Gradient Coding With Dynamic Clustering for Straggler-Tolerant Distributed Learning

Baturalp Buyukates, *Member, IEEE*, Emre Ozfatura, *Member, IEEE*, Sennur Ulukus, *Fellow, IEEE*, and Deniz Gündüz, *Fellow, IEEE*

***Abstract*— Distributed implementations are crucial in speeding up large scale machine learning applications. Distributed gradient descent (GD) is widely employed to parallelize the learning task by distributing the dataset across multiple workers. A significant performance bottleneck for the per-iteration completion time in distributed synchronous GD is *straggling* workers. Coded distributed computation techniques have been introduced recently to mitigate stragglers and to speed up GD iterations by assigning redundant computations to workers. In this paper, we introduce a novel paradigm of dynamic coded computation, which assigns redundant data to workers to acquire the flexibility to dynamically choose from among a set of possible codes depending on the past straggling behavior. In particular, we propose gradient coding (GC) with dynamic clustering, called GC-DC, and regulate the number of stragglers in each cluster by dynamically forming the clusters at each iteration. With time-correlated straggling behavior, GC-DC adapts to the straggling behavior over time; in particular, at each iteration, GC-DC aims at distributing the stragglers across clusters as uniformly as possible based on the past straggler behavior. For both homogeneous and heterogeneous worker models, we numerically show that GC-DC provides significant improvements in the average per-iteration completion time without an increase in the communication load compared to the original GC scheme.**

***Index Terms*— Distributed coded computation, gradient descent, straggler mitigation, gradient coding, clustering.**

Manuscript received 15 August 2021; revised 7 January 2022 and 19 March 2022; accepted 3 April 2022. Date of publication 12 April 2022; date of current version 16 June 2023. This work was supported by EC H2020-MSCA-ITN-2015 project SCAVENGE under grant number 675891, by the European Research Council project BEACON under grant number 677854, and by CHIST-ERA grant CHIST-ERA-18-SDCDN-001 (funded by EPSRC-EP/T023600/1). An earlier version of this paper was presented in part at the IEEE International Conference on Communications, Montreal, Canada, June 2021 [DOI: 10.1109/ICC42927.2021.9500346]. The associate editor coordinating the review of this article and approving it for publication was S. Jin. *(Corresponding author: Sennur Ulukus.)*

Baturalp Buyukates was with the Department of Electrical and Computer Engineering, University of Maryland, College Park, MD 20742 USA. He is now with the Department of Electrical and Computer Engineering, University of Southern California, Los Angeles, CA 90007 USA (e-mail: buyukate@usc.edu).

Emre Ozfatura is with the Department of Electrical and Electronic Engineering, Imperial College London, London SW7 2BX, U.K. (e-mail: m.ozfatura@imperial.ac.uk).

Sennur Ulukus is with the Department of Electrical and Computer Engineering, University of Maryland, College Park, MD 20742 USA (e-mail: ulukus@umd.edu).

Deniz Gündüz is with the Department of Electrical and Electronic Engineering, Imperial College London, London SW7 2BX, U.K., and also with the Department of Engineering "Enzo Ferrari," University of Modena and Reggio Emilia (Unimore), 41121 Modena, Italy (e-mail: d.gunduz@imperial.ac.uk).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TCOMM.2022.3166902.

Digital Object Identifier 10.1109/TCOMM.2022.3166902

I. Introduction

GRADIENT descent (GD) methods are widely used in machine learning problems to optimize the model parameters in an iterative fashion. When the size of the training datasets and the complexity of the trained models are formidable, it is not feasible to train the model on a single machine within a reasonable time frame. To speed up GD iterations, gradient computations can be distributed across multiple workers. In a typical parameter server (PS) framework with synchronous GD iterations, the dataset is distributed across the workers, and each worker computes a gradient estimate, also called a *partial gradient*, based on its own local dataset. The PS aggregates these partial gradients to obtain the full gradient and update the model. In this distributed setting, the main performance bottleneck is the slowest *straggling* workers. Many recent works have focused on developing straggler-tolerant distributed GD schemes (see survey in [2] for distributed learning in wireless networks and in [3] for coded distributed computing). In these works, the main theme is to assign redundant computations to workers to overcome the potential delays caused by straggling workers. One way to obtain redundancy is through coded dataset assignment to workers, i.e., coded computation [4]–[25]. Among these works, a particular use case of coded computation is distributed matrix multiplication [8]–[10], [20], [26], where, encoded submatrices are distributed to the workers so that certain number of straggling/unresponsive workers can be tolerated. In the case of linear computations, such as matrix-vector multiplications, linear coding techniques, including MDS codes and rateless codes, are utilized [4], [5], [14]–[16], [21]. Another technique to introduce redundancy to the system is through coded communication, i.e., coded transmission [27]–[37], in which case workers perform computations on uncoded data but send certain linear combinations of the results to the PS. The most prominent technique in coded transmission is gradient coding [27], which is detailed below and is the main focus of this work. A third way of introducing redundancy to computations is by simply using backup computations, i.e., uncoded computation [38]–[43]. In this case, no coding is implemented as in naive distributed computation, but unlike the naive approach, the PS assigns some back up computations to workers in order to compensate for the slower straggling workers.

In this paper, we consider the gradient coding (GC) framework introduced in [27]. The dataset is distributed across the workers in an uncoded but redundant manner, and the workers return coded computations to the PS. We note that

this can also model a scenario, in which data is collected directly by the workers, instead of being distributed by the server. Redundancy can either be created by data sharing among the workers, or may be inherent due to the data collection/generation mechanism. Thanks to the redundancy in the local datasets, partial gradients from only a subset of the workers will be sufficient to recover the full gradient. Coded combinations retrieved by the workers are designed such that any subset of responses from sufficiently many workers will allow the computation of the full gradient by the PS. Further details of GC are presented in the next section.

To improve the performance of the GC scheme, reference [32] proposes a static clustering technique, which entails dividing the workers into smaller clusters and applying the original GC scheme at the cluster level. This technique is shown to improve the average computation time compared to the original GC scheme. With clustering, unlike in the original GC scheme, the number of tolerated stragglers scales with the number of clusters when the stragglers are uniformly distributed among the clusters. However, this may not be the case in practical scenarios as evident in the measurements taken over Amazon EC2 clusters that indicate a time-correlated straggling behavior for the workers [18], [27]. In this case, the advantage of clustering diminishes since the stragglers are not uniform across clusters.

To mitigate this problem and to further improve the performance, in this paper, we introduce a novel paradigm of dynamic coded computation, which assigns more data samples to workers than the actual computation load (per-iteration) to give them the flexibility in choosing the computations they need to carry out at each iteration. This allows the PS to choose which subset of computations each worker should try to complete at each iteration, and which coded combination it should transmit back to the PS. In particular, to reduce the potential solution space, we propose a novel GC scheme with dynamic clustering, called GC-DC, where the PS decides on the clusters to be formed at each iteration. At each iteration in GC-DC, the PS forms the clusters such that the stragglers are distributed across the clusters as uniformly as possible based on the workers' past straggling behavior. We numerically show that the proposed GC-DC scheme significantly improves the average per-iteration completion time without an increase in the communication load under both homogeneous and heterogeneous worker environments.[1]

The rest of this paper is organized as follows: In Section II, we present the GC and GC with clustering frameworks. In Section III, we introduce the GC with dynamic codeword assignment scheme to improve the average iteration completion time of the static GC schemes and present the problem formulation. In Section IV, we transform the GC with dynamic codeword assignment problem to a dynamic clustering problem and illustrate its advantage over the original GC and GC with static clustering schemes. Section V presents the proposed greedy dynamic clustering strategy and Section VI demonstrates its effectiveness through numerical simulations over the existing static GC schemes. Finally, we conclude this paper in Section VII with a summary of the main results along with a discussion of some future directions.

[1]In the proposed GC-DC scheme, at each iteration, each worker sends a single codeword consisting of a linear combination of the computed partial gradients back to the PS. Since the number of transmitted messages, which is one in the case of GC-DC, is the same as those of the GC and GC-SC schemes, the proposed GC-DC scheme does not increase the communication load compared to the existing GC and GC-SC schemes.

II. Preliminaries: Gradient Coding (GC) and Clustering

In this section, we describe our system model. Since our focus in this work is on gradient coding and clustering, we first present the original GC scheme [27] and the GC with static clustering scheme [32] and demonstrate their limitations, which motivate the proposed dynamic gradient coding approach, which we formally state in Section III.

In many machine learning problems, given a labeled dataset $\mathcal{D} = \{(\mathbf{x}_1, y_1), \dots (\mathbf{x}_s, y_s)\}$, where $\mathbf{x}_1, \dots, \mathbf{x}_s \in \mathbb{R}^d$ are the data points with corresponding labels $y_1, \dots, y_s \in \mathbb{R}$, the goal is to solve the following optimization problem

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta} \in \mathbb{R}^d}{\operatorname{argmin}} \sum_{i=1}^{s} l(\mathbf{x}_i, y_i, \boldsymbol{\theta}), \tag{1}$$

where l is the application-specific loss function and $\boldsymbol{\theta} \in \mathbb{R}^d$ is the parameter vector to be optimized. The optimal parameter vector can be obtained iteratively using GD. The *full gradient* computed over the whole dataset at iteration t is given by $\boldsymbol{g}^{(t)} = \sum_{i=1}^{s} \nabla l(\mathbf{x}_i, y_i, \boldsymbol{\theta}_t)$. When the size of the dataset, s, is large, the computation of the full gradient becomes a performance bottleneck. To speed up GD iterations, gradient computations can be distributed across multiple workers. However, in many implementations, particularly in the context of 'serverless' computing, e.g., Microsoft Azure, Amazon Web Services (AWS), the workers' completion time of assigned tasks can be highly heterogeneous and stochastic over time. In those cases, the overall computation speed of each iteration becomes limited by the slowed *straggling* server. Coded computing techniques tackle the bottleneck due to stragglers by introducing redundant computations in a structured manner such that additional computations carried out by faster servers can compensate for the stragglers.

A. Gradient Coding (GC)

GC is a distributed coded computation technique introduced in [27] to perform distributed GD across K workers. The complete dataset $\mathcal{D}$ is divided into K non-overlapping equal-size mini-batches, $\mathcal{D}_1, \dots, \mathcal{D}_K$, and each worker is assigned multiple mini-batches. We denote the set of indices of mini-batches assigned to the kth worker with $\mathcal{I}_k$, $k \in [K] \triangleq \{1, \dots, K\}$. Let $\boldsymbol{g}_k^{(t)}$ denote the partial gradient for the parameter vector $\boldsymbol{\theta}_t$ evaluated over mini-batch $\mathcal{D}_k$ at the tth GD iteration, i.e.,

$$\boldsymbol{g}_k^{(t)} = \frac{1}{|\mathcal{D}_k|} \sum_{(\mathbf{x}, y) \in \mathcal{D}_k} \nabla l(\mathbf{x}, y, \boldsymbol{\theta}_t). \tag{2}$$

We note that the *full gradient* is given by $\boldsymbol{g}^{(t)} = \frac{1}{K} \sum_{k=1}^{K} \boldsymbol{g}_k^{(t)}$. To tolerate straggling workers, GC assigns redundant mini-batches, and hence, redundant computations, to the workers.

If a mini-batch $\mathcal{D}_i$ is assigned to worker k, i.e., $i \in \mathcal{I}_k$, then the corresponding partial gradient $\boldsymbol{g}_i^{(t)}$ is computed by the kth worker. *Computation load*, r, denotes the number of

mini-batches assigned to each worker, i.e., $|\mathcal{I}_k| = r, \forall k \in [K]$. At each iteration, each worker first computes the r partial gradients, one for each mini-batch available locally, and sends a linear combination of the results, $\boldsymbol{c}_k^{(t)} \triangleq \mathcal{L}_k(\boldsymbol{g}_i^{(t)} : i \in \mathcal{I}_k)$, called a *coded partial gradient*. Thus, in GC, each worker is responsible for computing a single predefined coded partial gradient. The underlying code structure in GC, which dictates the linear combinations formed by each worker, exploits the available redundancy so that the PS can recover the full gradient from only a subset of the combinations. Accordingly, from now on, we refer to the coded partial gradients formed by the workers simply as *codewords*. As shown in [27], the GC scheme can tolerate up to $r-1$ persistent stragglers[2] at each iteration. Formally, for any set of non-straggling workers $\mathcal{W} \subseteq [K]$ with $|\mathcal{W}| = K - r + 1$, there exists a set of coefficients $\mathcal{A}_{\mathcal{W}} = \left\{a_k^{(t)} : k \in \mathcal{W}\right\}$ such that

$$\sum_{k \in \mathcal{W}} a_k^{(t)} \boldsymbol{c}_k^{(t)} = \frac{1}{K} \sum_{k=1}^{K} \boldsymbol{g}_k^{(t)}. \tag{3}$$

Thus, at each iteration t, the full gradient $\boldsymbol{g}^{(t)}$ can be recovered from any $K - r + 1$ codewords.

Next, we present the idea of *clustering* that was introduced in [32] to reduce the average per-iteration completion time of the GC scheme.

B. Gradient Coding With Static Clustering (GC-SC)

In GC with clustering, we divide the workers into P disjoint equal-size clusters. Let $\mathcal{K}_p \subset [K]$ denote the set of workers in cluster p, $p \in [P]$, where $\mathcal{K}_q \cap \mathcal{K}_p = \emptyset$ for $q \neq p$, and $\bigcup_{p \in [P]} \mathcal{K}_p = [K]$. We denote the cluster size by $\ell \triangleq \frac{K}{P}$, where we assume that K is divisible by P for simplicity. The assignment of the workers to the clusters is dictated by an $\ell \times p$ worker assignment matrix, denoted by $\mathbf{A}_{cluster}$, where each column corresponds to a different cluster and the entries in each column correspond to indices of the workers assigned to that cluster. This worker assignment matrix is fixed throughout the training process, hence the name *static clustering*. From now on, we refer to the GC with static clustering scheme as GC-SC.

In GC-SC, each worker is assigned r mini-batches based on its cluster. This is represented by an $r \times K$ data assignment matrix $\mathbf{A}_{data}$, where each column corresponds to a different worker, and the entries in column i, $i \in [K]$, represent the mini-batches (correspondingly the partial gradient computations) assigned to the ith worker. Equivalently, data assignment can be represented by a $1 \times K$ codeword assignment matrix $\mathbf{A}_{code}$, which represents the codewords assigned to the workers, where the codeword assigned to the ith worker in the pth cluster is denoted by $c_{p,i}$, for $p \in [P]$, $i \in [\ell]$. Let $\mathcal{I}_{\mathcal{K}_p}$ denote the set of mini-batches assigned to the workers in the pth cluster, i.e., $\mathcal{I}_{\mathcal{K}_p} = \bigcup_{k \in \mathcal{K}_p} \mathcal{I}_k$. In GC-SC, the GC scheme is applied to each cluster separately and the workers in cluster p aim at computing

$$\frac{1}{|\mathcal{I}_{\mathcal{K}_p}|} \sum_{k \in \mathcal{I}_{\mathcal{K}_p}} \boldsymbol{g}_k^{(t)}. \tag{4}$$

To illustrate the advantage of the clustering technique, consider $K = 12$, $r = 2$, and $P = 4$. Here, the workers are divided into 4 clusters, each consisting of $\ell = 3$ workers, and each cluster is responsible for computing 3 of the total 12 partial gradients. Since $r = 2$, each worker aims at computing the assigned 2 partial gradients.

In our example, the worker assignment can be specified by the following matrix:

$$\mathbf{A}_{cluster} = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 6 & 7 & 8 & 5 \\ 9 & 10 & 11 & 12 \end{bmatrix}. \tag{5}$$

In this assignment, workers 1, 6 and 9 are in the first cluster, workers 2, 7 and 10 are in the second cluster, and so on. The corresponding $\mathbf{A}_{data}$ is given in (6), as shown at the bottom of the page, for the cluster assignment in (5). In (6), workers in each cluster are represented by a different color. We use blue, red, magenta, and green for clusters 1, 2, 3, and 4, respectively. The corresponding $\mathbf{A}_{code}$ for the cluster assignment in (5) is given in (7), as shown at the bottom of the page, where, codewords corresponding to different clusters are shown in different colors. Each codeword in $\mathbf{A}_{code}$ is a linear combination of $r = 2$ partial gradients. For example, $c_{1,1}$ is a linear combination of partial gradients g_1 and g_2; $c_{1,2}$ is a linear combination of partial gradients g_2 and g_3, and $c_{1,3}$ is a linear combination of partial gradients g_3 and g_1. Thus, given $\mathbf{A}_{cluster}$, either $\mathbf{A}_{data}$ or $\mathbf{A}_{code}$ is sufficient to completely characterize the partial computations that will be carried out by each worker.

In the original GC scheme, the PS waits until it receives $K - r + 1 = 11$ results at each iteration; hence only $r - 1 = 1$ straggler can be tolerated. With clustering, the PS needs to receive at least $\ell - r + 1 = 2$ results from each cluster to recover the full gradient. Thus, the non-straggling threshold is still $K - r + 1$, since more than one straggler cannot be tolerated if they are in the same cluster. However, the non-straggling threshold represents a worst case scenario. With clustering, up to 4 stragglers can be tolerated if they are uniformly distributed across clusters, i.e., one straggler per cluster, as shown in "Realization 1" in Fig. 1. This shows that,

[2]These are the straggler workers that either cannot complete any computation or whose computations are not used while recovering the full gradient [32].

$$\mathbf{A}_{data} = \begin{bmatrix} g_1 & g_4 & g_7 & g_{10} & g_{11} & g_2 & g_5 & g_8 & g_3 & g_6 & g_9 & g_{12} \\ g_2 & g_5 & g_8 & g_{11} & g_{12} & g_3 & g_6 & g_9 & g_1 & g_4 & g_7 & g_{10} \end{bmatrix} \tag{6}$$

$$\mathbf{A}_{code} = \begin{bmatrix} c_{1,1} & c_{2,1} & c_{3,1} & c_{4,1} & c_{4,2} & c_{1,2} & c_{2,2} & c_{3,2} & c_{1,3} & c_{2,3} & c_{3,3} & c_{4,3} \end{bmatrix} \tag{7}$$

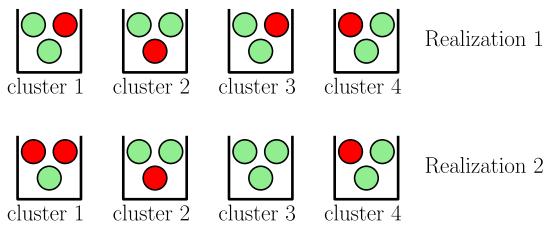


Fig. 1. Two possible straggler realizations where red and green circles represent the straggling and non-straggling workers, respectively.

with clustering, the full gradient can be recovered in a much larger set of realizations compared to the original GC scheme. Thus, even if the non-straggling threshold (which corresponds to the worst case scenario) remains the same, clustering will reduce the average per-iteration completion time.

Formally, with clustering, it is possible to tolerate $r-1$ stragglers in each cluster in the best case scenario, which is when the stragglers are uniformly distributed among the clusters. In this case, it is possible to tolerate $P(r-1)$ stragglers in total. However, this advantage of clustering diminishes in the case of non-uniform distributed stragglers among the clusters, which may be the case in practice. As shown in "Realization 2" in Fig. 1, even though there are still 8 non-straggling workers, the PS cannot compute the full gradient (in the case of persistent stragglers) when the stragglers are not uniformly distributed across the clusters. The problem we tackle in this work is to design a dynamic gradient coding approach that enables full recovery even in the case of non-uniform straggler distribution across clusters. To this end, in the next section, we introduce the concept of dynamic codeword assignment, which dynamically changes codewords computed by the workers at each iteration based on the past straggler behavior to further improve the average iteration time compared to the existing static implementations.

III. GC With Dynamic Codeword Assignment

In the conventional coded computation approaches, including GC, the assignment of the dataset to the workers and the code to be used are static and set at the beginning of the training. That is, at every iteration, a worker tries to compute the gradient estimates for all the mini-batches assigned to it, and returns their exact same linear combination. Thus, in order to recover the desired computation result at each iteration, the codes are designed for the worst case scenario. The core idea behind dynamic codeword assignment is to change the codewords assigned to the workers dynamically based on the observed straggling behavior. Dynamic codeword assignment is driven by two policies; namely, *data assignment* and *codeword assignment*. The data assignment policy, denoted by Π_d, is executed only once at the beginning of training and assigns up to m mini-batches to each worker, where m denotes the memory constraint, i.e.,

$$\Pi_d : \mathcal{D} \mapsto \{\mathcal{I}_1, \dots, \mathcal{I}_K : |\mathcal{I}_k| \leq m\}. \tag{8}$$

Even though each worker can be allocated up to m mini-batches, each will compute only r of them at each iteration; hence, the computation load at each iteration remains the same. We can have $\binom{m}{r}$ codewords that can be assigned to each worker depending on which subset of r computations it carries out among m possibilities. Here, we introduce $\mathcal{C} = \{\mathcal{C}_1, \dots, \mathcal{C}_K\}$, where $\mathcal{C}_k$ denotes the set of feasible codewords corresponding to dataset $\mathcal{I}_k$. That is, $\mathcal{C}_k$ denotes the set of codewords that may be assigned to the kth worker at each iteration, where each codeword is a linear combination of r gradient estimates that can be computed by this worker.

We would like to highlight that with dynamic codeword assignment, the PS will specify at each iteration which codeword must be computed by each worker. This introduces additional communication requirement compared to the static schemes, such as GC and GC-SC. On the other hand, this information can be piggybacked on other control information that must be communicated from the PS to the workers at each iteration, such as signalling the end of an iteration and the transmission of the updated model parameters. However, it is still important to keep this additional information minimal by designing a codebook with minimal $|\mathcal{C}_k|$.

At the beginning of each iteration t, codeword assignment policy Π_a is executed by the PS based on the past straggler behavior of the workers up to iteration t, $\mathbf{S}^{[t-1]}$, i.e.,

$$\Pi_a^{(t)}(\mathbf{S}^{[t-1]}, \Pi_d) : \mathcal{C} \mapsto \mathbf{c}^t = \left\{c_1^t, \dots, c_K^t\right\}, \tag{9}$$

where $c_k^t \in \mathcal{C}_k$ is the codeword assigned to the kth worker at iteration t and $\mathbf{S}^{[t-1]} \triangleq (\mathbf{S}^1, \dots, \mathbf{S}^{t-1})$, while $\mathbf{S}^t = (S_1^t, \dots, S_K^t)$ denotes the straggler behavior at each iteration t, where $S_k^t = 0$ if the kth worker is a straggler at iteration t, and $S_k^t = 1$ otherwise.[3] [4]

The completion time of iteration t for a given data assignment policy Π_d depends on the codeword assignment $\mathbf{c}_t$ and the straggler realization $\mathbf{S}^t$. Here, our objective is to minimize the expected completion time of each iteration based on the past straggler behavior for a given Π_d:

$$\min_{\Pi_a^{(t)}} \mathbb{E}_{\mathbf{S}^t|\mathbf{S}^{[t-1]},\Pi_d} Q(\mathbf{c}^t, \mathbf{S}^t), \tag{10}$$

where $Q(\mathbf{c}^t, \mathbf{S}^t)$ is the completion time of iteration t under codeword assignment $\mathbf{c}_t$ and the straggler realization $\mathbf{S}^t$.

We remark that the codeword assignment policy $\Pi_a^{(t)}$ highly depends on the data assignment policy Π_d since in most of the coded computation scenarios the data assignment policy is driven by the employed coding strategy. Thus, designing a data assignment policy Π_d without any prior knowledge on the coding strategy is a challenging task. To this end, in the next section, we reformulate the dynamic codeword assignment problem where the coding strategy, consequently the set of codewords, are fixed at the beginning and data assignment is performed based on the underlying coding strategy.

[3]In this work, we assume an on/off straggling behavior for each worker such that a worker's straggling status can change over iterations. Workers can still deliver computation results in the straggling state but their computations are much slower. This type of two-state straggling behavior is observed in empirical studies over Amazon EC2 clusters [18], [27].

[4]Here, in order to manage the complexity of the codeword assignment policy Π_a, especially when t is large, one can determine a number $\bar{t}$ such that $\bar{t} < t$ and consider the straggling behavior of the workers in the last $\bar{t}$ iterations, i.e., $\mathbf{S}^{t-\bar{t}}, \dots, \mathbf{S}^{t-1}$ in making the codeword assignments in iteration t instead of $\mathbf{S}^{[t-1]}$.

IV. GC WITH DYNAMIC CLUSTERING (GC-DC)

In this section, we reformulate the dynamic codeword assignment problem, and introduce the GC-DC scheme. For the construction of the GC-DC scheme, we perform three steps; namely, codeword construction, codeword distribution, and dynamic clustering, where the first two steps are executed once at the beginning of training and the last one is executed at each iteration. Our code construction will be based on GC-SC presented in Section II-B, and we will transform the dynamic codeword assignment problem into a dynamic clustering problem. We note that the number of clusters P is fixed and decided at the beginning of the training.

A. Codeword Construction

In the GC-DC scheme, we will request each worker to compute and return a codeword at each iteration. Remember that each codeword is a specified linear combination of the gradient estimates for a subset of r mini-batches, and the PS and the workers need to agree on how to form these linear combinations in advance. Here, the set of codewords $\mathcal{C}$ is a union of smaller disjoint codeword sets, i.e., $\mathcal{C} = \bigcup_{p=1}^{P} \mathcal{C}^p$, such that the codewords in each set $\mathcal{C}^p$, $p \in [P]$, are encoded and decoded independently and correspond to a particular cluster. For example, in (7), $\mathcal{C}^1 = \{c_{1,1}, c_{1,2}, c_{1,3}\}$, where $\mathcal{C}^1$ is disjoint from the rest of the codeword set.

B. Codeword Distribution

The codewords in $\mathcal{C}$ are distributed among the workers according to a policy Π_c, i.e.,

$$\Pi_c(\mathcal{C}) : \mathcal{C} \mapsto \{\mathcal{C}_1, \ldots, \mathcal{C}_K\}, \tag{11}$$

where we remark that $\mathcal{C}_k$ denotes the set of codewords that can be assigned to the kth worker at each iteration. Now, let $\mathcal{I}(c) \subseteq \mathcal{D}$ be the minimal subset of mini-batches that is sufficient to construct codeword c, where we have $|\mathcal{I}(c)| \leq r$. Given the codeword distribution policy Π_c, any feasible data assignment policy Π_d should satisfy the following constraint

$$\mathcal{I}_k \supseteq \bigcup_{c \in \mathcal{C}_k} \mathcal{I}(c), \quad \forall k \in [K]. \tag{12}$$

Based on this constraint, we observe that, given $\Pi_c(\mathcal{C})$, the minimum memory is used when $\mathcal{I}_k = \bigcup_{c \in \mathcal{C}_k} \mathcal{I}(c), \forall k \in [K]$. Thus, we note that the data assignment policy Π_d is determined according to the codeword distribution policy Π_c. In other words, we first perform codeword distribution and then assign the corresponding mini-batches to the workers.

Next, we describe the codeword distribution policy Π_c in (11) for the proposed GC-DC scheme. We first assign each worker to n clusters. Each cluster p corresponds to a set of codewords $\mathcal{C}^p$ with $|\mathcal{C}^p| = \ell$. We say that a worker is in cluster p, if that worker is assigned all ℓ codewords in $\mathcal{C}^p$. Hence, in the proposed scheme, each worker is assigned codewords from an n-subset of $\{\mathcal{C}^1, \ldots, \mathcal{C}^P\}$.[5] With this, we form a worker cluster assignment matrix $\mathbf{A}_{cluster}$ of size $\ell n \times P$. The pth column of $\mathbf{A}_{cluster}$ shows the workers assigned to the pth cluster, where w_k denotes the kth worker, $k \in [K]$. An example $\mathbf{A}_{cluster}$ for our continuing example is given in (13) for $n = 2$,

$$\mathbf{A}_{cluster} = \begin{bmatrix} w_1 & w_2 & w_3 & w_4 \\ w_6 & w_7 & w_8 & w_5 \\ w_9 & w_{10} & w_{11} & w_{12} \\ w_4 & w_1 & w_2 & w_3 \\ w_7 & w_8 & w_5 & w_6 \\ w_{10} & w_{11} & w_{12} & w_9 \end{bmatrix}. \tag{13}$$

When assigning workers to clusters, we start by dividing workers into groups of P according to their indices. For example, in our continuing example for $P = 4$ and $K = 12$, these groups are $\{w_1, \ldots, w_4\}$, $\{w_5, \ldots, w_8\}$, and $\{w_9, \ldots, w_{12}\}$. Then, we utilize a circular shift operator and sample n shift amounts in $\{0, \ldots, P-1\}$ uniformly at random without replacement for each of these groups. We circularly shift each of these groups according to the corresponding sampled shift amounts and form the worker cluster assignment matrix $\mathbf{A}_{cluster}$. For example, in the first and fourth rows of (13), the shift amounts for workers $\{w_1, \ldots, w_4\}$ are 0 and 1, respectively. As a result of these random shifts, worker w_1 is assigned to the first and second clusters, worker w_2 is assigned to the second and third clusters, and so on. Similarly, from the second and fifth rows of (13), we observe that the shift amounts for workers $\{w_5, \ldots, w_8\}$ are 3 and 2, respectively. We note that, since the random shifts for the same set of workers, e.g., workers $\{w_1, \ldots, w_4\}$, are sampled without replacement, each worker is assigned to exactly $n = 2$ distinct clusters.

We remark that, given n, the memory requirement m of the proposed GC-DC scheme is given by $m = n\ell$. Thus, for $n = 2$ and $\ell = 3$, each worker stores 6 mini-batches in this example.

By constructing $\mathbf{A}_{cluster}$, we essentially perform the codeword distribution as each worker is assigned all ℓ codewords for each of the n clusters that it is associated with. For example, from (13) we deduce that worker 1 has all the codewords in sets $\mathcal{C}^1$ and $\mathcal{C}^2$, i.e., $\mathcal{C}_1 = \mathcal{C}^1 \cup \mathcal{C}^2 = \{c_{1,1}, c_{1,2}, c_{1,3}, c_{2,1}, c_{2,2}, c_{2,3}\}$. With this, we perform the data assignment and assign corresponding mini-batches to each worker to form the data assignment matrix such that the constraint in (12) is satisfied with equality. Correspondingly, $\mathcal{I}_1 = \{\mathcal{D}_1, \ldots, \mathcal{D}_6\}$ so that worker 1 can compute partial gradients $g_1, \ldots, g_6$ to form any one of these 6 codewords.

C. Dynamic Clustering

The key idea behind dynamic clustering is to associate each worker to more than one cluster by assigning more than r mini-batches to each worker. Assuming that a worker is associated with n clusters, each worker is assigned a total of $n\ell$ codewords so that a worker can replace any worker in the n clusters it is associated with by computing a codeword that would be computed by the worker to be replaced in the original GC scheme with clustering. Then, at each iteration the PS

[5] That is, under the proposed GC-DC scheme, we have $|\mathcal{C}_k| = n\ell$ such that each worker may be assigned all ℓ codewords for each of the clusters that it belongs to.

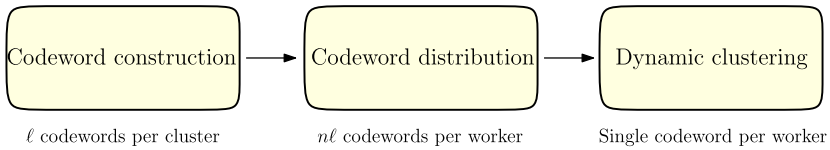


Fig. 2. Three steps of the proposed GC-DC scheme. Codeword construction and codeword distribution are executed only once at the beginning of training whereas dynamic clustering is executed at each iteration.

selects one of the $n\ell$ codewords for each worker based on the previous straggler realization through a codeword assignment policy Π_a given in (9). We note that, even though more than one codeword is assigned to each worker, computation load is still r as in the original GC scheme, and each worker still computes only one codeword consisting of r partial gradient computations at each iteration.

To see the benefit of the proposed GC-DC scheme, we consider $\mathbf{A}_{cluster}$ and corresponding codewords for a particular straggler realization $\mathbf{S} = [1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1]$, where, colors follow the cluster assignment in the static clustering case, i.e., $\mathbf{A}_{cluster}$ given in (5). Under the GC-SC scheme, it is not possible to recover partial gradients corresponding to the third cluster as we do not have $\ell - r + 1 = 2$ non-straggling workers in that cluster.[6] Moreover, if this straggling behavior persists for a substantial duration of time, the overall computation time will suffer drastically. To mitigate this, in the case of dynamic clustering, we observe in (13) that worker w_5 can replace worker w_3 since it can compute codeword $c_{3,1}$ which is the codeword that was originally assigned to worker w_3 in (7) in the GC-SC scheme. This does not affect the recoverability of the partial gradients assigned to the fourth cluster, to which worker w_5 initially belongs, since that cluster has 2 more non-straggling workers, workers w_4 and w_{12}. Further, worker w_2 can replace worker w_8 so that all partial gradients can be recovered successfully. Equivalently, we have assigned the clusters such that non-straggling workers w_2 and w_5 now belong to the 3rd cluster by ensuring that all other clusters still have at least $\ell - r + 1 = 2$ non-straggling workers. Thus, dynamic clustering increases the set of straggler realizations for which the full gradient recovery is possible compared to static clustering.

Since each worker can replace any worker in all the n clusters that it is assigned to, we essentially form the clusters, dynamically at each iteration through codeword assignments, hence the name *dynamic clustering*. That is, based on the codeword distribution presented in Section IV-B, we can assign ℓ workers to each cluster according to the given worker cluster assignment matrix $\mathbf{A}_{cluster}$ without explicitly stating which worker will compute which codeword. With this, our aim is to dynamically form clusters at each iteration to minimize the average completion time of an iteration given the past straggler behavior and the worker-cluster assignment matrix $\mathbf{A}_{cluster}$.

[6]We note that this is the case assuming straggling workers do not return any computation results. Even if they do, whenever there are less than $\ell - r + 1$ non-straggling workers in a cluster, the PS has to wait for at least one of the straggling workers to return its computation which may incur a significant delay in the completion time of that iteration.

Next, we characterize the average completion time of an iteration for a given cluster assignment. We denote the kth smallest of random variables $Y_1, \ldots, Y_n$ as $Y_{k:n}$. The completion time of iteration t for cluster p is given by the time the PS receives the earliest $\ell - r + 1$ results from that cluster such that

$$Q^p(\mathbf{c}^t, \mathbf{S}^t) = \{X^p_{1,r}, \ldots, X^p_{\ell,r}\}_{\ell-r+1:\ell}, \quad p \in [P], \tag{14}$$

where $\mathbf{c}^t$ is the set of codewords assigned to the workers as in (9) and $X^p_{k,r}$, $k \in [\ell]$, is the computation duration of the kth worker of cluster p, i.e., the time it takes for that worker to compute r partial gradients. Noting that iteration t ends when each cluster recovers its corresponding partial gradients, completion time of iteration t is given by

$$Q(\mathbf{c}^t, \mathbf{S}^t) = \max_{p \in [P]} Q^p(\mathbf{c}^t, \mathbf{S}^t). \tag{15}$$

Since some of the workers are stragglers, computation capabilities of the workers are not identical. In this case, minimizing the iteration completion time given in (15) through cluster assignments is not an analytically tractable problem. Instead, in the next section, we propose a greedy dynamic clustering strategy that aims to uniformly place stragglers across clusters at each iteration to speed up GC.

Before we close this section, in Fig. 2, we summarize the steps of the proposed GC-DC scheme. The scheme starts with creating ℓ codewords for each cluster during the codeword construction step. Next, in the codeword distribution step, each worker is assigned to n distinct clusters so that each worker is assigned the corresponding mini-batches for all of the $n\ell$ codewords. Finally, in the dynamic clustering step, each worker is assigned a single codeword at each iteration among these $n\ell$ possible codewords, essentially reforming the clusters at each iteration.

V. Greedy Dynamic Clustering Strategy

In line with the observations on Amazon EC2 instances in [18], [27], in this section, we consider a stochastic straggling behavior for the workers. In particular, we assume that workers' computation statistics are independent from each other, and follow a two-state Markov process. That is, at each iteration a worker can be either in a straggling or a non-straggling state. Once a worker starts straggling, it operates significantly slower than the non-straggling performance and remains straggling for a while. This may model an increased load at a worker for a period of time, which reduces the computational resources that can be allocated for the specific computation task. Our proposed greedy algorithm utilizes this time-correlated straggling behavior to assign straggling

workers to different clusters. At each iteration, the PS identifies the stragglers based on the past observations and implements a greedy dynamic clustering strategy to uniformly distribute the stragglers across clusters to improve the completion time of each iteration. We note that the performance gain of the proposed GC-DC scheme is prominent when the computation speeds of the workers are not identically distributed over iterations, e.g., they exhibit time-correlated straggling behavior, as the GC-DC scheme gains from adapting to the straggling behavior by carefully placing the workers to clusters at each iteration.

Inspired by the bin packing problem [44], we consider clusters as bins and workers as balls as in Fig. 1. Unlike the bin packing problem, which aims to place balls of different volumes into a minimum number of bins of finite volume, in our setting, the number of bins (clusters) is fixed and our aim is to distribute the straggling workers as uniformly as possible to clusters using the worker cluster assignment matrix $\mathbf{A}_{cluster}$. Our dynamic clustering algorithm has two phases: in the first phase, based on the previous straggler realization, we place straggler and non-straggler workers into clusters separately following a specific order, and in the second phase, any placement conflict that may happen in the first phase (i.e., if a worker cannot be placed into any of the remaining clusters) is resolved through worker swap between the corresponding clusters. During worker placement, clusters take turns based on a specified order and we implement a greedy policy such that, once its turn comes, each cluster selects the first available worker that can be assigned to that cluster based on the given worker cluster assignment matrix $\mathbf{A}_{cluster}$.

In what follows we describe in detail the proposed dynamic clustering strategy, which is also presented in Algorithm 1. Given the worker cluster assignment matrix $\mathbf{A}_{cluster}$, without loss of generality, we first reorder workers in each cluster according to their indices such that

$$\mathbf{A}_{cluster}(i,p) < \mathbf{A}_{cluster}(j,p), \quad i < j, \ p \in [P], \tag{16}$$

where $\mathbf{A}_{cluster}(i,p)$ denotes the index of the worker in the ith position in cluster p. For example, in $\mathbf{A}_{cluster}$ given in (13), $\mathbf{A}_{cluster}(1,2)$ is 2 since it corresponds to worker w_2. Once its turn comes, each cluster starts selecting workers with the lowest indices first. We note that, if the workers have heterogeneous computing capabilities, then in this step we order workers according to their speed of computation, such that the fastest workers are selected first, which we will consider in Section VI-B. For ease of exposition, here, we provide the algorithm when all the straggling workers have identical computation statistics, and similarly all the non-straggling workers have the same computation statistics with each other. Therefore, there is no preference among workers within each group, and ordering them according to their indices is appropriate.

We assume that at the end of each iteration, each worker accurately detects its straggling status and informs the PS using an instantaneous feedback. The straggling state information is in general not available to the worker before that iteration ends due to the unpredictable and highly varying nature of computing resources in distributed computing systems.

Algorithm 1 Proposed Dynamic Clustering Strategy

1: Given $\mathbf{A}_{cluster}$, K, P, n, $\mathbf{S}^0$ such that w.l.o.g. $\mathbf{A}_{cluster}(i,p) < \mathbf{A}_{cluster}(j,p)$ for $i < j$, $p \in [P]$
2: **for** $t = 1, \ldots, T$ **do**
3: Observe $\mathbf{S}^{t-1}$ and deduce $\mathcal{K}_f$ and $\mathcal{K}_s$, i.e., sets of non-straggling and straggling workers in iteration $t-1$
4: **Phase I:**
5: Place workers to clusters following an order
6: **if** $|\mathcal{K}_f| \geq |\mathcal{K}_s|$ **then**
7: Place non-stragglers first
8: **else**
9: Place stragglers first
10: **Phase II:**
11: Conflict resolution in the case of an assignment problem in Phase I
12: **Order determination:**
13: $O_f(p) < O_f(\bar{p})$ if $|\mathcal{K}_f^p| < |\mathcal{K}_f^{\bar{p}}|$ or ($|\mathcal{K}_f^p| = |\mathcal{K}_f^{\bar{p}}|$ and $p < \bar{p}$) for $p, \bar{p} \in [P]$
14: Use O_s in the case of straggler placement with $\mathcal{K}_s^p$ for $p \in [P]$
15: **Non-straggler placement:**
16: $i = 1$
17: **while** $|\mathcal{K}_f| > 0$ and $i < M$ **do**
18: $j = \mod(i, P)$ with $j \leftarrow P$ when $\mod(i, P) = 0$
19: Cluster to assign is $\bar{p}$ such that $O_f(\bar{p}) = j$
20: **if** $size(\text{cluster } \bar{p}) < \ell$ **then**
21: Assign the first non-straggling worker from $\mathbf{A}_{cluster}(:, \bar{p})$ to cluster $\bar{p}$
22: Remove the assigned worker from $\mathcal{K}_f$ and $\mathbf{A}_{cluster}$
23: $i = i + 1$
24: **Straggler placement:**
25: Follow steps 16-23 using $\mathcal{K}_s$ and O_s
26: **Conflict resolution:**
27: Given a conflicted worker k and corresponding conflicted cluster p
28: Identify the clusters $\mathcal{P}_k$ that worker k can be assigned to such that $|\mathcal{P}_k| = n$
29: $i = 1$
30: **while** Worker k is not assigned to any cluster **do**
31: Select cluster $\bar{p}$ such that $\bar{p} = \mathcal{P}_k(i)$
32: **if** There is a worker $\bar{k}$ in cluster $\bar{p}$ such that $w_{\bar{k}} \in \mathbf{A}_{cluster}(:, p)$ **then**
33: Assign worker $\bar{k}$ to cluster p
34: Assign worker k to cluster $\bar{p}$
35: $i = i + 1$

Since the current straggling behavior is random following the underlying Markov process, at iteration t, the algorithm starts by deducing the sets of non-straggling and straggling workers $\mathcal{K}_f$ and $\mathcal{K}_s$ from $\mathbf{S}^{t-1}$. We note that, at each iteration, $\mathcal{K}_f \cup \mathcal{K}_s = [K]$. The proposed algorithm uses the straggler statistics from iteration $t-1$ to perform dynamic clustering at iteration t, which makes this algorithm suitable for Markovian straggling models.

A. Phase I—Worker Placement

We place straggling and non-straggling workers separately to the clusters following a specific order. If the number of non-straggling workers is higher than the stragglers, i.e., $|\mathcal{K}_f| \geq |\mathcal{K}_s|$, we start by placing the non-stragglers and vice-versa.

For the sake of demonstration, we assume $|\mathcal{K}_f| \geq |\mathcal{K}_s|$ and place the non-straggling workers first. Let O_f denote the order in which the clusters select workers such that $O_f(p)$ gives the order in which the pth cluster selects workers. To determine the exact order, we define $\mathcal{K}_f^p$ and $\mathcal{K}_s^p$, which denote the set of non-straggling and straggling nodes that can be assigned to cluster p, respectively. We remark that worker k can be assigned to cluster p if it is in column p of $\mathbf{A}_{cluster}$, i.e., $w_k \in \mathbf{A}_{cluster}(:, p)$. With this, we determine the order vector such that

$$O_f(p) < O_f(\bar{p}) \text{ if } |\mathcal{K}_f^p| < |\mathcal{K}_f^{\bar{p}}| p, \quad \bar{p} \in [P]. \tag{17}$$

That is, clusters with less availability select workers first. In the case of equal availability, i.e., $|\mathcal{K}_f^p| = |\mathcal{K}_f^{\bar{p}}|$, cluster with the smaller index selects first, i.e., $O_f(p) < O_f(\bar{p})$ for $p < \bar{p}$. The order for straggler placement O_s is determined accordingly using $\mathcal{K}_s^p$, for $p \in [P]$.

Once the order O_f is determined, non-straggling workers are placed into clusters following O_f. As stated in lines 16-23 of Algorithm 1, once its turn comes, each cluster p with an open spot, i.e., each cluster p that currently has less than ℓ workers, selects the first available non-straggling worker from $\mathbf{A}_{cluster}(:, p)$, $p \in [P]$. Once a non-straggling worker is assigned to a cluster, we remove it from $\mathcal{K}_f$ and $\mathbf{A}_{cluster}$. We note that this assignment continues until there is no unassigned non-straggling worker left in $\mathcal{K}_f$ or a placement conflict is observed. Then, the straggler workers are placed following a similar procedure with the order vector O_s.

During Phase I, the algorithm makes at most M such placement attempts, where $M > 0$ is a sufficiently large number. If after M turns, a worker cannot be assigned to any of the remaining clusters, this indicates a placement conflict and we move on to the second phase of the algorithm.

B. Phase II—Conflict Resolution

Assume that there is a placement conflict at the end of Phase I such that worker k cannot be placed to the remaining cluster p. That is, all of the n clusters that worker k can be assigned to are full, i.e., already have ℓ workers, and cluster p needs one more worker. In such a case, the second conflict resolution phase of the algorithm starts.

Let $\mathcal{P}_k$ denote the set of possible clusters for worker k such that $|\mathcal{P}_k| = n$. In the conflict resolution step, as stated in lines 26-35 of Algorithm 1, we look for a worker $\bar{k}$, which has been assigned to one of the clusters in $\mathcal{P}_k$ in Phase I such that $w_{\bar{k}} \in \mathbf{A}_{cluster}(:, p)$. That is, even though worker $\bar{k}$ has been assigned to cluster $\bar{p} \in \mathcal{P}_k$ during Phase I, it can be assigned to cluster p as well. Once we detect first such worker, we swap its position with worker k. That is, we assign worker k, the conflicted worker, to cluster $\bar{p}$ and worker $\bar{k}$ to cluster p, the conflicted cluster.

We note that there might be multiple placement conflicts at the end of Phase I, in which case the conflict resolution step is repeated until all cases are resolved.

To illustrate the proposed worker replacement policy in detail, we consider the cluster assignment matrix in (13), and without loss of generality, order workers in an increasing index order in each column to obtain

$$\mathbf{A}_{cluster} = \begin{bmatrix} w_1 & w_1 & w_2 & w_3 \\ w_4 & w_2 & w_3 & w_4 \\ w_6 & w_7 & w_5 & w_5 \\ w_7 & w_8 & w_8 & w_6 \\ w_9 & w_{10} & w_{11} & w_9 \\ w_{10} & w_{11} & w_{12} & w_{12} \end{bmatrix}, \tag{18}$$

where the straggling workers are shown in red. The straggler realization for this example is $\mathbf{S} = [1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1]$. Here, we have 5 straggling and 7 non-straggling workers, i.e., $|\mathcal{K}_s| = 5$ and $|\mathcal{K}_f| = 7$.

Since there are more non-straggling workers than stragglers, we place the non-straggling workers first. To determine a non-straggling worker placement order, we find the number of available non-straggling workers in each cluster. One can observe in (18) that, cluster 1 and cluster 2 have 4 available non-straggling workers that can be assigned to these clusters whereas cluster 3 and cluster 4 have 3 available non-straggling workers. That is, we have $|\mathcal{K}_f^1| = |\mathcal{K}_f^2| = 4$ and $|\mathcal{K}_f^3| = |\mathcal{K}_f^4| = 3$. Based on these, we deduce a placement order $O_f = [3, 4, 1, 2]$ such that clusters take turns based on this placement order.[7] At each turn of a particular cluster, a single worker is assigned to that cluster according to the aforementioned greedy policy. In our example, we start with the third cluster and w_2 is assigned to this cluster. Then, the fourth cluster gets w_4 and so on. This process continues until all the non-straggling workers are placed into clusters (or until a placement conflict is observed). If a cluster is assigned $\ell = 3$ workers, we say that cluster is full and do not assign any more workers to that cluster. Next, we determine the placement order of straggling workers in a similar fashion. One can deduce from (18) that the order of placement for the stragglers is $O_s = [1, 2, 3, 4]$ as clusters 1 and 2 have the least availability. Based on this order, stragglers are also placed using the greedy policy described above and the first phase terminates with the worker placement shown in Fig. 3. Here, we observe a placement conflict as w_{12} has not been assigned to any cluster whereas cluster 1 needs one more worker, but w_{12} cannot be assigned there.

We start the second phase of the proposed worker placement algorithm to place w_{12} into a cluster that has a worker which can be assigned to the first cluster. We see from (18) that w_{12} can be assigned to clusters 3 or 4. None of the workers which has been assigned to cluster 3 in Phase I can be assigned to the first cluster. Then, the algorithm looks as cluster 4 and identifies that w_4, which has been assigned to the fourth cluster in the first phase, can go to the first cluster. With this, we swap

[7] In a more refined implementation, this order can dynamically change after each round of worker placement, i.e., after all clusters select one worker, to better reflect the clusters with less availability as worker placement continues.

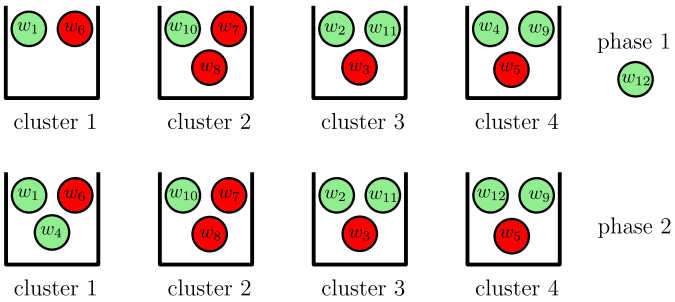


Fig. 3. The proposed worker placement strategy.

workers w_4 and w_{12}, which yields the final placement in Fig. 3.

At the end of the algorithm we see that the stragglers are placed into the clusters as uniformly as possible: cluster 2 has two stragglers while the remaining clusters have only 1 straggler each. We note that since we have only 7 non-straggling workers, less than the worst case scenario of $P(\ell - r + 1) = 8$ non-stragglers, the full recovery is not possible for the static clustering scheme. Thus, the proposed dynamic clustering scheme does not improve the worst case scenario. Rather, it speeds up the GC scheme by uniformly placing the stragglers across clusters. This process is repeated at each iteration to dynamically change the clusters based on the straggler observations.

We note that at the end of the first phase, there are 4 other workers, namely workers w_4, w_7, w_9, and w_{10}, that can be placed into the first cluster, which had placement conflict at the end of Phase I of the algorithm. Even if $\ell = 3$ of them would have been assigned to cluster 2, which worker w_{12} cannot be assigned, the remaining one of them still would have been assigned to either cluster 3 or 4. Thus, it is guaranteed that cluster 3 and cluster 4 have at least one worker that can be assigned to cluster 1 so that the placement conflict can be resolved. The next lemma formally states this guarantee.

Lemma 1: Assume that we have a conflicted worker k which cannot be assigned to the remaining cluster p in Phase I. Then, if

$$n > \frac{P(K-1)}{2K}, \tag{19}$$

it is guaranteed that at least one worker in one of the clusters in $\mathcal{P}_k$ can be assigned to cluster p so that the placement conflict can be resolved.

Proof: In the proof we consider the worst case scenario such that $\ell - 1$ workers have already been assigned to cluster p in Phase I. Thus, in the remaining $P - 1$ clusters other than cluster p, there are $n\ell - \ell + 1$ workers that can be assigned to cluster p. We want to make sure that, at the end of Phase I of the algorithm, at least one of those workers is assigned to a cluster in set $\mathcal{P}_k$, which, as previously stated, denotes the set of clusters that worker k, the conflicted worker, can be assigned to. Except cluster p, there are $P - n - 1$ clusters that worker k cannot be assigned to. These $P - n - 1$ clusters can at most have $(P - n - 1)\ell$ workers after Phase I. Thus, as long as $n\ell - \ell + 1 > (P - n - 1)\ell$, there is at least one worker that can be assigned to cluster p in one of the clusters in $\mathcal{P}_k$, which yields (19) since $\ell = \frac{K}{P}$. ■

In the previous example, (19) is satisfied since $K = 12$, $P = 4$, and $n = 2$ such that $n > \frac{11}{6}$.

In the next section, we analyze the performance of this dynamic clustering strategy through numerical simulations.

VI. Numerical Results

In this section, we provide numerical results comparing the proposed GC-DC scheme with GC-SC as well as the original GC scheme using a model-based scenario for computation latencies.[8] For the simulations, we consider a linear regression problem over synthetically created training and test datasets, as in [7], of sizes 2000 and 400, respectively. We set the size of the model to $d = 1000$. A single simulation consists of $T = 400$ iterations. For all the simulations, we use learning rate $\eta = 0.1$. To model the computation delays at the workers, we adopt the commonly used shifted exponential model [45], and assume that the probability of completing r partial gradient computations at worker k by time t is given by

$$\mathbb{P}[X_{k,r} \leq t] \triangleq \begin{cases} 1 - e^{-\mu_k(\frac{t}{r} - \alpha_k)}, & \text{if } t \geq r\alpha_k, \\ 0, & \text{otherwise}, \end{cases} \tag{20}$$

where $\alpha_k > 0$ is a constant shift indicating that a single computation duration cannot be smaller than α_k and $\mu_k > 0$ denotes the straggling effect. We consider two different models for the time-correlated straggling behavior: the homogeneous and heterogeneous worker models, which we discuss next.

A. Gilbert-Elliot Model With Homogeneous Workers

We model the straggling behavior of the 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. Specifically, in (20) we have rate μ_f in state f and rate μ_s in state s, where $\mu_f > \mu_s$ as in [25], [46]. That is, each worker has two possible rates based on its straggling statistics. We assume that the state transitions only occur at the beginning of each iteration with probability p; that is, with probability $1 - p$ the state remains the same. A low switching probability p indicates that the straggling behavior tends to remain the same in consecutive iterations with occasional transitions. We set $p = 0.05$ or $p = 0.2$, $\alpha = 0.01$, $\mu_s = 0.1$, and $\mu_f = 10$. We assume that the transition probability p along with the computation rates μ_s and μ_f are known to the PS. At the end of each iteration, workers inform the PS regarding their straggling status before the next iteration starts. With this information along with the knowledge of transition probability p, the PS performs the dynamic clustering accordingly. For example, when $p \leq 0.5$, the PS assumes that each worker will continue with the same straggling behavior from the past iteration.[9]

[8] The fractional repetition scheme can also be used for GC in our work in addition to the cyclic GC we use as a baseline. However, the proposed cyclic GC scheme is preferred as it does not impose any constraint on the (K, r) pairs.

[9] After a sufficiently long observation period, the PS can accurately estimate the transition probability p as it is the same for all the workers and iterations.

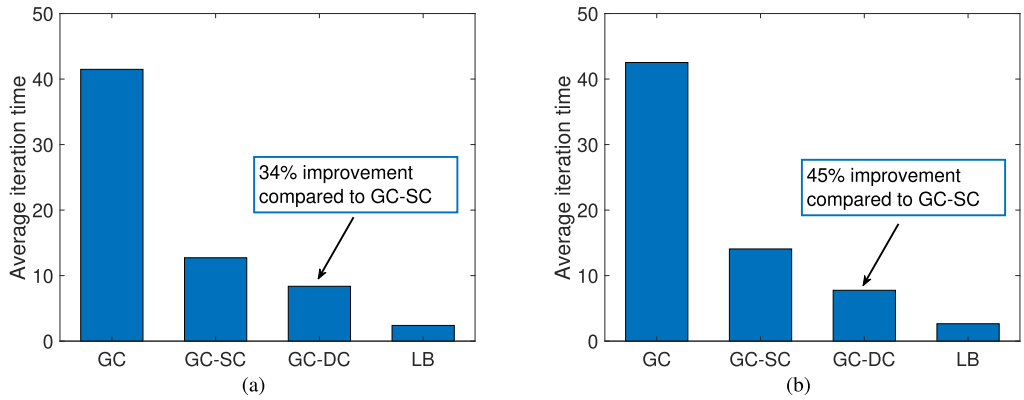


Fig. 4. Average per-iteration completion time under the Gilbert-Elliot model with homogeneous workers for $K = 20$, $P = 5$, $r = 3$, $n = 3$, and $p = 0.05$ (a) under imperfect SSI, (b) under perfect SSI.

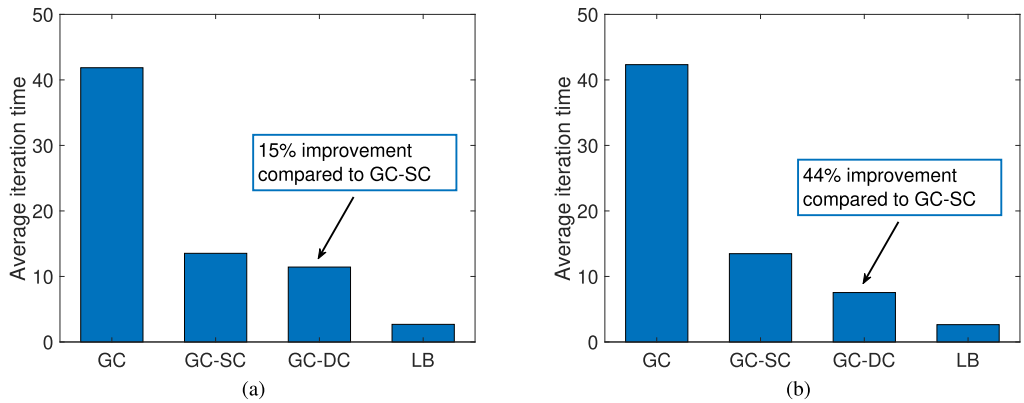


Fig. 5. Average per-iteration completion time under the Gilbert-Elliot model with homogeneous workers for $K = 20$, $P = 5$, $r = 3$, $n = 3$, and $p = 0.2$ (a) under imperfect SSI, (b) under perfect SSI.

In the first simulation, we set $K = 20$, $P = 5$, $r = 3$, and $n = 3$. We start with 10 stragglers initially. In Fig. 4, we plot the average per-iteration completion time of the original GC scheme, GC scheme with static clustering (GC-SC), GC scheme with the proposed dynamic clustering (GC-DC), and a lower bound, denoted by LB. Here, the lower bound is obtained by assuming that the full gradient is recovered as soon as the earliest $P \times (\ell - r + 1)$ workers finish their computations at each iteration, independently of the codeword assignment matrix. We remark that this lower bound is rather an idealistic scenario as it requires the perfect knowledge of computation times at each iteration as well as $n = P$, i.e., all workers can be assigned to all the clusters. We observe in Fig. 4(a) that clustering schemes significantly improve the performance compared to the original GC scheme. The best performance is achieved when the dynamic clustering, the GC-DC scheme, is implemented and the performance improvement compared to the GC-SC scheme is approximately 34%.

In Fig. 4(a), we have considered the case in which the PS does not know the exact straggler realization at the beginning of an iteration, and uses previous observation to implement the dynamic clustering strategy. In the second simulation in Fig. 4(b), we consider the same scenario as in the first simulation, but assume that the PS knows the exact straggler realization at the beginning of each iteration, which we call perfect straggler state information (SSI). That is, in the case of perfect SSI, the PS knows exactly which workers will straggle in the current iteration, and therefore, the proposed dynamic clustering algorithm does not suffer from transitions in the straggling behavior from one iteration to the next. In this case we see similar trends as in Fig. 4(a), but observe that the GC-DC scheme results in a larger improvement in the average per-iteration completion time (around 45%) than that of the imperfect SSI case.

In Fig. 5 we consider a case in which the straggler state transitions occur more frequently and set $p = 0.2$. We see in Fig. 5(a) that under imperfect SSI, with a larger p value, GC-DC still performs the best, but the improvement over GC-SC is less compared to Fig. 4(a) when $p = 0.05$. On the other hand, under perfect SSI, i.e., the PS knows the exact straggler realization at the beginning of each iteration, the effect of increased p is not observed and we have approximately 45% improvement over GC-SC as in Fig. 4(b).

Next, in Fig. 6, we consider a larger system that consists of $K = 100$ workers. Since the number of workers is higher, we set $P = 10$ such that workers are divided into 10 clusters and we set $r = 6$, $n = 6$, and $p = 0.2$. We see in both Figs. 6(a) and (b) that the proposed GC-DC scheme outperforms the static GC schemes under both imperfect and perfect SSI conditions, respectively.

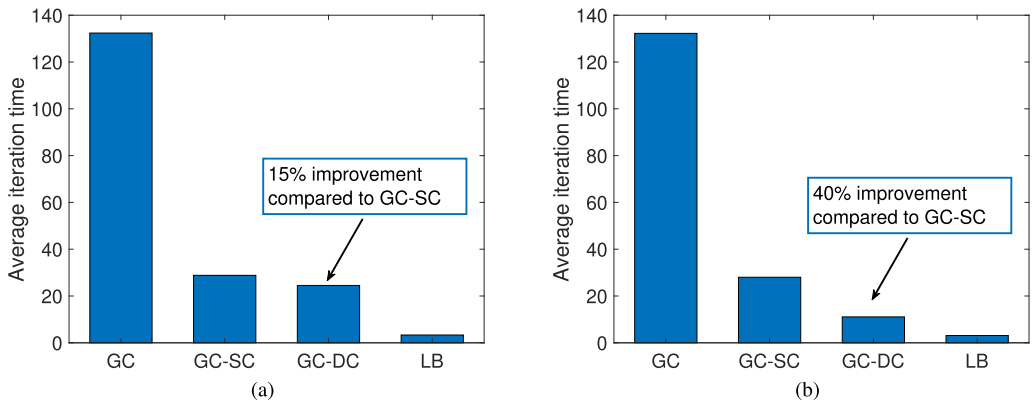


Fig. 6. Average per-iteration completion time under the Gilbert-Elliot model with homogeneous workers for $K = 100$, $P = 10$, $r = 6$, $n = 6$, and $p = 0.2$ (a) under imperfect SSI, (b) under perfect SSI.

TABLE I
THE PERFORMANCE OF THE GC-SC AND GC-DC SCHEMES AS A FUNCTION OF P FOR $K = 100$, $r = 10$, $n = P$, AND $p = 0.05$

	$P = 1$	$P = 2$	$P = 4$	$P = 5$	$P = 10$
GC-SC	166.81	113.18	58.20	37.23	0.95
GC-DC	166.81	111.73	50.33	23.28	0.70

Finally, in Table I, we study the effect of the number of clusters P on the average iteration time in clustering schemes, i.e., GC-SC and the proposed GC-DC scheme. We have $K = 100$ workers and set $r = 10$ and $n = P$. In Table I, we observe that when we have a single cluster, i.e., $P = 1$, both schemes yield the same performance. We note that the performance of the GC scheme is independent of the number of clusters and is equal to 166.81 in this case for all P values. Table I shows that the proposed GC-DC scheme outperforms the GC-SC scheme for all P values and the performance of either scheme improves as P increases even though each worker computes r partial gradients for all P values. This is because the number of workers in a cluster decreases as P increases so that each cluster is more likely to finish its assigned partial gradient computations faster.

B. Heterogeneous Worker Model

In this model, we assume that workers have different computation rates μ_k, $k \in [K]$. In this case, we specify a straggling threshold $\tau > 0$, and a worker k is treated as a straggler if $\mu_k < \tau$.

1) Gilbert-Elliot Model With Heterogeneous Workers: We study the case in which each worker's straggling behavior is modeled by a two-state Markov chain such that $\mu_k = \mu_{k,f}$ if worker k is not straggling and $\mu_k = \mu_{k,s}$ if worker k is a straggler. At the beginning of each iteration, a worker's straggling mode switches with probability p. Here, first we sample the non-straggling computation rates of each worker $\mu_{k,f}$ uniformly at random from the interval $[0, 5]$ and set $\alpha_k = 0.01$, $p = 0.05$ or $p = 0.2$, for $k \in [K]$. We model the straggling computation rates of workers $\mu_{k,s}$ such that for worker k we have $\mu_{k,s} = \frac{\mu_{k,f}}{10}$, $k \in [K]$. That is, in the straggling mode, each worker is $10\times$ slower than its typical non-straggling performance, which is motivated by the measurements taken over Amazon EC2 clusters that indicate a similar performance drop in the straggling mode [18]. With this, computation rates of the workers in the straggling mode are uniformly distributed in $[0, 0.5]$. We assume that the non-straggling computation rates $\mu_{k,f}$ are known to the PS for $k \in [K]$ after a certain number of iterations and from these, the PS can deduce the straggling computation rates $\mu_{k,s}$.

Equipped with these, after each iteration, the PS is informed about the straggling status of each worker and performs the proposed greedy dynamic clustering scheme with a modification as follows: Instead of ordering the workers according to (16), we order them according to their rates μ_k, $k \in [K]$. In this case, once its turn comes, each cluster selects the fastest available worker first rather than selecting the one with the smallest index first.

We note that since the computation rates are sampled randomly, a worker's straggling computation rate can still be higher than another worker's non-straggling rate. To account for these scenarios, we set the straggling threshold $\tau = 0.5$. That is, as long as a worker's rate is below 0.5 we treat that worker as a straggler. We did not utilize such a threshold in the homogeneous worker model since in that case workers have identical computation rates μ_f and μ_s in the non-straggling and straggling states, respectively, such that $\mu_s < \mu_f$.

Simulation results for this setup are provided in Fig. 7. We average the results over 30 independent simulations for a fixed $\mathbf{A}_{cluster}$ that is generated according to the procedure described in Section IV-B. We observe in Figs 7(a) and 7(b) for $p = 0.05$, and in Figs 8(a) and 8(b) for $p = 0.2$ that the GC-DC scheme outperforms the static clustering schemes, namely GC and GC-SC. The performance improvement is larger in the case of perfect SSI and when $p = 0.05$.

2) Heterogeneous Workers With Time-Varying Rates: So far, we have modeled the straggling behavior based on a Gilbert-Elliot mode. In this subsection, instead of a two-state Markov chain model, we consider that the straggling parameters of the workers are time-varying. We assume that each worker samples its rate uniformly at random from the interval $[0, 5]$ and set $\alpha_k = 0.01$ for all $k \in [K]$. We assume that at the

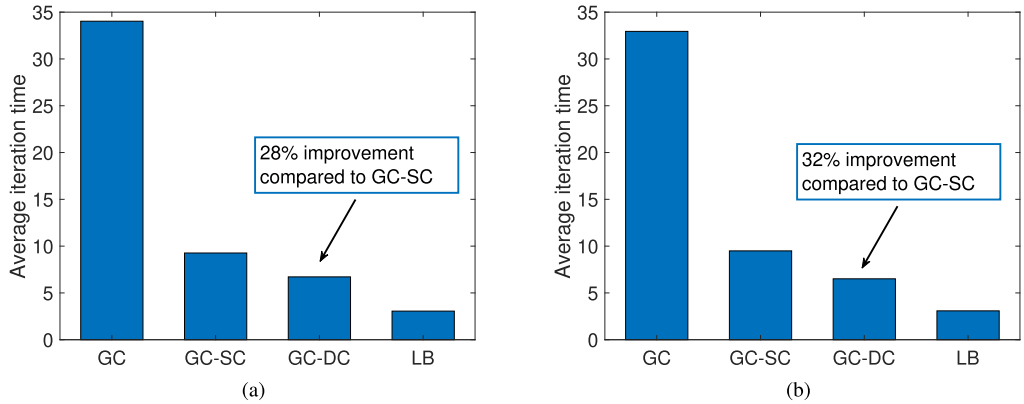


Fig. 7. Average per-iteration completion time under the Gilbert-Elliot model with heterogeneous workers for $K = 20$, $P = 5$, $r = 3$, $n = 3$, $p = 0.05$, and $\tau = 0.5$ (a) under imperfect SSI, (b) under perfect SSI.

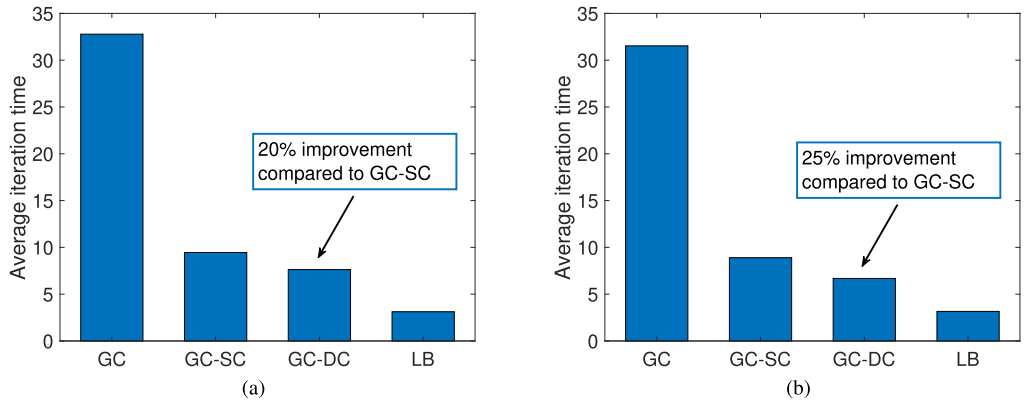


Fig. 8. Average per-iteration completion time under the Gilbert-Elliot model with heterogeneous workers for $K = 20$, $P = 5$, $r = 3$, $n = 3$, $p = 0.2$, and $\tau = 0.5$ (a) under imperfect SSI, (b) under perfect SSI.

beginning of each iteration, each worker re-samples its rate with probability p such that with probability $1 - p$ its rate stays the same. That is, we have

$$\mu_{k,t+1} = (1 - a_{t+1})\mu_{k,t} + a_{t+1} \cdot U[0, 5], \tag{21}$$

where, $\mu_{k,t}$ denotes the rate of worker k at iteration t, a_t is an i.i.d. Bernoulli(p) random variable, i.e., $\mathbb{P}(a_t = 1) = p, \forall t$, and $U[a, b]$ denotes a uniform random variable over interval $[a, b]$. In simulations, we use the scenario in the Fig. 4 and start with 10 stragglers. We initialize the rates of stragglers with $\mu_{k,0} = U[0, \tau)$ and rates of non-straggling workers with $\mu_{k,0} = U[\tau, 5]$. In this setup, we set $p = 0.05$ or $p = 0.2$.

Since the computation capabilities of the workers are not identical, we apply the proposed greedy dynamic clustering scheme with the same modification as above. We note that this model requires the workers to accurately detect their computation rates at the end of each iteration and send them to the PS before the next iteration starts.

First, we consider the case in which $\tau = 1$. In this case, we observe in Figs. 9(a) and (b) that the GC-DC scheme outperforms the GC and GC-SC schemes but the improvement compared to the GC-SC scheme is not significant. In fact, we see that in the case of perfect SSI the improvement is around 20% compared to the GC-SC scheme whereas when the straggler realizations are not known to the PS in advance this improvement drops to approximately 16%.

Next, we set $\tau = 0.1$ such that the proposed greedy dynamic clustering scheme specifically targets the slowest workers and carefully places them across clusters. In Figs. 10(a) and (b), we observe for $p = 0.05$ that the GC-DC scheme performs the best and the improvement compared to the GC-SC scheme is more significant. We also note that in Fig. 10, the performance improvement is larger but the average iteration times are also larger for all three schemes compared to the case in Fig. 9. This is because when $\tau = 0.1$, we initialize the rates of the workers considering $10\times$ slower stragglers compared to when $\tau = 1$. We set $p = 0.2$ in Fig. 11, and observe that the GC-DC scheme still performs the best compared to the static GC schemes even though the improvement in performance decreases as the transitions occur more frequently compared to the case in which $p = 0.05$. We finally note that all the simulation results given in Figs. 9, 10, and 11 are averaged over 30 independent simulations for a fixed $\mathbf{A}_{cluster}$ that is generated according to the procedure described in Section IV-B.

C. Simulations for Shared Access Scenario

In general, the concept of a straggler refers to a state in which a worker either does not respond at all or responds

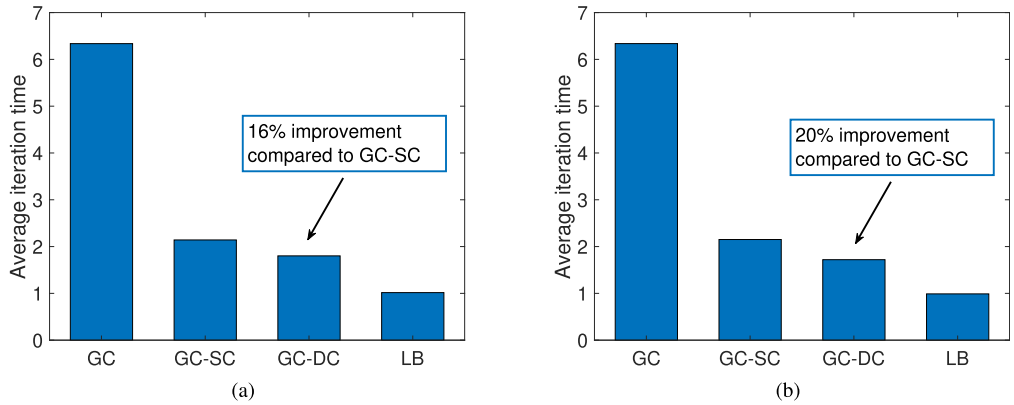


Fig. 9. Average per-iteration completion time under the heterogeneous worker model with time-varying rates for $K = 20$, $P = 5$, $r = 3$, $n = 3$, $p = 0.05$, and $\tau = 1$ (a) under imperfect SSI, (b) under perfect SSI.

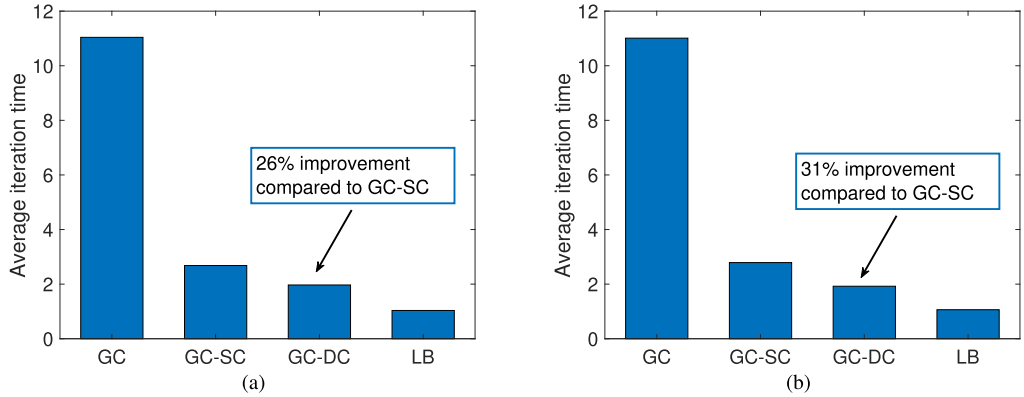


Fig. 10. Average per-iteration completion time under the heterogeneous worker model with time-varying rates for $K = 20$, $P = 5$, $r = 3$, $n = 3$, $p = 0.05$, and $\tau = 0.1$ (a) under imperfect SSI, (b) under perfect SSI.

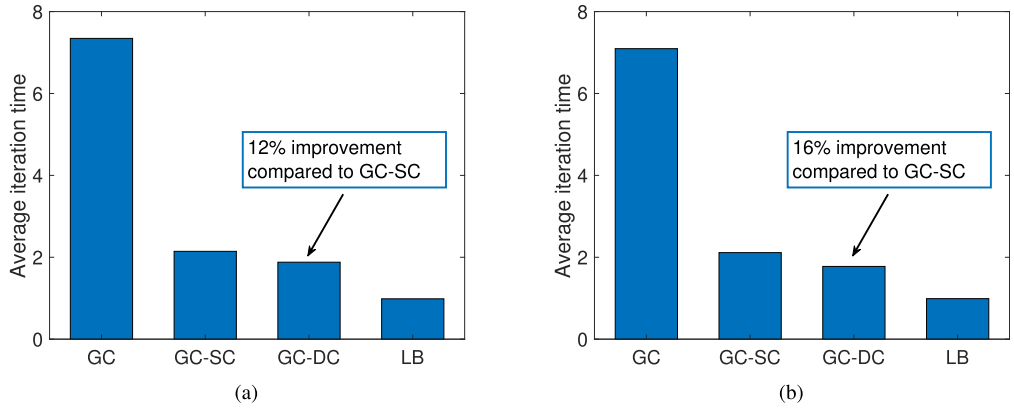


Fig. 11. Average per-iteration completion time under the heterogeneous worker model with time-varying rates for $K = 20$, $P = 5$, $r = 3$, $n = 3$, $p = 0.2$, and $\tau = 0.1$ (a) under imperfect SSI, (b) under perfect SSI.

with a certain delay. The delay mentioned here might be due to several factors, such as the internal delay of the processing units, communication delay due to possible link failures, or due to overloading of the workers. The latter is often observed when the computational resources are open to access from multiple users without any central entity to regulate the user requests or to perform resource allocation. In such cases, workers with excessive user requests might be identified as stragglers due to their long response time. To this end, we perform experiments to analyze the response time of the workers with respect to the number of ongoing computational requests. The experiments are conducted in our clusters where GeForce RTX 2080 Ti Graphics Cards with CUDA toolkit 11.0 are employed as workers. In the experiments, we consider training of ResNet-20 architecture for classification on CIFAR-10 dataset with a batch size of

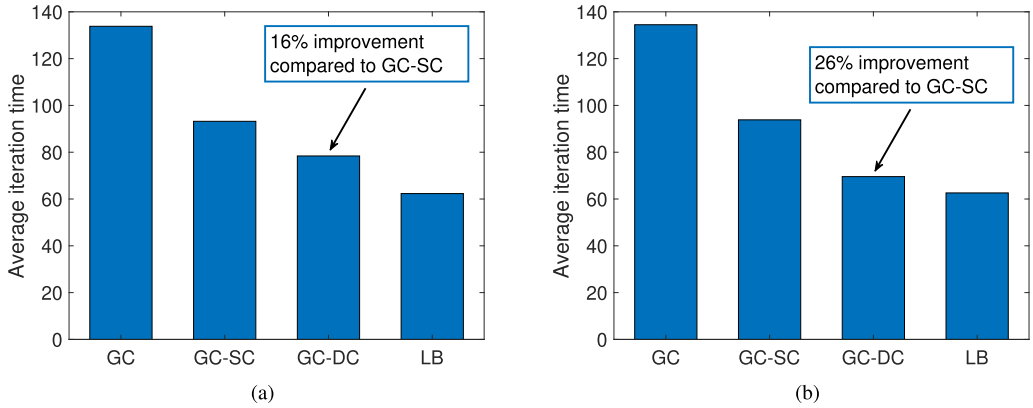


Fig. 12. Average per-iteration completion time under the shared access model for $K = 20$, $P = 5$, $r = 3$, $n = 3$, $\lambda = 0.1$, $\gamma = 7$, $C = 10$ (a) under imperfect SSI, (b) under perfect SSI.

64 as the computational task. To measure the impact of the workload on the workers, we generate multiple users with the same computational task and measure the latency for a single iteration with SGD framework implemented with PyTorch. Consequently, we observe that, when there is a single request, the completion time of a single iteration is 20 milliseconds (ms), however, this latency increases linearly with the number of ongoing tasks. Based on this observation, we simulate a scenario in which the users arrive at the system according to a certain probabilistic model and stay in the system for a random period of time depending on the complexity of the task.

Here, we assume that requests arrive at the workers according to a Poisson distribution with rate $\lambda = 0.1$. Each arriving task stays in the system for a number of iterations that is sampled uniformly at random from the interval $[2, 5]$. We assume that each worker can serve at most $C = 10$ users at a time. This means that, for a single worker, a single iteration may take between 20 ms and 200 ms based on our measurements. We denote the straggling threshold in this case by γ and set $\gamma = 7$. That is, a worker is considered straggling if it has 7 or more ongoing task computations. Simulations results for this setup are provided in Fig. 12. These results are averaged over 30 independent simulations for a fixed $\mathbf{A}_{cluster}$. In Fig. 12, we observe that the proposed GC-DC scheme outperforms the static GC schemes under this more practical setup. The improvement is more visible in the case of perfect SSI, i.e., when the PS knows the number of ongoing computations at each worker. In this setup, the performance of the GC-DC scheme is much closer to LB compared to the results in Section VI. Especially in the perfect SSI case, the gap is less than 10%, which indicates that the proposed GC-DC algorithm promises to improve the performance in more practical shared access scenarios over computing clusters.

VII. Discussion and Conclusion

In this work, we considered coded computing for large-scale distributed learning problems in the presence of straggling workers, and introduced a novel scheme, called GC-DC, to reduce the average per-iteration completion time of the static GC schemes. GC-DC employs the GC scheme with clustering introduced in [32], and assigns additional data to the workers without increasing the per-iteration computation load at each worker compared to the original GC scheme. By utilizing the extra degree-of-freedom offered by additional data, but without increasing the computation load at each iteration, the proposed GC-DC scheme dynamically assigns workers to different clusters at each iteration, in order to distribute the stragglers to clusters as uniformly as possible. Under a time-correlated straggler model, GC-DC can improve the overall computation speed by dynamically adapting to the straggling behavior. We showed through numerical simulations, for both homogeneous and heterogeneous worker models, that the proposed GC-DC scheme can drastically improve the average per-iteration completion time without an increase in the communication load.

We would like to highlight that the proposed redundant data assignment approach with dynamic computations is a fairly general paradigm, and the proposed cluster-based GC approach is only one of many possible coding techniques that can be employed. A possible future research direction is considering heterogeneous cluster sizes. The proposed model assumes that number of workers in each cluster is fixed. That is, cluster sizes are equal to $\ell = \frac{K}{P}$. One can consider varying the cluster sizes to further decrease the average iteration time. Also, in the proposed technique, workers are assigned to clusters based on an order that does not change during the assignment process. To improve the performance, one can consider adaptively changing this worker assignment order. Another potential research direction is to consider a more complex straggling behaviour across the workers, such as non-Markovian, Markovian with higher memory, or correlated straggling behaviour across workers. Such models would require considering all the past straggling behavior when making dynamic clustering assignments, and reinforcement learning techniques can be employed to find the policy that chooses the best code or best clustering strategy to be used at each iteration.

References

[1] B. Buyukates, E. Ozfatura, S. Ulukus, and D. Gunduz, "Gradient coding with dynamic clustering for straggler mitigation," in *Proc. IEEE ICC*, Jun. 2021, pp. 1–6.

[2] M. Chen *et al.*, "Distributed learning in wireless networks: Recent progress and future challenges," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 12, pp. 3579–3605, Oct. 2021.

[3] J. S. Ng *et al.*, "A survey of coded distributed computing," 2020, *arXiv:2008.09048*.

[4] 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, Mar. 2018.

[5] N. Ferdinand and S. C. Draper, "Hierarchical coded computation," in *Proc. IEEE ISIT*, Jun. 2018, pp. 1620–1624.

[6] R. K. Maity, A. S. Rawat, and A. Mazumdar, "Robust gradient descent via moment encoding with LDPC codes," in *Proc. IEEE ISIT*, Jul. 2019, pp. 2734–2738.

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

[8] M. Fahim, H. Jeong, F. Haddadpour, S. Dutta, V. Cadambe, and P. Grover, "On the optimal recovery threshold of coded matrix multiplication," in *Proc. Allerton Conf.*, Oct. 2017, pp. 1264–1270.

[9] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Polynomial codes: An optimal design for high-dimensional coded matrix multiplication," in *Proc. NIPS*, Dec. 2017, pp. 1–11.

[10] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Straggler mitigation in distributed matrix multiplication: Fundamental limits and optimal coding," in *Proc. IEEE ISIT*, Jun. 2018, pp. 2022–2026.

[11] S. Dutta, Z. Bai, H. Jeong, T. M. Low, and P. A. Grover, "A unified coded deep neural network training strategy based on generalized polydot codes," in *Proc. IEEE ISIT*, Jun. 2018, pp. 1585–1589.

[12] H. Park, K. Lee, J. Sohn, C. Suh, and J. Moon, "Hierarchical coding for distributed computing," in *Proc. IEEE ISIT*, Jun. 2018, pp. 1630–1634.

[13] S. Kiani, N. Ferdinand, and S. C. Draper, "Exploitation of stragglers in coded computation," in *Proc. IEEE ISIT*, Jun. 2018, pp. 1988–1992.

[14] A. B. Das, L. Tang, and A. Ramamoorthy, "C^3LES: Codes for coded computation that leverage stragglers," in *Proc. IEEE ITW*, Nov. 2018, pp. 1–5.

[15] E. Ozfatura, S. Ulukus, and D. Gündüz, "Distributed gradient descent with coded partial gradient computations," in *Proc. IEEE ICASSP*, May 2019, pp. 3492–3496.

[16] A. Mallick, M. Chaudhari, and G. Joshi, "Fast and efficient distributed matrix-vector multiplication using rateless fountain codes," in *Proc. IEEE ICASSP*, May 2019, pp. 8192–8196.

[17] E. Ozfatura, D. Gündüz, and S. Ulukus, "Speeding up distributed gradient descent by utilizing non-persistent stragglers," in *Proc. IEEE ISIT*, Jul. 2019, pp. 2729–2733.

[18] C. S. Yang, R. Pedarsani, and A. S. Avestimehr, "Timely coded computing," in *Proc. IEEE ISIT*, Jul. 2019, pp. 2798–2802.

[19] Y. Yang, M. Interlandi, P. Grover, S. Kar, S. Amizadeh, and M. Weimer, "Coded elastic computing," in *Proc. IEEE ISIT*, Jul. 2019, pp. 2654–2658.

[20] H. Park and J. Moon, "Irregular product coded computation for high-dimensional matrix multiplication," in *Proc. IEEE ISIT*, Jul. 2019, pp. 1782–1786.

[21] R. Bitar, Y. Xing, Y. Keshtkarjahromi, V. Dasari, S. El Rouayheb, and H. Seferoglu, "Private and rateless adaptive coded matrix-vector multiplication," 2019, *arXiv:1909.12611*.

[22] Y. Sun, J. Zhao, and D. Gunduz, "Heterogeneous coded computation across heterogeneous workers," in *Proc. IEEE Globecom*, Dec. 2019, pp. 1–6.

[23] B. Hasircioglu, J. Gomez-Vilardebo, and D. Gündüz, "Bivariate polynomial coding for efficient distributed matrix multiplication," *IEEE J. Sel. Areas Inf. Theory*, vol. 2, no. 3, pp. 814–829, Sep. 2021.

[24] B. Buyukates and S. Ulukus, "Timely distributed computation with stragglers," *IEEE Trans. Commun.*, vol. 68, no. 9, pp. 5273–5282, Jun. 2020.

[25] E. Ozfatura, B. Buyukates, D. Gündüz, and S. Ulukus, "Age-based coded computation for bias reduction in distributed learning," in *Proc. IEEE Globecom*, Dec. 2020, pp. 1–6.

[26] S. Dutta, M. Fahim, F. Haddadpour, H. Jeong, V. Cadambe, and P. Grover, "On the optimal recovery threshold of coded matrix multiplication," 2018, *arXiv:1801.10292*.

[27] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient coding: Avoiding stragglers in distributed learning," in *Proc. ICML*, Aug. 2017, pp. 3368–3376.

[28] M. Ye and E. Abbe, "Communication-computation efficient gradient coding," in *Proc. ICML*, Jul. 2018, pp. 5610–5619.

[29] W. Halbawi, N. Azizan, F. Salehi, and B. Hassibi, "Improving distributed gradient descent using Reed–Solomon codes," in *Proc. IEEE ISIT*, Jun. 2018, pp. 2027–2031.

[30] J. Zhang and O. Simeone, "LAGC: Lazily aggregated gradient coding for straggler-tolerant and communication-efficient distributed learning," 2019, *arXiv:1905.09148*.

[31] S. Kadhe, O. O. Koyluoglu, and K. Ramchandran, "Gradient coding based on block designs for mitigating adversarial stragglers," in *Proc. IEEE ISIT*, Jul. 2019, pp. 2813–2817.

[32] E. Ozfatura, D. Gündüz, and S. Ulukus, "Gradient coding with clustering and multi-message communication," in *Proc. IEEE Data Sci. Workshop*, Jun. 2019, pp. 42–46.

[33] H. Wang, S. Guo, B. Tang, R. Li, and C. Li, "Heterogeneity-aware gradient coding for straggler tolerance," in *Proc. IEEE ICDCS*, Jul. 2019, pp. 555–564.

[34] L. Tauz and L. Dolecek, "Multi-message gradient coding for utilizing non-persistent stragglers," in *Proc. Asilomar Conf.*, Nov. 2019, pp. 2154–2159.

[35] R. Bitar, M. Wootters, and S. El Rouayheb, "Stochastic gradient coding for straggler mitigation in distributed learning," *IEEE J. Sel. Areas Inf. Theory*, vol. 1, no. 1, pp. 277–291, May 2020.

[36] N. Charalambides, M. Pilanci, and A. O. Hero, "Weighted gradient coding with leverage score sampling," in *Proc. IEEE ICASSP*, May 2020, pp. 5215–5219.

[37] N. Raviv, I. Tamo, R. Tandon, and A. G. Dimakis, "Gradient coding from cyclic MDS codes and expander graphs," *IEEE Trans. Inf. Theory*, vol. 66, no. 12, pp. 7475–7489, Dec. 2020.

[38] J. Chen, X. Pan, R. Monga, S. Bengio, and R. Jozefowicz, "Revisiting distributed synchronous SGD," 2016, *arXiv:1604.00981*.

[39] S. Li, S. M. M. Kalan, A. S. Avestimehr, and M. Soltanolkotabi, "Near-optimal straggler mitigation for distributed gradient methods," in *Proc. IEEE IPDPS*, May 2018, pp. 857–866.

[40] 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 *Proc. AISTATS*, Apr. 2018, pp. 803–812.

[41] N. Ferdinand and S. C. A. Draper, "Anytime stochastic gradient descent: Time to hear from all the workers," in *Proc. Allerton Conf.*, Oct. 2018, pp. 552–559.

[42] A. Behrouzi-Far and E. Soljanin, "On the effect of task-to-worker assignment in distributed computing systems with stragglers," in *Proc. Allerton Conf.*, Oct. 2018, pp. 560–566.

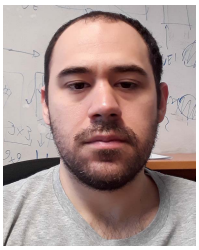
[43] M. M. Amiri and D. Gündüz, "Computation scheduling for distributed machine learning with straggling workers," *IEEE Trans. Signal Process.*, vol. 67, no. 24, pp. 6270–6284, Dec. 2019.

[44] B. Korte and J. Vygen, *Combinatorial Optimization: Theory Algorithms*. Berlin, Germany: Springer, 2008.

[45] E. Ozfatura, S. Ulukus, and D. Gündüz, "Straggler-aware distributed learning: Communication computation latency trade-off," *Entropy, Special Issue Interplay Storage, Comput., Commun. Inf.-Theoretic Perspective*, vol. 22, no. 5, p. 544, May 2020.

[46] B. Buyukates and S. Ulukus, "Age of information with Gilbert-Elliot servers and samplers," in *Proc. CISS*, Mar. 2020, pp. 1–6.

Baturalp Buyukates (Member, IEEE) received the B.S. degree from the Department of Electrical and Electronics Engineering, Bilkent University, Turkey, in 2016, and the M.S. and Ph.D. degrees from the Department of Electrical and Computer Engineering, University of Maryland, College Park, MD, USA, in 2020 and 2022, respectively. He is currently a Post-Doctoral Research Associate with the Information Theory and Machine Learning (vITAL) Research Laboratory, University of Southern California. His research interests include communication theory, machine learning, and networks, with a recent focus on federated learning, age of information, and distributed computation.



Emre Ozfatura (Member, IEEE) received the B.Sc. degree in electronics engineering (math minor) and the M.Sc. degree in electronics engineering from Sabanci University, Turkey, in 2012 and 2015, respectively, and the Ph.D. degree from the Department of Electrical and Electronic Engineering, Imperial College London, U.K., in 2021. He is currently a Post-Doctoral Research Associate with the Information Processing and Communications (IPC) Laboratory, Imperial College London. His research interests are video streaming applications, distributed content storage, distributed computation, federated learning, and robust learning.

Sennur Ulukus (Fellow, IEEE) received the B.S. and M.S. degrees in electrical and electronics engineering from Bilkent University and the Ph.D. degree in electrical and computer engineering from the Wireless Information Network Laboratory (WINLAB), Rutgers University.

She is currently an Anthony Ephremides Professor in information sciences and systems with the Department of Electrical and Computer Engineering, University of Maryland at College Park, where she also holds a joint appointment with the Institute for Systems Research (ISR). Prior to joining UMD, she was a Senior Technical Staff Member at AT&T Labs-Research. Her research interests are in information theory, wireless communications, machine learning, and signal processing and networks, with recent focus on private information retrieval, age of information, machine learning for wireless, distributed coded computing, group testing, physical layer security, energy harvesting communications, and wireless energy and information transfer.

Dr. Ulukus is a Distinguished Scholar–Teacher with the University of Maryland. She received the 2003 IEEE Marconi Prize Paper Award in Wireless Communications, the 2019 IEEE Communications Society Best Tutorial Paper Award, the 2020 IEEE Communications Society Women in Communications Engineering (WICE) Outstanding Achievement Award, the 2020 IEEE Communications Society Technical Committee on Green Communications and Computing (TCGCC) Distinguished Technical Achievement Recognition Award, the 2005 NSF CAREER Award, the 2011 ISR Outstanding Systems Engineering Faculty Award, and the 2012 ECE George Corcoran Outstanding Teaching Award. She was a Distinguished Lecturer of the IEEE Information Theory Society from 2018 to 2019. She is the TPC Chair of 2021 IEEE Globecom; and the TPC Co-Chair of 2019 IEEE ITW, 2017 IEEE ISIT, 2016 IEEE Globecom, 2014 IEEE PIMRC, and 2011 IEEE CTW. She has been an Area Editor of the IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS since 2019 and a Senior Editor of the IEEE TRANSACTIONS ON GREEN COMMUNICATIONS AND NETWORKING since 2020. She was an Area Editor of the IEEE TRANSACTIONS ON GREEN COMMUNICATIONS AND NETWORKING (2016–2020), an Editor of the IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS Series on Green Communications and Networking (2015–2016), an Associate Editor of the IEEE TRANSACTIONS ON INFORMATION THEORY (2007–2010), and an Editor of the IEEE TRANSACTIONS ON COMMUNICATIONS (2003–2007). She was a Guest Editor for the IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS (2008, 2015, and 2021), JOURNAL OF COMMUNICATIONS AND NETWORKS (2012), and the IEEE TRANSACTIONS ON INFORMATION THEORY (2011).

Deniz Gündüz (Fellow, IEEE) received the B.S. degree in electrical and electronics engineering from METU, Turkey, in 2002, and the M.S. and Ph.D. degrees in electrical engineering from the NYU Tandon School of Engineering (formerly Polytechnic University) in 2004 and 2007, respectively.

After his Ph.D. degree, he served as a Post-Doctoral Research Associate with Princeton University; a Consulting Assistant Professor with Stanford University; and a Research Associate with CTTC, Barcelona, Spain. In September 2012, he joined the Electrical and Electronic Engineering Department, Imperial College London, U.K., where he is currently a Professor of information processing and serves as the Deputy Head of the Intelligent Systems and Networks Group. He is also a part-time Faculty Member with the University of Modena and Reggio Emilia, Italy, and has held visiting positions at the University of Padova (2018–2020) and Princeton University (2009–2012). His research interests lie in the areas of communications and information theory, machine learning, and privacy. He is a Distinguished Lecturer of the IEEE Information Theory Society (2020–2022). He was a recipient of the Consolidator (2022) and Starting (2016) Grants of the European Research Council (ERC), IEEE Communications Society—Communication Theory Technical Committee (CTTC) Early Achievement Award in 2017, and several best paper awards. He is an Area Editor of the IEEE TRANSACTIONS ON INFORMATION THEORY, IEEE TRANSACTIONS ON COMMUNICATIONS, and the IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS (JSAC), Special Series on Machine Learning in Communications and Networks. He also serves as an Editor for the IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS.