messages in order to prevent duplicate messages as much as possible. Accordingly, the challenge is to distribute submatrices $\mathbf{W}_1, \dots, \mathbf{W}_K$ among the codewords in a uniform fashion.

1) Order-wise uniformity: By order-wise uniformity, we impose a constraint on the code construction such that computations with the same order must have the same degree, and among the codewords with the same computation order, each submatrix $\mathbf{W}_k$ must appear in exactly the same number of codewords. Formally speaking, let

$$\tilde{\mathcal{W}}_k^j = \left\{ \tilde{\mathbf{W}}_{i,j} : \alpha_{j,k}^i \neq 0,\ i \in [K],\ k \in [K] \right\} \tag{5}$$

be the set of coded messages at computation order $j \in [r]$ containing submatrix $\mathbf{W}_k$. Then, the order-wise uniformity constraint

$$|\tilde{\mathcal{W}}_k^j| = m_j,\ \forall j \in [r],\ k \in [K] \tag{6}$$

2) Worker-wise uniformity: Worker-wise uniformity imposes a constraint on the coded computations assigned to each worker, such that the assigned coded do not contain any common submatrices. Formally speaking for any worker $i \in [K]$, if $\alpha_{j,k}^i \neq 0$, then $\alpha_{l,k}^i = 0$, $\forall\ l \in [r] \setminus \{j\}$.

Next, we introduce an encoding structure which ensures both order-wise and worker-wise uniformity.

IV. Randomly Circular Shifted Code Design

Algorithm 1 RCSC algorithm

Data assignment phase:
$L = \sum_{i=1}^{m} \mathbf{m}(i)$
Choose a random subset $\mathcal{I} \subset [K]$ s.t. $|\mathcal{I}| = L$
$\bar{\mathbf{W}} = [\mathbf{W}_1, \mathbf{W_2}, \dots, \mathbf{W}_K]$
for row index $i = 1, 2, \dots, L$ **do**
    Randomly choose a element $j \in \mathcal{I}$
    Update $\mathcal{I}$: $\mathcal{I} \leftarrow \mathcal{I} \setminus \{j\}$
    $A(i,:) = circshift(\mathbf{w}, j-1)$
Data encoding phase:
for worker $k = 1, 2, \dots, K$ **do**
    for message $j = 1, \dots, r$ **do**
        Starting row index: $\tilde{l} = \sum_{i=1}^{j-1} \mathbf{m}(i) + 1$
        Ending row index: $\bar{l} = \sum_{i=1}^{j} \mathbf{m}(i)$
        $c_{k,j} = \sum_{l=\tilde{l}}^{\bar{l}} A(k,l)$

In this section, we introduce the randomly circular shifted (RCS) code design for coded computation which consist of two steps; namely data assignment and code construction. Before explaining these steps in detail, we first define the degree vector $\mathbf{d}$ of length r, where its ith entry $\mathbf{d}(i)$ denotes the degree of the coded computations assigned to workers in order i, $1 \leq i \leq r$. Based on the aforementioned design criteria we set $\mathbf{d}(1) = 1$ and, for any $i < j$, we have $\mathbf{d}_i \leq \mathbf{d}_j$. Once the degree vector $\mathbf{d}$ is fixed, the two phases of RCSC design can be implemented.

In the first phase, an assignment matrix is formed by using random circular shifts on the vector of submatrices $\bar{\mathbf{W}} = [\mathbf{W}_1, \dots, \mathbf{W}_K]$. At the beginning of the first phase, an

$$\mathbf{A}_{rcsc} = \begin{bmatrix} \mathbf{W}_1 & \mathbf{W}_2 & \mathbf{W}_3 & \dots & \mathbf{W}_{20} \\ \mathbf{W}_4 & \mathbf{W}_5 & \mathbf{W}_6 & \dots & \mathbf{W}_3 \\ \mathbf{W}_{11} & \mathbf{W}_{12} & \mathbf{W}_{13} & \dots & \mathbf{W}_{10} \\ \mathbf{W}_{15} & \mathbf{W}_{16} & \mathbf{W}_{17} & \dots & \mathbf{W}_{14} \\ \mathbf{W}_6 & \mathbf{W}_7 & \mathbf{W}_8 & \dots & \mathbf{W}_5 \\ \mathbf{W}_{18} & \mathbf{W}_{19} & \mathbf{W}_{20} & \dots & \mathbf{W}_{17} \end{bmatrix}.$$

Fig. 2: Assignment matrix for $K = 20$ and $\mathcal{I} = \{1, 4, 6, 11, 15, 6, 18\}$

$$\begin{bmatrix} \mathbf{W}_1 \\ \mathbf{W}_4 \\ \mathbf{W}_{11} \\ \mathbf{W}_{15} \\ \mathbf{W}_6 \\ \mathbf{W}_{18} \end{bmatrix} \rightarrow \mathbf{C}_1 = \begin{bmatrix} \tilde{\mathbf{W}}_{1,1} \\ \tilde{\mathbf{W}}_{1,2} \\ \tilde{\mathbf{W}}_{1,3} \end{bmatrix} = \begin{bmatrix} \mathbf{W_1} \\ \mathbf{W_4 + W_{11}} \\ \mathbf{W_{15} + W_6 + W_{18}} \end{bmatrix}.$$

Fig. 3: Illustration of the encoding phase for the first worker and the corresponding first column of the computation assignment matrix, $\mathbf{C}_1$.

index set $\mathcal{I} \subset [K]$ of size $L = \sum_{i=1}^{r} \mathbf{m}(i)$ is randomly chosen. Then using the elements of $\mathcal{I}$ as a parameter of the circular shift operator on vector $\bar{\mathbf{W}} = [\mathbf{W}_1, \dots, \mathbf{W}_K]$, an assignment matrix $\mathbf{A}_{RCS}$ can be formed as illustrated in Algorithm 1 (line 5-8). Let us illustrate this on a simple example. Consider $K = 20$ workers and $\mathcal{I} = \{1, 4, 6, 11, 15, 6, 18\}$. Then, for the i row of the $\mathbf{A}_{RCS}$ a random number $j \in \mathcal{I}$ is chosen and $\mathbf{W}$ is circularly shifted by $j - 1$ and following that j discarded from $\mathcal{I}$. For sake of simplicity, we assume that numbers are chosen randomly with the given order $1, 4, 6, 11, 15, 6, 18$ and the corresponding assignment matrix is illustrated in Fig. 2,where Once the assignment matrix is fixed, codewords can be generated based on the degree vector $\mathbf{d}$ for each column independently and identically. The colors in the assignment matrix represent the submatrices that will appear in the same codeword. The code generation for the first user, using the submatrices on the first column of the assignment matrix, is illustrated in Fig. 3.

We note that each coded message corresponds to a linear equation, and in the decoding phase any approach for solving a set of linear equations can be utilized, e.g., we can form a matrix with the coefficients of the coded messages, $\alpha^i_{j,k}$, that is each coded message is represented by a binary row vector, and obtain the *reduced row echelon* form. In our design, we limit the degrees of the codewords, thus the row representation of each coded message is highly sparse, particularly the first message of each user only contains one submatrix. Besides, the number coded messages, $K \times r$, is also limited, therefore use Gaussian elimination for decoding do not induce a high complexity. However, for larger scale implementations, a successive decoding procedure using only recovered symbols, similar to the one used for LT codes, can be employed to reduce decoding complexity further. In the scope of this work, we focus on the computation time at the workers and leave the decoding schemes with reduced complexity as a future extension of this work. Nevertheless, we want to highlight that the decoding phase overlaps with the computation phase, that is to say PS does not wait for a certain number of codewords to start decoding, which may help to reduce overall latency.

V. Extension to Coded Communication Scenario

Above, we have mainly focused on coded computation in the context of a linear regression problem, where the main computation task boils down to distributed matrix-vector multiplication. In a more general learning problem, in which the computations cannot be expressed as a linear transform of the dataset, we cannot employ a similar coded computation technique. Instead, the redundancy can be achieved by assigning the same computation task to multiple workers. Communication load of such an implementation can be reduced by coded communication, where each worker sends to PS a linear combinations of its computations.

$$\mathbf{A}_{rcsc} = \begin{bmatrix} \mathbf{g}_1 & \mathbf{g}_2 & \mathbf{g}_3 & \dots & \mathbf{g}_{20} \\ \mathbf{g}_4 & \mathbf{g}_5 & \mathbf{g}_6 & \dots & \mathbf{g}_3 \\ \mathbf{g}_{11} & \mathbf{g}_{12} & \mathbf{g}_{13} & \dots & \mathbf{g}_{10} \\ \mathbf{g}_{15} & \mathbf{g}_{16} & \mathbf{g}_{17} & \dots & \mathbf{g}_{14} \\ \mathbf{g}_6 & \mathbf{g}_7 & \mathbf{g}_8 & \dots & \mathbf{g}_5 \\ \mathbf{g}_{18} & \mathbf{g}_{19} & \mathbf{g}_{20} & \dots & \mathbf{g}_{17} \end{bmatrix}.$$

Fig. 4: Assignment matrix for $K = 20$ and $\mathcal{I} = \{1, 4, 6, 11, 15, 6, 18\}$

Let $\mathcal{G} = \{\mathbf{g}_1, \dots, \mathbf{g}_K\}$ be the set of partial gradients corresponding to datasets $\mathcal{D}_1, \dots, \mathcal{D}_K$, respectively. In the GC scheme with computation load r, r partial gradient computations, denoted by $\mathcal{G}_k$, are assigned to worker k [8]. At each iteration, after computing these r partial gradients, each worker sends a linear combination of its results:

$$\mathbf{c}_k^{(t)} \triangleq \mathcal{L}_k(\mathbf{g}_i^{(t)} : \mathbf{g}_i \in \mathcal{G}_k), \tag{7}$$

where $\mathbf{g}_i^{(t)}$ denotes the partial gradient computed based on dataset $\mathcal{D}_i$ in iteration t. We refer to the linear combinations $\mathbf{c}_1, \dots, \mathbf{c}_K$ as *coded partial gradients*. The PS waits until it receives sufficiently many coded partial gradients to recover the full gradient. It is shown in [8] that, for any set of non-straggler workers $\tilde{\mathcal{K}} \subseteq [K]$ with $|\tilde{\mathcal{K}}| = K - r + 1$, there exists a set of coefficients $\mathcal{A}_{\tilde{\mathcal{K}}} = \{a_k : k \in \tilde{\mathcal{K}}\}$ such that

$$\sum_{k \in \tilde{\mathcal{K}}} a_k c_k^{(t)} = \frac{1}{K} \sum_{k=1}^{K} g_k^{(t)}. \tag{8}$$

GC scheme is designed for full gradient recovery and limits the number of messages per-user per-iteration to one. The MMC variation of the GC is studied before in [11], [13]. However, proposed RCSC schem takes into account the order of computations to allow partial recovery. We note that implementation of RCSC in coded communication almost identical to one used in coded computation with a small variation that encoding phase executed after the computation, thus, unlike coded computation, computation load both depends on the number of codewords and theirs degrees. To clarify consider the previously given example with $K = 20$, again we use an

$$\begin{bmatrix} \mathbf{g}_1 \\ \mathbf{g}_4 \\ \mathbf{g}_{11} \\ \mathbf{g}_{15} \\ \mathbf{g}_6 \\ \mathbf{g}_{18} \end{bmatrix} \rightarrow \begin{bmatrix} \tilde{\mathbf{g}}_{1,1} \\ \tilde{\mathbf{g}}_{1,2} \\ \tilde{\mathbf{g}}_{1,3} \end{bmatrix} = \begin{bmatrix} \mathbf{g}_1 \\ \mathbf{g}_4 + \mathbf{g}_{11} \\ \mathbf{g}_{15} + \mathbf{g}_6 + \mathbf{g}_{18} \end{bmatrix}.$$

Fig. 5: Illustration of the encoding phase (coded communication) for the first worker.

assignment matrix of $L \times K$ to assign partial gradient, with a certain order, to each worker to compute as illustrated in Fig. 4.

We want to highlight that in the coded communication scenario encoding takes place after the computation, therefore the computational load $r = L = 6$ in this example, unlike the coded computation scenario where $r = 3$. Again, once the assignment matrix is formed, coded messages for each worker are constructed according to given order vector $\mathbf{m}$ and based on the assignment matrix $\mathbf{A}_{RCS}$ as illustrated in Fig. 5.

VI. Generalized RCS codes

In the introduced RCS code structure, the main computation task is divided into K equal sub-tasks. However, if the variation on the computational speed of the workers are small, it might be better to divide main task into much smaller tasks in order to better utilizes the computational resources []. Here, we remark that although the introduced RCS code structure is based dividing the main task into K equal size subtask, it can be easily extended to more general cases.

Lets consider the coded computation scenario for the generalized RCS codes. We remark that in generalized RCS codes, the encoding part will remain same but we will change the construction procedure of $\mathbf{A}_{RCS}$. First, $\mathbf{W}$ is divided into KN disjoint submatrices, i.e., $\mathbf{W}_1, \ldots, \mathbf{W}_{KN}$ and then these summatrices are divided into N groups $\mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(N)}$each containing K submatrices, i.e., $\mathbf{W}^{(i)} = [\mathbf{W}_{(i-1)K+1}, \ldots, \mathbf{W}_{iK}]$. Then for each group $\mathbf{W}^{(i)}$, an assignment matrix $\mathbf{A}^{(i)}$ formed, again, using circular shifts based on a random set $\mathcal{I}_i$. Once assignment matrices $\mathbf{A}^{(i)}, \ldots \mathbf{A}^{(N)}$ are formed, $\mathbf{A}_{RCS}$ can be constructed by merging the matrices $\mathbf{A}^{(i)}, \ldots \mathbf{A}^{(N)}$ according to a given vector $\mathbf{z}$ which basically illustrates how the row of the submatrices are concatenated, i.e., if $\mathbf{z}(l) = i$, then lth row of $\mathbf{A}_{RCS}$ is taken from $\mathbf{A}^{(i)}$. Next, we introduce a simple example for $K = 4$ and $N = 2$ to clarify the overall procedure. For the given setup, we consider $\mathcal{I}_1 = \{1, 3\}$, $\mathcal{I}_2 = \{1, 4, 3\}$, and $\mathbf{z} = [1, 2, 2, 1, 1]$, and the construction procedure of $\mathbf{A}_{RCS}$ is illustrated in Fig. 6. Furthermore, we also illustrate the computation assignment matrix $\mathbf{C}_{RCS}$ in Fig. 7 for given $\mathbf{d} = [1, 1, 3]$. We remark here that as in previous example illustrated in Fig. 3, each user is allowed to send at most 3 messages but now the computation load $r = 3/2$ instead of 3. Hence, if the computation speed of the workers are close it might be better to divide main task into more than K subtasks to utilize computation resources better. Here, we remark that first assigned computations, in uncoded form, are from $\mathbf{W}^{(2)}$ and second assigned computations, in uncoded form, are from $\mathbf{W}^{(1)}$. Hence, considering only first two assigned computations recovery probability of $\mathbf{W}_i\theta$, $\mathbf{W}_i \in \mathbf{W}^{(2)}$, is higher compared to recovery probability of $\mathbf{W}_j\theta$, $\mathbf{W}_j \in \mathbf{W}^{(1)}$. Therefore, in the third codeword two submatrices are taken from $\mathbf{W}^{(2)}$ and one submatrix is taken from $\mathbf{W}^{(1)}$. Consequently, by playing $\mathbf{d} = [1, 1, 3]$ and $\mathbf{z} = [1, 2, 2, 1, 1]$ different operating points can be achieved.

VII. Numerical Results and Discussions

For the numerical analysis, we will first analyze the convergence performance of the partial recovery strategy for coded computation with RCSC under different tolerance requirements, then we will compare the *average per-iteration completion time* of the RCSC scheme with UC-MM and MDS coded compuation (MCC)scheme and, finally we extend our analysis to coded communication setup.

A. Simulation Setup

For the statistics of computation speed, we adopt the model in [15], where the probability of completing exactly s computations by time t, $P_s(t)$, is given by

$$P_s(t) = \begin{cases} 0, & \text{if } t < s\alpha, \\ 1 - e^{-\mu(\frac{t}{s} - \alpha)}, & s\alpha \leq t < (s+1)\alpha, \\ e^{-\mu(\frac{t}{s+1} - \alpha)} - e^{-\mu(\frac{t}{s} - \alpha)} & (s+1)\alpha < t, \end{cases} \quad (9)$$

where α is the minimum required time to finish a computation task, and μ is the average number of computations completed in unit time. For the simulations, we consider a linear regression problem over synthetically created training and test datasets, as in [18], of size of 2000 and 400, respectively. We also assume that the size of the model $d = 1000$ and the number of workers $K = 40$. Finally, we set $\mu = 10$ and $\alpha = 0.01$ for the statistics of computation speed in (9). For the RCSC scheme we choose the degree vector $\mathbf{m} = [1, 2, 3]$, hence $r = 3$, and we execute the Algorithm 1 accordingly. For all simulations, we use learning rate $\lambda = 0.1$.

B. Simulation Results

Once we finalize the simulation setup, we consider three different subscenarios, each corresponding to a different tolerance requirement $q = 0$ (which corresponds to full recovery), $q = 0.15$, $q = 0.3$ and plot the test error over $T = 100$ iterations[1] in Fig. 8. One can observe that, although convergence speed reduces with increasing tolerance rate, partial recovery does not harm the convergence behaviour much especially if the tolerance level is moderate, e.g., $q = 0.15$. We then repeat the same experiments for the model size $d = 400$, but now for $T = 80$ iterations, which is illustrated in Fig. 9 and demonstrates similar trends. Up to now, we show how the tolerance rate affects the convergence performance with respected to number of iterations, however yet another question we want to answer is how much reduction on per-iteration time can partial recovery scheme achieve.

[1]For the convergence plot, we take average over 10 independent simulations.

$$\mathbf{A}^{(1)} = \begin{bmatrix} \mathbf{W}_1 & \mathbf{W}_2 & \mathbf{W}_3 & \mathbf{W}_4 \\ \mathbf{W}_3 & \mathbf{W}_4 & \mathbf{W}_1 & \mathbf{W}_2 \end{bmatrix} \quad \mathbf{A}^{(2)} = \begin{bmatrix} \mathbf{W}_5 & \mathbf{W}_6 & \mathbf{W}_7 & \mathbf{W}_8 \\ \mathbf{W}_8 & \mathbf{W}_5 & \mathbf{W}_6 & \mathbf{W}_7 \\ \mathbf{W}_7 & \mathbf{W}_8 & \mathbf{W}_5 & \mathbf{W}_6 \end{bmatrix} \quad \mathbf{A}_{RCS} = \begin{bmatrix} \mathbf{W}_5 & \mathbf{W}_6 & \mathbf{W}_7 & \mathbf{W}_8 \\ \mathbf{W}_1 & \mathbf{W}_2 & \mathbf{W}_3 & \mathbf{W}_4 \\ \mathbf{W}_3 & \mathbf{W}_4 & \mathbf{W}_1 & \mathbf{W}_2 \\ \mathbf{W}_8 & \mathbf{W}_5 & \mathbf{W}_6 & \mathbf{W}_7 \\ \mathbf{W}_7 & \mathbf{W}_8 & \mathbf{W}_5 & \mathbf{W}_6 \end{bmatrix}$$

Fig. 6: Assignment matrices for $\mathbf{W}_1$ and $\mathbf{W}_2$ according to $\mathcal{I}_1 = \{1, 3\}$ and $\mathcal{I}_2 = \{1, 4, 3\}$, respectively, and $\mathbf{A}_{RCS}$ based on $\mathbf{A}^{(1)}$, $\mathbf{A}^{(1)}$, and $\mathbf{z} = [1, 2, 2, 1, 1]$.

$$\mathbf{C}_{RCS} = \begin{bmatrix} \mathbf{W}_5 & \mathbf{W}_6 & \mathbf{W}_7 & \mathbf{W}_8 \\ \mathbf{W}_1 & \mathbf{W}_2 & \mathbf{W}_3 & \mathbf{W}_4 \\ \mathbf{W}_3 + \mathbf{W}_8 + \mathbf{W}_7 & \mathbf{W}_4 + \mathbf{W}_5 + \mathbf{W}_8 & \mathbf{W}_1 + \mathbf{W}_6 + \mathbf{W}_5 & \mathbf{W}_2 + \mathbf{W}_7 + \mathbf{W}_6 \end{bmatrix}.$$

Fig. 7: $\mathbf{C}_{RCS}$ based on $\mathbf{A}_{RCS}$ and $\mathbf{d} = [1, 1, 3]$

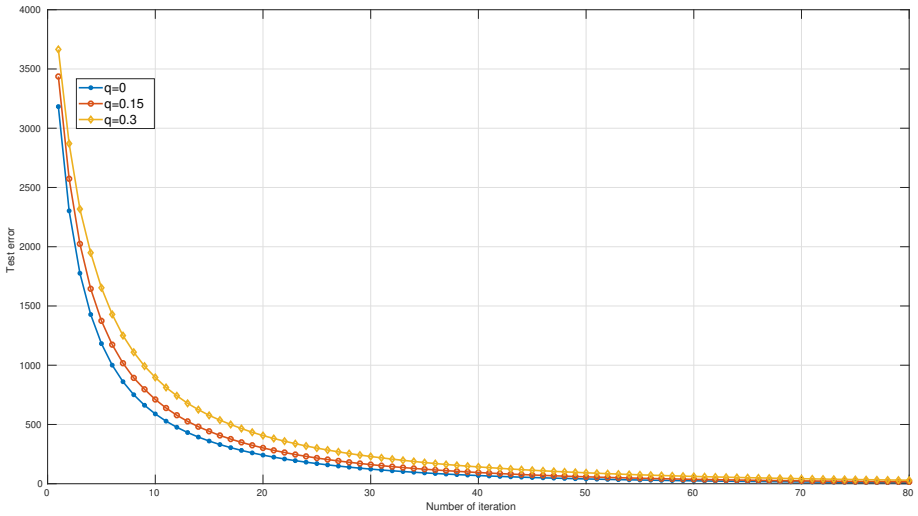

Fig. 8: Test error over $T = 80$ iterations, given model size $d = 1000$, with respect to tolerance rate $q = 0, 0.15, 0.3$, respectively.

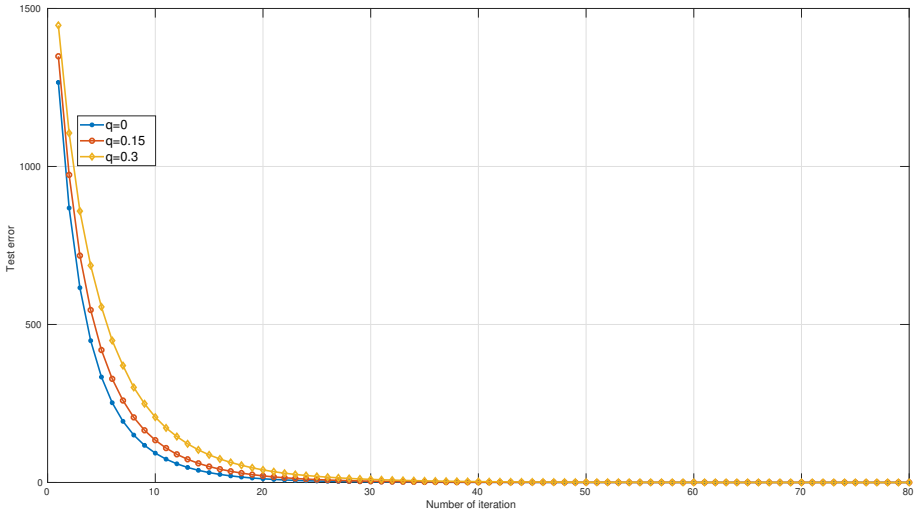

Fig. 9: Test error over $T = 80$ iterations, given model size $d = 400$, with respect to tolerance rate $q = 0, 0.15, 0.3$, respectively

Computation strategy / tolerance rate	RCSC			UC-MM			MCC
	$q = 0.0$	$q = 0.15$	$q = 0.3$	$q = 0$	$q = 0.15$	$q = 0.3$	
Average iteration time	0.1582	0.0935	0.0755	0.2424	0.1170	0.0799	0.1572
Number of received messages	63.94	42.80	34.51	81.29	51.16	36.70	14

TABLE III: Comparison of the proposed RCSC scheme with UC-MM and MCC schemes for the computational load $r = 3$

Hence, we also compare the per-iteration time of three different schemes namely RCSC, UC-MM and MCC for the computational load $r = 3$. For RCSC, we use the order vector $\mathbf{m} = [1, 2, 3]$, for UC-MM we use cyclic shifted assignment [19] and finally, for MCC we use $(\lceil K/r \rceil, K) = (14, 40)$ MDS code.

We compare the three schemes under the two performance metrics *average completion time* and *number of received message* which demonstrate how fast an iteration completed and induced communication load, respectively. From table III, one can observe that for the full recovery, i.e., $q = 0$, RSCS achieves approximately the same minimum average per-iteration time with MCC and outperforms the UC-MM scheme. We also observe that with allowing partial recovery it is possible to achieve approximately 40% to 50% reduction on the per iteration completion time with $q = 0.15$ and $q = 0.30$, respectively. Here, we note that partial recovery approach can be also employed in UC-MM scheme, however, as demonstrated in Table III, RCSC outperforms UC-MM for all given tolerance rates. Besides, with RCSC scheme PS completes an iteration with less number of received messages which means compared UC-MM RCSC induces less communication load and congestion, for instance when $q = 0$ UC-MM requires on the average 28% more messages to complete an iteration compared to proposed RCSC scheme, which is shown to be also critical on the performance of the real implementations [43]. Here, we note that MCC requires the minimum number of messages to complete an iteration. Hence, for $q = 0$ MCC might be a better alternative, nevertheless yet another advantage of RCSC on MCC is that ,unlike the MCC, decoding process is executed in parallel to the computation. Finally, taking into account all simulation results, we conclude that RCSC scheme operate most efficiently when the partial recovery is aimed with low tolerance rate, e.g., $q = 0.15$, since in this regime we can see the advantage of both using coded computation and partial recovery. To clarify this point, we observe that increasing $q = 0.15$ to $q = 0.3$ makes considerable impact on the accuracy plot but reduction on the per-iteration time is relatively small. Besides, as q increases performance of the UC-MM scheme gets closer to the performance RCSC.

Although, RCSC is initially designed for coded computation can be implemented for coded communication as well. Therefore, we repeat our experiments to compare performance of RCSC, UC-MM and GC for the computational load $r = 6$. For the RCSC we again use the degree vector $\mathbf{m} = [1, 2, 3]$. The simulation results are illustrated in Table IV. We observe that in terms of average per-iteration completion time UC-MM and RCSC outperforms the GC and UC-MM even performs slightly better than RCSC. On the other hand, in terms of the communication load UC-MM has the worst performance. Hence, the key advantage of RCSC on coded communication scenario is to seek a balance between the computation time and the communication time. At this point, we also want to highlight that UC-MM can be considered as a special case of the RCSC where the degree of the all message are one. In overall, one can play with the degree vector to seek a difference trade-off between communication and computation latency.

C. Discussions

In the scope of this paper, our aim is to design a new code construction strategy for existing straggler-aware coded computation/communication schemes in order to enable them to seek a certain balance between the update accuracy and per-iteration computation time and the communication load. However, we believe that our approach brings out new research problems that targeting further enhancement on the learning procedure. Here, we briefly explain some of possible future extension for the completeness of the paper. Also for reproducibility of our simulation results and for ease of extension of our scheme we make the code available [46].

1) Double threshold scheme: One of the first possible immediate extension is to use two thresholds to decide when to terminate an iteration instead of using only one which is the tolerance rate. In the implementation of the double threshold scheme, after sending the latest model, PS starts keeping tine and collects messages from workers until for a given fixed duration. Once the duration is completed, PS checks whether the requirement due to tolerance rate is satisfied or not and if it is not satisfied then continues to receive messages from workers until it is satisfied.

2) Adaptive tolerance rate: As discussed in several papers, the update accuracy have different impacts on the convergence in different phases of the training. Hence, the tolerance rate can be adjusted over the time to obtain a better convergence result.

3) Memory enhanced updates: In our simulations, at each iteration we use only the currently recovered computation results, however, it might be possible to utilize the computation results from previous iterations to compensate the missing computation results of the current iteration similar to the *momentum SGD* framework.

VIII. Conclusions

We recall that the purpose of CCPR approach and RCSC design is to provide flexibility in seeking a balance between the per-iteration completion time, the update accuracy and the communication load. By conducting different experiments using different tolerance rates we show that RCSC can help to reduce per-iteration completion time with a reasonable reduction on the update accuracy which can be tolerated over iterations. We also show that compared to UC-MM, which can

Computation strategy / tolerance rate	RCSC			UC-MM			GC
	$q = 0.0$	$q = 0.15$	$q = 0.3$	$q = 0$	$q = 0.15$	$q = 0.3$	
Average iteration time	0.2219	0.1231	0.0940	0.1874	0.0986	0.0736	1.2575
Number of received messages	62.56	41.55	32.37	99.63	55.06	38.30	35

TABLE IV: Comparison of the proposed RCSC scheme with UC-MM and GC schemes for the computational load $r = 6$

also employ partial recovery, RCSC requires, on the average, less number of messages to complete an iteration which mean lower communication load. We also briefly summarize possible directions for future works which can enhance the convergence performance further.

REFERENCES

[1] S. Li, S. M. M. Kalan, A. S. Avestimehr, and M. Soltanolkotabi, "Near-optimal straggler mitigation for distributed gradient methods," in *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, May 2018, pp. 857–866.

[2] N. Ferdinand and S. C. Draper, "Anytime stochastic gradient descent: A time to hear from all the workers," in *2018 56th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, Oct 2018, pp. 552–559.

[3] M. Mohammadi Amiri and D. Gündüz, "Computation scheduling for distributed machine learning with straggling workers," *IEEE Transactions on Signal Processing*, vol. 67, no. 24, pp. 6270–6284, Dec 2019.

[4] A. Behrouzi-Far and E. Soljanin, "On the effect of task-to-worker assignment in distributed computing systems with stragglers," in *2018 56th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, Oct 2018, pp. 560–566.

[5] J. Chen, R. Monga, S. Bengio, and R. Józefowicz, "Revisiting distributed synchronous SGD," *CoRR*, vol. abs/1604.00981, 2016. [Online]. Available: http://arxiv.org/abs/1604.00981

[6] M. F. Aktas and E. Soljanin, "Straggler mitigation at scale," *CoRR*, vol. abs/1906.10664, 2019. [Online]. Available: http://arxiv.org/abs/1906.10664

[7] D. Wang, G. Joshi, and G. W. Wornell, "Efficient straggler replication in large-scale parallel computing," *ACM Trans. Model. Perform. Eval. Comput. Syst.*, vol. 4, no. 2, pp. 7:1–7:23, Apr. 2019. [Online]. Available: http://doi.acm.org/10.1145/3310336

[8] 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*, ser. Proceedings of Machine Learning Research, D. Precup and Y. W. Teh, Eds., vol. 70. International Convention Centre, Sydney, Australia: PMLR, 06–11 Aug 2017, pp. 3368–3376.

[9] M. Ye and E. Abbe, "Communication-computation efficient gradient coding," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. Stockholmsmässan, Stockholm Sweden: PMLR, 10–15 Jul 2018, pp. 5610–5619.

[10] W. Halbawi, N. Azizan, F. Salehi, and B. Hassibi, "Improving distributed gradient descent using reed-solomon codes," in *2018 IEEE Int. Symp. on Inf. Theory (ISIT)*, June 2018, pp. 2027–2031.

[11] E. Ozfatura, D. Gündüz, and S. Ulukus, "Gradient coding with clustering and multi-message communication," in *2019 IEEE Data Science Workshop (DSW)*, June 2019, pp. 42–46.

[12] S. Sasi, V. Lalitha, V. Aggarwal, and B. S. Rajan, "Straggler mitigation with tiered gradient codes," 2019.

[13] L. Tauz and L. Dolecek, "Multi-message gradient coding for utilizing non-persistent stragglers," in *2019 53rd Asilomar Conference on Signals, Systems, and Computers*, 2019, pp. 2154–2159.

[14] N. Charalambides, M. Pilanci, and A. O. Hero, "Weighted gradient coding with leverage score sampling," in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020, pp. 5215–5219.

[15] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," *IEEE Trans. Inf. Theory*, vol. 64, no. 3, pp. 1514–1529, March 2018.

[16] N. Ferdinand and S. C. Draper, "Hierarchical coded computation," in *2018 IEEE Int. Symp. Inf. Theory (ISIT)*, June 2018, pp. 1620–1624.

[17] R. K. Maity, A. Singh Rawa, and A. Mazumdar, "Robust gradient descent via moment encoding and ldpc codes," in *2019 IEEE International Symposium on Information Theory (ISIT)*, July 2019, pp. 2734–2738.

[18] S. Li, S. M. M. Kalan, Q. Yu, M. Soltanolkotabi, and A. S. Avestimehr, "Polynomially coded regression: Optimal straggler mitigation via data encoding," *CoRR*, vol. abs/1805.09934, 2018.

[19] E. Ozfatura, D. Gündüz, and S. Ulukus, "Speeding up distributed gradient descent by utilizing non-persistent stragglers," in *2019 IEEE International Symposium on Information Theory (ISIT)*, July 2019, pp. 2729–2733.

[20] S. Dutta, M. Fahim, F. Haddadpour, H. Jeong, V. Cadambe, and P. Grover, "On the optimal recovery threshold of coded matrix multiplication," *IEEE Transactions on Information Theory*, pp. 1–1, 2019.

[21] Q. Yu, M. Maddah-Ali, and S. Avestimehr, "Polynomial codes: an optimal design for high-dimensional coded matrix multiplication," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 4403–4413.

[22] H. Park, K. Lee, J. Sohn, C. Suh, and J. Moon, "Hierarchical coding for distributed computing," in *2018 IEEE International Symposium on Information Theory (ISIT)*, June 2018, pp. 1630–1634.

[23] A. Mallick, M. Chaudhari, U. Sheth, G. Palanikumar, and G. Joshi, "Rateless codes for near-perfect load balancing in distributed matrix-vector multiplication," 2018.

[24] S. Kiani, N. Ferdinand, and S. C. Draper, "Exploitation of stragglers in coded computation," in *2018 IEEE International Symposium on Information Theory (ISIT)*, June 2018, pp. 1988–1992.

[25] A. B. Das, L. Tang, and A. Ramamoorthy, "C3les: Codes for coded computation that leverage stragglers," in *2018 IEEE Information Theory Workshop (ITW)*, Nov 2018, pp. 1–5.

[26] E. Ozfatura, S. Ulukus, and D. Gündüz, "Distributed gradient descent with coded partial gradient computations," in *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2019, pp. 3492–3496.

[27] F. Haddadpour, Y. Yang, M. Chaudhari, V. R. Cadambe, and P. Grover, "Straggler-resilient and communication-efficient distributed iterative linear solver," *CoRR*, vol. abs/1806.06140, 2018.

[28] H. Wang, S. Guo, B. Tang, R. Li, and C. Li, "Heterogeneity-aware gradient coding for straggler tolerance," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, 2019, pp. 555–564.

[29] M. Kim, J. Sohn, and J. Moon, "Coded matrix multiplication on a group-based model," in *2019 IEEE International Symposium on Information Theory (ISIT)*, July 2019, pp. 722–726.

[30] Y. Yang, M. Interlandi, P. Grover, S. Kar, S. Amizadeh, and M. Weimer, "Coded elastic computing," in *2019 IEEE International Symposium on Information Theory (ISIT)*, July 2019, pp. 2654–2658.

[31] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Straggler mitigation in distributed matrix multiplication: Fundamental limits and optimal coding," in *2018 IEEE International Symposium on Information Theory (ISIT)*, June 2018, pp. 2022–2026.

[32] P. Soto, J. Li, and X. Fan, "Dual entangled polynomial code: Three-dimensional coding for distributed matrix multiplication," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. Long Beach, California, USA: PMLR, 09–15 Jun 2019, pp. 5937–5945.

[33] H. Park and J. Moon, "Irregular product coded computation for high-dimensional matrix multiplication," in *2019 IEEE International Symposium on Information Theory (ISIT)*, July 2019, pp. 1782–1786.

[34] Y. Sun, J. Zhao, S. Zhou, and D. Gunduz, "Heterogeneous coded computation across heterogeneous workers," in *2019 IEEE Global Communications Conference (GLOBECOM)*, 2019, pp. 1–6.

[35] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Coded fourier transform," *CoRR*, vol. abs/1710.06471, 2017. [Online]. Available: http://arxiv.org/abs/1710.06471

[36] C.-S. Yang, R. Pedarsani, and A. S. Avestimehr, "Timely-throughput optimal coded computing over cloud networks," in *Proceedings of the Twentieth ACM International Symposium on Mobile Ad Hoc Networking and Computing*, ser. Mobihoc '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 301–310.

[37] B. Buyukates and S. Ulukus, "Timely distributed computation with stragglers," 2019.
[38] B. Hasircioglu, J. Gomez-Vilardebo, and D. Gunduz, "Bivariate polynomial coding for exploiting stragglers in heterogeneous coded computing systems," 2020.
[39] R. Bitar, M. Wootters, and S. El Rouayheb, "Stochastic gradient coding for straggler mitigation in distributed learning," *IEEE Journal on Selected Areas in Information Theory*, pp. 1–1, 2020.
[40] H. Wang, Z. B. Charles, and D. S. Papailiopoulos, "Erasurehead: Distributed gradient descent without delays using approximate gradient coding," *CoRR*, vol. abs/1901.09671, 2019. [Online]. Available: http://arxiv.org/abs/1901.09671
[41] S. Wang, J. Liu, and N. Shroff, "Fundamental limits of approximate gradient coding," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 3, no. 3, Dec. 2019. [Online]. Available: https://doi.org/10.1145/3366700
[42] S. Dutta, G. Joshi, S. Ghosh, P. Dube, and P. Nagpurkar, "Slow and stale gradients can win the race: Error-runtime trade-offs in distributed SGD," in *The 21st International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2018.
[43] E. Ozfatura, S. Ulukus, and D. Gunduz, "Straggler-aware distributed learning: Communication computation latency trade-off," 2020.
[44] M. Luby, "LT codes," in *43rd Annual IEEE Symposium on Foundations of Computer Science*, November 2002.
[45] V. Bioglio, M. Grangetto, R. Gaeta, and M. Sereno, "An optimal partial decoding algorithm for rateless codes," in *2011 IEEE International Symposium on Information Theory Proceedings*, July 2011, pp. 2731–2735.
[46] E. Ozfatura, "coded computation with partial recovery," https://github.com/emre1925/coded-computation-with-partial-recovery, 2020.