Age-Based Coded Computation for Bias Reduction in Distributed Learning

Emre Ozfatura[1], Baturalp Buyukates[2], Deniz Gündüz[1], and Sennur Ulukus[2]

[1]Department of Electrical and Electronic Engineering, Imperial College London, UK
[2]Department of Electrical and Computer Engineering, University of Maryland, MD, USA

***Abstract*—Coded computation can speed up distributed learning in the presence of straggling workers. Partial recovery of the gradient vector can further reduce the computation time at each iteration; however, this can result in biased estimators, which may slow down convergence, or even cause divergence. Estimator bias is particularly prevalent when the straggling behavior is correlated over time, which results in the gradient estimators being dominated by a few fast servers. To mitigate biased estimators, we design a *timely* dynamic encoding framework for partial recovery that includes an ordering operator that changes the codewords and computation orders at workers over time. To regulate the recovery frequencies, we adopt an *age* metric in the design of the dynamic encoding scheme. The proposed age-based scheme prioritizes the recovery of computations with relatively large age. We show through numerical results that the proposed dynamic encoding strategy increases the timeliness of the recovered computations, which, as a result, reduces the bias in model updates, and accelerates the convergence compared to conventional static partial recovery schemes.**

I. Introduction

One of the main factors behind the success of machine learning algorithms is the availability of large datasets for training. However, as datasets become ever larger, the required computation becomes impossible to execute in a single machine within a reasonable time frame. This computational bottleneck can be overcome by distributed learning across multiple machines, called *workers*.

Gradient descent (GD) is the most common approach in supervised learning, and can be easily distributed. By employing a *parameter server* (PS) type framework [1], the dataset can be divided among workers, and at each iteration, workers compute gradients based on their local data, which are aggregated by the PS. However, slow, so-called *straggling*, workers are the Achilles heel of distributed GD (DGD) since the PS has to wait for all the workers to complete an iteration. A wide range of straggler-mitigation strategies have been proposed in recent years [2]–[19]. The main notion is to introduce redundancy in the computations assigned to each worker, so that fast workers can compensate for the stragglers.

Most of the coded computation solutions for straggler mitigation suffers from two drawbacks: First, they allow each worker to send only a single message per iteration, which results in the under-utilization of computational resources [19]. Second, they are designed to recover the full gradient at each iteration, which may unnecessarily increase the average completion time of an iteration. Multi-message communication (MMC) strategy addresses the first drawback by allowing each worker to send multiple messages per-iteration, thus, seeking a balance between computation and communication latency [5]–[8], [20]–[23]. The second drawback is addressed in [24] by combining coded computation with partial recovery (CCPR) to provide a trade-off between the average completion time of an iteration and the accuracy of the recovered gradient estimate.

If the straggling behavior is not independent and identically distributed over time and workers, which is often the case in practice, the gradient estimates recovered by the CCPR scheme become biased. For example, this may happen when a worker straggles over multiple iterations. Regulating the recovery frequency of the partial computations to make sure that each partial computation contributes to the model updates as equally as possible is critical to avoid biased updates. We use the age of information (AoI) metric to track the recovery frequency of partial computations.

AoI has been proposed to quantify the data freshness over systems that involve time-sensitive information [25]. AoI has found applications in queueing and networks, scheduling and optimization, and reinforcement learning (see the survey in [26]). Recently, [27] considered the age metric in a distributed computation system that handles time-sensitive computations, and [28] introduced an age-based metric to quantify the staleness of each update in a federated learning system. In our work, we associate an age to each partial computation and use this age to track the time passed since the last time each partial computation has been recovered.

In this paper, we design a dynamic encoding framework for the CCPR scheme that includes a timely dynamic order operator to prevent biased updates, and improve the performance. The proposed scheme increases the timeliness of the recovered partial computations by changing the codewords and their computation order over time. To regulate the recovery frequencies, we use the *age of partial computations (AoPC)* in the design of the dynamic order operator. We show by numerical experiments on a linear regression problem that the proposed dynamic encoding scheme increases the timeliness of the recovered computations, yields less biased model updates, and as a result, achieves better convergence performance compared to the conventional static encoding framework.

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).

II. System Model and Problem Formulation

For completeness, we first present the coded computation framework and the CCPR scheme.

A. DGD with Coded Computation

We focus on the least-squares linear regression problem, where the loss function is the empirical mean squared error

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

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

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

where η_t is the learning rate and $\boldsymbol{\theta}_t$ is the parameter vector at the tth iteration. Gradient of the loss function in (1) is

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

where $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_N]^T$ and $\mathbf{y} = [y_1, \ldots, y_N]^T$. In (3), only $\boldsymbol{\theta}_t$ changes over iterations. Thus, the key computational task at each iteration is the matrix-vector multiplication of $\mathbf{W}\boldsymbol{\theta}_t$, where $\mathbf{W} \triangleq \mathbf{X}^T\mathbf{X} \in \mathbb{R}^{d\times d}$. To speed up GD, execution of this multiplication can be distributed across K *workers*, by simply dividing $\mathbf{W}$ into K equal-size disjoint submatrices. However, under this naive approach, computation time is limited by the *straggling* workers [10].

Coded computation is used to tolerate stragglers by encoding the data before it is distributed among workers to achieve certain redundancy. That is, with coded computation, redundant partial computations are created such that the result of the overall computation can be obtained from a subset of the partial computations. Thus, up to a certain number of stragglers can be tolerated since the PS can recover the computation result without getting partial results from all workers. Many coded computation schemes, including MDS [6], [10], [15], LDPC [11], and rateless codes [21], and their various variants have been studied in the literature.

B. Coded Computation with Partial Recovery (CCPR)

In naive uncoded distributed gradient computation, straggling workers result in erasures in the gradient vector as illustrated in Fig. 1. The main motivation behind coded computation schemes is to find the minimum number of responsive workers to guarantee the recovery of the gradient vector without any erasures. Alternatively, the CCPR scheme [24] allows erasures on the gradient vector to reduce the computation time while controlling the number of erasures to guarantee certain accuracy for the gradient estimate. To implement the CCPR scheme, we employ a linear code structure similar to LT codes. In the case of centralized encoding, $\mathbf{W}$ is initially divided into K disjoint submatrices $\mathbf{W}_1, \ldots, \mathbf{W}_K \in \mathbb{R}^{d/K\times d}$ using which r coded submatrices, $\tilde{\mathbf{W}}_{i,1}, \ldots, \tilde{\mathbf{W}}_{i,r}$, are created and

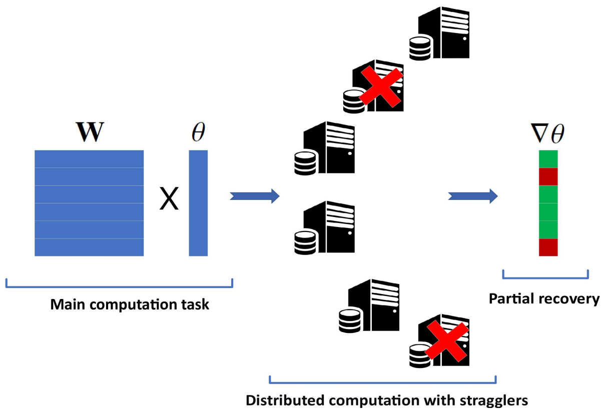

Fig. 1. Illustration of partial recovery in a naive distributed computation scenario with 6 workers, 2 of which are stragglers.

assigned to each worker i, $i \in [K]$, by the PS, where $\tilde{\mathbf{W}}_{i,j}$ is a linear combination of the K submatrices, i.e.,

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

Here, by using an analogy with LT codes, one can consider K submatrices $\mathbf{W}_1, \ldots, \mathbf{W}_K$ as symbols and their linear combinations in (4) as codewords, where the number of non-zero coefficients defines their degree. Following the encoding phase, at each iteration t, the ith worker performs the computations $\tilde{\mathbf{W}}_{i,1}\boldsymbol{\theta}_t, \ldots, \tilde{\mathbf{W}}_{i,r}\boldsymbol{\theta}_t$ in the given order, and sends the results one by one as soon as they are completed. In the meantime, the PS collects coded computations from all the workers until it successfully recovers $(1-q)\times 100$ percent of the gradient entries. Parameter q denotes the *tolerance*, a design parameter, which can be chosen according to the learning problem. We want to remark that due to the centralized encoding mechanism, coded submatrices are fixed throughout all the iterations. However, by considering a distributed encoding framework, as we explain next, it is possible to update coded submatrices through the iterations to have a more flexible design.

C. Distributed Dynamic Coded Submatrix Generation

Unlike the static centralized encoding scheme, inspired by the random circularly shifted (RCS) code design in [24], we consider a distributed dynamic coded submatrix generation framework. Initially all the submatrices are distributed among the workers. Then, at each iteration, workers dynamically generate their coded submatrices following a prescribed procedure. In particular, this dynamic framework utilizes three operators: *data partition* $D(\cdot)$, *ordering* $O(\cdot)$, and *encoding* $E(\cdot)$. The data partition operator distributes the submatrices $\mathcal{W} = \{\mathbf{W}_1, \ldots, \mathbf{W}_K\}$ among the workers such that

$$D_i(\mathcal{W}, M) : \mathcal{W} \mapsto \mathcal{W}_i, \quad |\mathcal{W}_i| \leq M, \tag{5}$$

where M is the given memory constraint, and $\mathcal{W}_i$ is the set of submatrices assigned to the ith worker. We assume that operator $D(\cdot)$ is executed, for each worker, only once at the beginning of the process, and $\mathcal{W}_i$, $i \in [K]$, remains the same over the iterations. The order operator $O(\cdot)$ is used to form an ordered set from the initial set $\mathcal{W}_i$ for encoding, i.e.,

$$O_{i,t}(\mathcal{W}_i) : \mathcal{W}_i \mapsto \tilde{\mathcal{W}}_{i,t}, \tag{6}$$

where $\tilde{\mathcal{W}}_{i,t}$ is an ordered set representing the order of computation at each iteration t for the ith worker. We remark that unlike the data partition operator, order operator may change over time. These two operators together can be represented by an assignment matrix $\mathbf{A}_t$, whose ith column is given by $\tilde{\mathcal{W}}_{i,t}$.

Once the assignment matrix $\mathbf{A}_t$ is fixed, the encoding process is executed according to a *degree vector* $\mathbf{m}$, which identifies the degree of each codeword based on its computation order. Encoding is executed for each worker independently. The encoder operator $E(\cdot)$ maps the ordered set of data (submatrices) to ordered set of codewords of size L, where L is the length of $\mathbf{m}$, i.e.,

$$E_{i,t}(\tilde{\mathcal{W}}_{i,t}, \mathbf{m}) : \tilde{\mathcal{W}}_{i,t} \mapsto \tilde{\mathcal{C}}_{i,t} = \left\{ \mathbf{C}_{i,1}^t, \dots, \mathbf{C}_{i,L}^t \right\}. \tag{7}$$

The encoding operator first divides set $\tilde{\mathcal{W}}_{i,t}$ into L disjoint subsets, $\mathcal{W}_{i,1}^t, \dots, \mathcal{W}_{i,L}^t$, such that $|\mathcal{W}_{i,l}^t| = \mathbf{m}(l)$. We note that the degree vector is chosen according to the RCS code design, such that $\mathbf{m}(L) > \mathbf{m}(L-1) > \ldots > \mathbf{m}(1) = 1$. Then, at iteration t, the coded submatrix of the ith worker with computation order ℓ, denoted by $\mathbf{C}_{i,\ell}^t$, is constructed as

$$\mathbf{C}_{i,\ell}^t = \sum_{\mathbf{W}_k \in \mathcal{W}_{i,\ell}^t} \mathbf{W}_k. \tag{8}$$

An example assignment matrix $\mathbf{A}_t$ is given below for $K = 20$ and $M = 6$:

$$\mathbf{A}_t = \begin{bmatrix} \mathbf{W}_1 & \mathbf{W}_2 & \mathbf{W}_3 & \dots & \mathbf{W}_{20} \\ \mathbf{W}_4 & \mathbf{W}_5 & \mathbf{W}_6 & \dots & \mathbf{W}_3 \\ \mathbf{W}_{11} & \mathbf{W}_{12} & \mathbf{W}_{13} & \dots & \mathbf{W}_{10} \\ \mathbf{W}_{15} & \mathbf{W}_{16} & \mathbf{W}_{17} & \dots & \mathbf{W}_{14} \\ \mathbf{W}_6 & \mathbf{W}_7 & \mathbf{W}_8 & \dots & \mathbf{W}_5 \\ \mathbf{W}_{18} & \mathbf{W}_{19} & \mathbf{W}_{20} & \dots & \mathbf{W}_{17} \end{bmatrix}. \tag{9}$$

The elements of the assignment matrix $\mathbf{A}_t$ are colored to illustrate the first step of the encoding operator, for $\mathbf{m} = [1, 2, 3]$, where colors blue, red and brown represent the submatrices used to generate the first, second, and third codewords, respectively. The encoding phase for the first worker at iteration t is illustrated below:

$$\tilde{\mathcal{W}}_{1,t} = \begin{bmatrix} \mathbf{W}_1 \\ \mathbf{W}_4 \\ \mathbf{W}_{11} \\ \mathbf{W}_{15} \\ \mathbf{W}_6 \\ \mathbf{W}_{18} \end{bmatrix} \rightarrow \tilde{\mathcal{C}}_{1,t} = \begin{bmatrix} \mathbf{W}_1 \\ \mathbf{W}_4 + \mathbf{W}_{11} \\ \mathbf{W}_{15} + \mathbf{W}_6 + \mathbf{W}_{18} \end{bmatrix}. \tag{10}$$

With this code, the worker first computes $\mathbf{W}_1\boldsymbol{\theta}_t$ and sends the result directly to the PS. Then, it computes $(\mathbf{W}_4 + \mathbf{W}_{11})\boldsymbol{\theta}_t$ sends the result to the PS. Finally, it computes $(\mathbf{W}_{15} + \mathbf{W}_6 + \mathbf{W}_{18})\boldsymbol{\theta}_t$ and sends the result to the PS.

Next, we formally state the problem using the data partition, ordering and encoding operators.

D. Problem Definition

The recovery of a partial computation $\mathbf{W}_k\boldsymbol{\theta}_t$ at iteration t depends on the data partition $\{D_i\}_{i\in[K]}$, ordering decisions

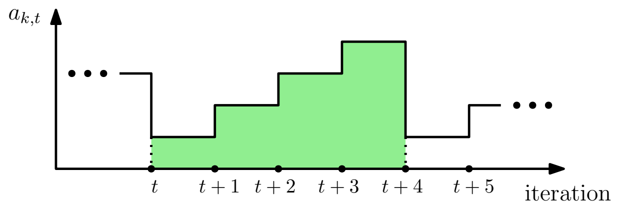

Fig. 2. Sample age evolution of $\mathbf{W}_k\boldsymbol{\theta}_{\bar{t}}$. Time t marks the beginning of iteration $t+1$. $\mathbf{W}_k\boldsymbol{\theta}_{\bar{t}}$ is recovered at iterations $\bar{t} = t$ and $\bar{t} = t+4$.

$\{O_{i,t}\}_{i\in[K]}$, encoding decisions $\{E_{i,t}\}_{i\in[K]}$, computation delay statistics of the workers, $\mathbf{d}_t$, and the tolerance q, i.e.,

$$\mathbf{r}_t = R(D, O_t, E_t, d_t, q), \tag{11}$$

where R is the recovery operation that returns a vector $\mathbf{r}_t$ which demonstrates the recovered partial computations such that $\mathbf{r}_t(k) = 1$ if $\mathbf{W}_k\boldsymbol{\theta}_t$ is recovered at the PS for $k \in [K]$.

In the partial recovery approach, without any further control on the assigned computations, operators are fixed throughout the training process. Thus, recovered submatrix indices may be correlated over time and some partial computations may not be recovered at all. We note that this kind of recovery behavior may lead to divergence especially when q is large, since the updates become biased. Our goal is to introduce a dynamic approach for the coded computation/partial recovery procedure to regulate the recovery frequency of each partial computation. For this, we first introduce an age-based performance metric.

We define the age of partial computation $\mathbf{W}_k\boldsymbol{\theta}_t$ at iteration t, denoted by $a_{k,t}$, as the number of iterations since the last time the PS recovered that partial computation. The age for each partial computation is updated at the end of each iteration in the following way

$$a_{k,t+1} = \begin{cases} a_{k,t} + 1, & \text{if } \mathbf{r}_t(k) = 0 \\ 1, & \text{if } \mathbf{r}_t(k) = 1 \end{cases}. \tag{12}$$

A sample age evolution of a partial computation is shown in Fig. 2 where partial computation $\mathbf{W}_k\boldsymbol{\theta}_{\bar{t}}$ is recovered at iterations $\bar{t} = t$ and $\bar{t} = t+4$. The average age of a partial computation over the training interval of T iterations is

$$a_k = \frac{1}{T}\sum_{t=1}^{T} a_{k,t}. \tag{13}$$

In order to make sure that each submatrix contributes to the model update as equally as possible during the training period, our goal is to keep the age of each partial computation under a certain threshold a_{th}. Thus, our objective is

$$\min_{\Pi(D,O,E)} \frac{1}{T}\sum_{t=1}^{T}\frac{1}{K}\sum_{k=1}^{K} \mathbb{1}_{\{a_{t,k} > a_{th}\}}, \tag{14}$$

where $\mathbb{1}_x$ is the indicator function that returns 1 if x holds, 0 otherwise. Here, a_{th} is a design parameter that determines the desired freshness level for the partial computations and can be adjusted according to the learning problem. We note that the problem in (14) is over all data partitions, ordering and encoding policies, thereby is hard to optimally solve.

Instead of solving (14) exactly, we introduce a timely dynamic ordering technique that can be used to regulate the recovery frequency of the partial computations.

III. Solution Approach: Timely Dynamic Ordering

In this section, we introduce timely dynamic ordering to better regulate the ages of partial computations and to avoid biased updates. We keep the data partition and encoding operators fixed and change only the ordering operator dynamically. This timely dynamic ordering is implemented by employing a vertical circular shift in the assignment matrix. With this, we essentially change the codewords and their computation order, which in turn, changes the recovered indices.

We first employ fixed vertical shifts for dynamic ordering. Then, we will dynamically adjust the shift amount based on the ages of the partial computations.

A. Fixed Vertical Shifts

In this code, which we call RCS-1, we employ one vertical shift for each worker at each iteration. That is, the order operator becomes

$$O_{i,t}(\mathcal{W}_i) : \mathcal{W}_i \mapsto circshift(\mathcal{W}_i, mod(t, L)), \qquad (15)$$

where $circshift$ is the circular shift operator and $mod(x, y)$ is a modulo operator returning the remainder of x/y. By using vertical shifts, coded computations transmitted to the PS from a particular worker change over time to prioritize certain partial computations. For example, if worker 1 employs the ordered set $\tilde{\mathcal{W}}_{1,t}$ and codewords $\tilde{\mathcal{C}}_{i,t}$ specified in (10) at iteration t, after applying one vertical shift, its computation order and codewords at iteration $t+1$, are given by

$$\tilde{\mathcal{W}}_{1,t+1} = \begin{bmatrix} \mathbf{W}_4 \\ \mathbf{W}_{11} \\ \mathbf{W}_{15} \\ \mathbf{W}_6 \\ \mathbf{W}_{18} \\ \mathbf{W}_1 \end{bmatrix} \rightarrow \tilde{\mathcal{C}}_{i,t+1} = \begin{bmatrix} \mathbf{W}_4 \\ \mathbf{W}_{11} + \mathbf{W}_{15} \\ \mathbf{W}_6 + \mathbf{W}_{18} + \mathbf{W}_1 \end{bmatrix}. \qquad (16)$$

Here, we see that, at iteration t, the worker prioritizes the computation of $\mathbf{W}_1\boldsymbol{\theta}_t$, while in the next iteration computation of $\mathbf{W}_4\boldsymbol{\theta}_{t+1}$ is prioritized. We note that, in this method, the shift amount is fixed to one shift at each iteration, and is independent of the ages of the partial computations.

Next, we introduce an age-based vertical shift scheme to control the order of computations.

B. Age-Based Vertical Shifts

In this code, which we call RCS-adaptive, we choose the vertical shift amount based on the current ages of the partial computations. That is, instead of shifting by 1 at each iteration, the shift amount changes across iterations based on the ages of the partial computations. To effectively avoid biased updates, we focus on recovering the partial computations with the highest age at the current iteration, that is, the computations that have not been recovered in a while. In line with the problem in (14), we term the partial computations with age higher than the threshold a_{th} as aged partial computations, which need to be recovered as soon as possible. To this end, a vertical shift amount is selected that places the maximum number of aged partial computations in the first position in the non-straggling workers' computation order so that they have a higher chance of recovery in the next iteration. In particular, to determine the shift amount in iteration $t+1$, the PS considers the computation order at the workers that have returned at least one computation in the previous iteration and determines a shift which places maximum number of aged partial computations in the first order in these workers. Upon determining the shift amount, every worker's assignment matrix is shifted by that amount in the next iteration. For example, if the age-based shift amount is 3 in iteration $t+1$, then the first user has

$$\tilde{\mathcal{W}}_{1,t+1} = \begin{bmatrix} \mathbf{W}_{15} \\ \mathbf{W}_6 \\ \mathbf{W}_{18} \\ \mathbf{W}_1 \\ \mathbf{W}_4 \\ \mathbf{W}_{11} \end{bmatrix} \rightarrow \tilde{\mathcal{C}}_{i,t+1} = \begin{bmatrix} \mathbf{W}_{15} \\ \mathbf{W}_6 + \mathbf{W}_{18} \\ \mathbf{W}_1 + \mathbf{W}_4 + \mathbf{W}_{11} \end{bmatrix}. \qquad (17)$$

Here, in iteration $t+1$, the first worker prioritizes the computation of $\mathbf{W}_{15}\boldsymbol{\theta}_{t+1}$.

IV. Numerical Results

In this section, we provide numerical results for comparing the proposed age-based partial computation scheme to alternative static schemes using a model-based scenario for computation latencies. For the simulations, we consider a linear regression problem over synthetically created training and test datasets, as in [12], of size of 2000 and 400, respectively. We also assume that the size of the model $d = 1000$ and the number of workers $K = 40$ while each worker can return $L = 3$ computations with $\mathbf{m} = [1, 2, 3]$. A single simulation includes $T = 400$ iterations. For all simulations, we use learning rate $\eta = 0.1$. To model the computation delays at the workers, we adopt the model in [19], and assume that the probability of completing s computations at any worker, performing s identical matrix-vector multiplications, by time t is given by

$$F_s(t) \triangleq \begin{cases} 1 - e^{-\mu(\frac{t}{s} - \alpha)}, & \text{if } t \geq s\alpha \\ 0, & \text{otherwise.} \end{cases} \qquad (18)$$

First, we consider an extreme scenario in the straggling behavior, where we assume there are 15 persistent stragglers that are fixed for all the $T = 400$ iterations which do not complete any partial computations. For the non-persistent stragglers, we set $\mu = 10$ and $\alpha = 0.01$.[1] In Fig. 3, we set the tolerance level $q = 0.3$, such that at each iteration the PS aims at recovering 28 of the total 40 partial computations. We see that the proposed timely dynamic encoding strategy with one vertical shift at each iteration, RCS$-$1, achieves a significantly better convergence performance than the conventional static encoding with RCS. When the ages of partial computations are taken into consideration in determining the order of

[1]To simulate the straggling behavior in our simulations, we take $\alpha = 10$ for the persistent stragglers so that effectively they do not complete any partial computations.

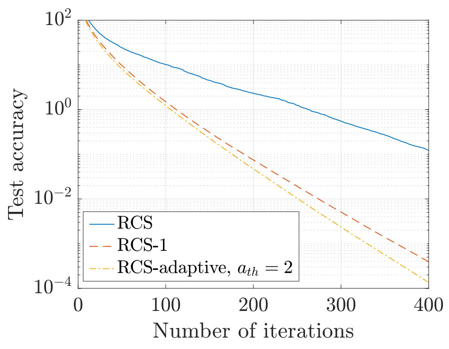

Fig. 3. Test accuracy (log-scale) vs. number of iterations with $q = 0.3$ and 15 persistent stragglers.

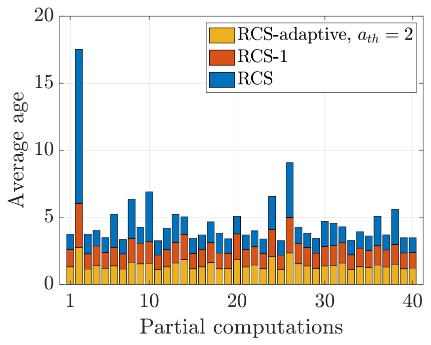

Fig. 4. Average ages of the partial computations with $q = 0.3$ and 15 persistent stragglers.

computation at each iteration with the proposed RCS-adaptive scheme with an age threshold of $a_{th} = 2$, we observe a further improvement in the convergence performance.

An interesting observation comes from Fig. 4, where we plot the average ages of the partial computations. While the proposed timely dynamic encoding strategy does not result in a better average age performance for every single partial computation, it targets the partial computations with the highest average age (see computation tasks 2, 10, and 26 in Fig. 4). By utilizing the dynamic order operator, we essentially lower the average age of the partial computations with the worst age performance at the expense of slight increase in the age of some of the remaining partial computations. As expected, age-based vertical shift strategy further lowers the average ages of the partial computations. Here, we can draw parallels with this result and [29], which shows that as long as each component is received every p iterations, the distributed SGD can maintain its asymptotic convergence rate. From Fig. 4, we can see that the proposed vertical shift operator guarantees that on average each task is received every 3 iterations, since the yellow bar in Fig. 4 is less than 3 for each partial computation.

We note that in Figs. 3 and 4 the performance gap between RCS-1 and RCS-adaptive schemes is narrow. This shows that the randomness introduced by a fixed vertical shift is already quite helpful in mitigating the biased updates with less stale partial computations.

In Table I, we look at the value of the objective function in (14) when $a_{th} = 2$ with $\mu = 10$ and $\alpha = 0.01$ for varying tolerance levels in the case of fixed 15 persistent stragglers throughout all the iterations. We observe that, for each tolerance level q, when RCS-1 is employed, we achieve a better performance than the static RCS scheme, whereas the age-based vertical shift method RCS-adaptive results in the best performance. This is because the RCS-adaptive scheme specifically targets the computational tasks that have average age higher than the threshold a_{th} to effectively create less biased model updates where each partial computation contributes to the learning task more uniformly.

Second, we consider a more realistic scenario and model the straggling behavior of workers based on a two-state Markov chain: a slow state s and a fast state f, such that computations are completed faster when a worker is in state f. This is similar to the Gilbert-Elliot service times considered in [13], [30]. Specifically, in (18) we have rate μ_f in state f and rate μ_s in state s where $\mu_f > \mu_s$. We assume that the state changes only occur at the beginning of each iteration with probability p; that is, with probability $1 - p$ the state stays the same. A low switching probability p indicates that the straggling behavior tends to stay the same in the next iteration. In Fig. 5, we set $p = 0.05$, $q = 0.3$, $\alpha = 0.01$, $\mu_s = 2$, and $\mu_f = 10$ and let 15 workers start at the slow state, i.e., initially we have 15 straggling workers. We note that with 15 initial stragglers and $p = 0$ we recover the setting considered in

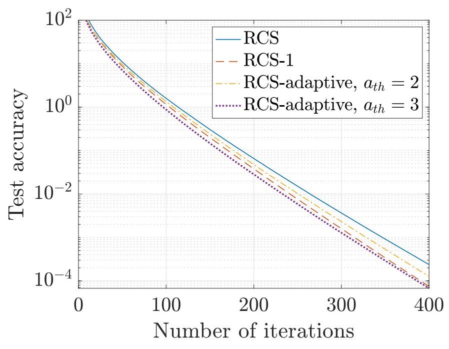

Fig. 5. Test accuracy (log-scale) vs. number of iterations with $q = 0.3$ and straggling behavior based on a 2-state Markov chain with a state transition probability of $p = 0.05$.

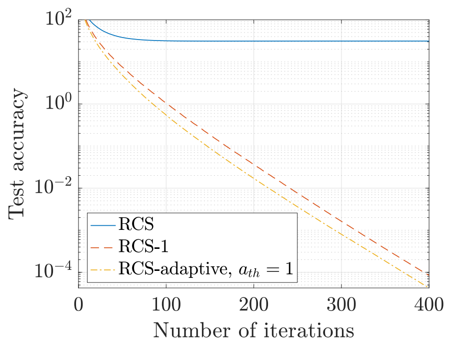

Fig. 6. Test accuracy (log-scale) vs. number of iterations for the uncoded scheme with $q = 0.2$ and 15 persistent stragglers.

Tolerance level	RCS	RCS-1	RCS-adaptive
$q = 0.1$	0.0261	0.0180	0.0156
$q = 0.2$	0.0681	0.0476	0.0451
$q = 0.3$	0.1316	0.0970	0.0919

TABLE I
THE VALUE OF THE OBJECTIVE FUNCTION IN (14) WHEN $a_{th} = 2$ FOR VARYING TOLERANCE LEVELS q.

Fig. 3. We observe in Fig. 5 that the proposed timely dynamic encoding strategy improves the convergence performance even though the performance improvement is less compared to the setting in Fig. 3. This is because, in this scenario, the straggling behavior is less correlated over iterations, which results in less biased model updates even for the static RCS scheme. Further, we see in Fig. 5 that the RCS-adaptive scheme with $a_{th} = 3$ performs the best, whereas the RCS-1 scheme outperforms the RCS-adaptive scheme when $a_{th} = 2$. This shows that the age threshold a_{th} needs to be tuned to get the best performance from the RCS-adaptive scheme.

Even though we focus on the distributed coded computation scenario in this work, the proposed dynamic order operator can be applied when the computations are assigned to workers in an uncoded fashion as well. To see the performance in the case of uncoded computations with MMC, we set $\mathbf{m} = [1, 1, 1]$ and $q = 0.2$ and consider the same setup as in Fig. 3. In Fig. 6, we observe that the static partial recovery scheme fails to converge since if coding is not implemented along with partial recovery, model updates are highly biased in the presence of persistent stragglers. However, when the dynamic order operator is employed, particularly the age-aware vertical shifts with $a_{th} = 1$, convergence is achieved.

V. Conclusion

MMC and partial recovery are two strategies designed to enhance the performance of coded computation employed for straggler-aware distributed learning. The main drawback of the partial recovery strategy is biased model updates that are caused when the straggling behaviors of the workers are correlated over time. To prevent biased updates, we introduce a timely dynamic encoding strategy which changes the codewords and their computation order over time. We use an age metric to regulate the recovery frequencies of the partial computations. By conducting several experiments on a linear regression problem, we show that dynamic encoding, particularly an age-based encoding strategy, can significantly improve the convergence performance compared to conventional static encoding schemes. Although our main focus is on coded computation, the advantages of the proposed strategy are not limited to the coded computation scenario. The proposed timely dynamic encoding strategy can be utilized for coded communication and uncoded computation scenarios as well.

References

[1] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su. Scaling distributed machine learning with the parameter server. In *USENIX Conference on Operating Systems Design and Implementation*, October 2014.

[2] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis. Gradient coding: Avoiding stragglers in distributed learning. In *ICML*, August 2017.

[3] W. Halbawi, N. Azizan, F. Salehi, and B. Hassibi. Improving distributed gradient descent using Reed-Solomon codes. In *IEEE ISIT*, June 2018.

[4] E. Ozfatura, D. Gunduz, and S. Ulukus. Gradient coding with clustering and multi-message communication. In *IEEE DSW*, June 2019.

[5] L. Tauz and L. Dolecek. Multi-message gradient coding for utilizing non-persistent stragglers. In *Asilomar Conference*, November 2019.

[6] N. Ferdinand and S. C. Draper. Hierarchical coded computation. In *IEEE ISIT*, June 2018.

[7] E. Ozfatura, D. Gunduz, and S. Ulukus. Speeding up distributed gradient descent by utilizing non-persistent stragglers. In *IEEE ISIT*, July 2019.

[8] S. Kiani, N. Ferdinand, and S. C. Draper. Exploitation of stragglers in coded computation. In *IEEE ISIT*, June 2018.

[9] N. Charalambides, M. Pilanci, and A. O. Hero. Weighted gradient coding with leverage score sampling. In *IEEE ICASSP*, May 2020.

[10] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran. Speeding up distributed machine learning using codes. *IEEE Tran. Inf. Theory*, 64(3):1514–1529, March 2018.

[11] R. K. Maity, A. Singh Rawa, and A. Mazumdar. Robust gradient descent via moment encoding and LDPC codes. In *IEEE ISIT*, July 2019.

[12] S. Li, S. M. M. Kalan, Q. Yu, M. Soltanolkotabi, and A. S. Avestimehr. Polynomially coded regression: Optimal straggler mitigation via data encoding. January 2018. Available on arXiv: 1805.09934.

[13] C. S. Yang, R. Pedarsani, and A. S. Avestimehr. Timely coded computing. In *IEEE ISIT*, July 2019.

[14] S. Dutta, M. Fahim, F. Haddadpour, H. Jeong, V. Cadambe, and P. Grover. On the optimal recovery threshold of coded matrix multiplication. *IEEE Trans. Inf. Theory*, 66(1):278–301, July 2019.

[15] H. Park, K. Lee, J. Sohn, C. Suh, and J. Moon. Hierarchical coding for distributed computing. In *IEEE ISIT*, June 2018.

[16] Y. Yang, M. Interlandi, P. Grover, S. Kar, S. Amizadeh, and M. Weimer. Coded elastic computing. In *IEEE ISIT*, July 2019.

[17] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr. Straggler mitigation in distributed matrix multiplication: Fundamental limits and optimal coding. In *IEEE ISIT*, June 2018.

[18] R. Bitar, M. Wootters, and S. El Rouayheb. Stochastic gradient coding for straggler mitigation in distributed learning. *IEEE Journal on Sel. Areas in Inf. Theory*, pages 1–1, 2020.

[19] E. Ozfatura, S. Ulukus, and D. Gunduz. Straggler-aware distributed learning: Communication computation latency trade-off. *Entropy, Special Issue on the Interplay Between Storage, Computing, and Communications from an Information-Theoretic Perspective*, 22(5):544, May 2020.

[20] E. Ozfatura, S. Ulukus, and D. Gündüz. Distributed gradient descent with coded partial gradient computations. In *IEEE ICASSP*, May 2019.

[21] A. Mallick, M. Chaudhari, and G. Joshi. Fast and efficient distributed matrix-vector multiplication using rateless fountain codes. In *IEEE ICASSP*, May 2019.

[22] B. Hasircioglu, J. Gomez-Vilardebo, and D. Gunduz. Bivariate polynomial coding for exploiting stragglers in heterogeneous coded computing systems, January 2020. Available on arXiv:2001.07227.

[23] M. Mohammadi Amiri and D. Gunduz. Computation scheduling for distributed machine learning with straggling workers. *IEEE Trans. on Sig. Proc.*, 67(24):6270–6284, December 2019.

[24] E. Ozfatura, S. Ulukus, and D. Gunduz. Coded distributed computing with partial recovery. July 2020. Available on arXiv:2007.02191.

[25] S. K. Kaul, R. D. Yates, and M. Gruteser. Real-time status: How often should one update? In *IEEE Infocom*, March 2012.

[26] Y. Sun, I. Kadota, R. Talak, and E. Modiano. Age of information: A new metric for information freshness. *Synthesis Lectures on Communication Networks*, 12(2):1–224, December 2019.

[27] B. Buyukates and S. Ulukus. Timely distributed computation with stragglers. October 2019. Available on arXiv: 1910.03564.

[28] H. H. Yang, A. Arafa, T. Q. S. Quek, and H. V. Poor. Age-based scheduling policy for federated learning in mobile edge networks. In *IEEE ICASSP*, May 2020.

[29] P. Jiang and G. Agrawal. A linear speedup analysis of distributed deep learning with sparse and quantized communication. In *NIPS*. December 2018.

[30] B. Buyukates and S. Ulukus. Age of information with Gilbert-Elliot servers and samplers. In *CISS*, March 2020.