# Hierarchical Broadcast Coding: Expediting Distributed Learning at the Wireless Edge

Dong-Jun Han, *Graduate Student Member, IEEE*, Jy-Yong Sohn, *Member, IEEE*, and Jaekyun Moon, *Fellow, IEEE*

***Abstract*— Distributed learning plays a key role in reducing the training time of modern deep neural networks with massive datasets. In this article, we consider a distributed learning problem where gradient computation is carried out over a number of computing devices at the wireless edge. We propose hierarchical broadcast coding, a provable coding-theoretic framework to speed up distributed learning at the wireless edge. Our contributions are threefold. First, motivated by the hierarchical nature of real-world edge computing systems, we propose a layered code which mitigates the effects of not only packet losses at the wireless computing nodes but also straggling access points (APs) or small base stations. Second, by strategically allocating data partitions to nodes in the overlapping areas between cells, our technique achieves the fundamental lower bound on computational load to combat stragglers. Finally, we take advantage of the broadcast nature of wireless networks by which wireless devices in the overlapping cell coverage broadcast to more than one AP. This further reduces the overall training time in the presence of straggling APs. Experimental results on Amazon EC2 confirm the advantage of the proposed methods in speeding up learning. Our design targets any gradient descent based learning algorithms, including linear/logistic regressions and deep learning.**

***Index Terms*— Distributed learning, gradient descent, wireless edge, stragglers.**

## I. Introduction

TRAINING a large-scale model on a massive dataset is of paramount importance in modern deep learning applications. To reduce the training time of such large-scale learning, extensive distributed learning schemes have been proposed in recent years [1]–[5]. While most current distributed computing/learning systems are based on the cloud (i.e., data centers), edge-facilitated computing [6]–[10] has drawn significant attention nowadays, which is motivated by the proliferation of edge servers and computing devices at the wireless edge. By utilizing mobile or Internet of Things (IoT) devices as computing nodes, which are more accessible for the users than the cloud, edge-facilitated computing enables low-latency applications such as virtual reality.

At the wireless edge, the running time of distributed learning algorithms suffers from two major limitations. First, there are packet losses caused by wireless fading channels [11], which are inevitable when gradient computations are transmitted over-the-air. Packet retransmission is required, which significantly increases overall training time, resulting in a severe form of straggling. Secondly, low-cost IoT devices when used as computing nodes can have very high variability in performance, which would cause unpredictable delays. In order to facilitate distributed learning at the wireless edge, efficient solutions are needed to handle this *straggler* issue which significantly slows down distributed learning.

### A. Related Works

In the context of stragglers, these bottlenecks are mostly studied in cloud-based distributed computing/learning scenarios in a simple master-worker setup [12]–[24]. Especially focusing on distributed learning, the authors of [18] proposed a framework called gradient coding, which introduced coding techniques in synchronous gradient descent to mitigate stragglers. By putting more computational load at the workers, gradient coding enables to recover the exact sum of total gradients at the master. It is shown that gradient coding outperforms the existing approach based on simply ignoring stragglers, which could miss data in each iteration and suffers from degraded performance. The authors of [25] proposed an efficient gradient coding scheme by characterizing trade-offs among computational load, straggler tolerance and communication cost. In [26]–[28], approximate gradient coding schemes are proposed, which aim at reducing the computation load at nodes by allowing the master to compute an approximate gradient. Gradient coding schemes are also proposed to protect distributed training against adversarial nodes [29]. In [30], the authors proposed a tree topology in gradient coding to reduce communication burden at the master. However, all previous works on gradient coding do not reflect the practical environments of the wireless edge and thus could not directly be applied to edge-facilitated learning systems; in practical wireless systems, mobile/IoT nodes are connected wirelessly to access points (APs) (e.g., small base stations), and these APs are connected to a local parameter server (PS), which can be viewed as a less resourceful cloud.

Manuscript received March 24, 2020; revised July 7, 2020 and October 5, 2020; accepted November 19, 2020. Date of publication December 7, 2020; date of current version April 9, 2021. This work was supported by the National Research Foundation of Korea under Grant 2019R1I1A2A02061135. The associate editor coordinating the review of this article and approving it for publication was M. Kaneko. *(Corresponding author: Dong-Jun Han.)*

The authors are with the School of Electrical Engineering, Korea Advanced Institute of Science and Technology (KAIST), Daejeon 34141, South Korea (e-mail: djhan93@kaist.ac.kr; jysohn1108@kaist.ac.kr; jmoon@kaist.edu).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TWC.2020.3040792.

Digital Object Identifier 10.1109/TWC.2020.3040792

1536-1276 © 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: Korea Advanced Inst of Science & Tech - KAIST. Downloaded on April 09,2021 at 01:39:55 UTC from IEEE Xplore. Restrictions apply.

Moreover, a large number mobile/IoT devices are located in the overlapping areas between cells, which comes from dense deployment of APs in 5G and beyond systems. In this hierarchical wireless setup, how to design efficient coding schemes to combat stragglers? How to fully utilize the computing nodes located in the overlapping cell areas to minimize the training time? Existing works on gradient coding do not provide any answers to these questions, which are the major limitations to be applied in edge-facilitated learning systems.

For speeding up matrix multiplication, hierarchical networks as well as wireless networks were considered in [31] and [32], [33]. In [34]–[36], wireless distributed computing schemes are proposed for data shuffling applications. Compared to these works, we consider applying codes across gradients for any given loss function, which were not considered before. Finally, compared to the existing works on federated learning [37]–[40] where the data is collected at individual devices (and thus coding is not applicable), we consider a conventional distributed learning problem where the central PS has the whole dataset and thus enables application of coding when allocating data partitions to the distributed nodes.

## B. Main Contributions

In this article, we focus on mitigating stragglers for distributed learning at the wireless edge. We propose hierarchical broadcast coding, a coding technique highly tailored to the practical environments of the wireless edge with following unique characteristics. First, our scheme reflects the hierarchical nature of edge computing systems [41], [42]; the mobile/IoT nodes are connected to the APs and these APs are connected to the PS. Here, not only the wireless computing nodes but also the APs can become stragglers depending on wired/wireless backhaul conditions, especially when the network traffic at the central PS is high due to congestions [43]. These straggling APs would critically degrade the performance of learning. By hierarchical coding, our scheme enables to tolerate any $s_1$ straggling computing nodes and any $s_2$ straggling APs in the system. Secondly, our scheme utilizes the computing nodes located in the overlapping areas in wireless cellular networks. Especially in today's small cell networks where pico/femto cells [44], [45] are densely deployed, almost all the nodes are located in the overlapping areas between cells. The nodes in the overlapping coverages could download data from more than one APs, a feature that can be exploited in designing codes. By precisely allocating data partitions to the computing nodes in the overlapping cell areas, our scheme achieves the fundamental lower bound on computational load to combat stragglers. The last characteristic of our scheme is to utilize the broadcast nature of wireless networks. The nodes located in the overlapping areas between cells can transmit their result(s) simultaneously to multiple APs by broadcasting. With a properly designed code, this aspect of our system can significantly reduce the training time in the presence of straggling APs. Our main contributions can be summarized as follows:

- We first derive the fundamental lower bound on computational load to combat $s_1$ straggling computing devices and $s_2$ straggling APs in the system, in a hierarchical wireless setup with broadcasting nodes in the overlapping cell areas.
- By strategically allocating data partitions to the nodes in the overlapping cell regions, we propose hierarchical broadcast coding, a scheme that can reduce the training time while achieving the fundamental bound on computational load to combat stragglers.
- Experiments are carried out by implementing the schemes on Amazon EC2 with PyTorch and MPI4py package. Extensive results on 1) ResNet18 with CIFAR-10 dataset and 2) logistic regression models with a dataset from Kaggle show that our scheme achieves the target accuracy much faster than existing methods.

Compared to gradient coding [18] and its related works, the hierarchical setup with a significant number of broadcasting nodes in the overlapping cell areas calls for new design strategies, making our solution unique. It is shown that the proposed scheme tailored to the hierarchical wireless setup show remarkable performance gain compared to the naive application of gradient coding.

*Organization:* Section II provides preliminaries on distributed learning and gradient coding. In Section III, we formulate the distributed learning problem in a wireless setup. Our main result on hierarchical broadcast coding is described in Section IV and experimental results on Amazon EC2 clusters are provided in Section V. Finally, we draw conclusions in Section VI.

*Notations:* Let $[n] := \{1, 2, \dots, n\}$. $\mathbf{1}_{a\times b}$ is an $a \times b$ all-ones matrix.

## II. Preliminaries

In this section, we first describe the basic setup of distributed learning. Then, we describe the concept of gradient coding [18], which is the most closely related work to our proposed idea.

### A. Distributed Learning

In distributed learning, the PS has the whole dataset and distributes across multiple computing nodes for data parallelization. For a given dataset $D = \{(x_i, y_i)\}_{i=1}^m$ with $i$-th data point $x_i \in \mathbb{R}^d$ and corresponding label $y_i \in \mathbb{R}$, we would like to learn the parameters of the model by solving $w^* = \arg\min_{w\in\mathbb{R}^d} \sum_{i=1}^m \ell(x_i, y_i; w)$ where $\ell(\cdot)$ is a loss function. Employing stochastic gradient descent, we first initialize the model at a specific point $w^{(0)}$ and then iteratively update the parameters according to

$$w^{(t+1)} = h(w^{(t)}, g^{(t)}), \tag{1}$$

where $g^{(t)} := \sum_{i=1}^m \nabla\ell(x_i, y_i; w^{(t)})$ is the gradient of the loss for the current model $w^{(t)}$ and $h$ is a gradient-based optimizer. We assume that there are $n$ computing nodes, and that dataset $D$ is partitioned into $k$ subsets of equal size: $\{D_i\}_{i=1}^k$. After computing the gradients for given data partitions, each node sends the sum of the assigned gradients to the PS. The PS

Authorized licensed use limited to: Korea Advanced Inst of Science & Tech - KAIST. Downloaded on April 09,2021 at 01:39:55 UTC from IEEE Xplore. Restrictions apply.

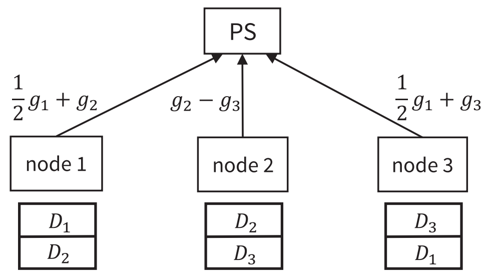


Fig. 1. Example of gradient coding with $k = n = 3$, $s = 1$, and a single parameter server (PS).

aggregates the gradients of all data partitions $\sum_{i=1}^{k} g_i$, where $g_i := \sum_{(x,y)\in D_i} \nabla\ell(x, y; w^{(t)})$ is the sum of gradients corresponding to data partition $D_i$. Finally, the model is updated according to (1) before moving to the next iteration.

### B. Gradient Coding

Gradient coding is a scheme which allows redundant computation at the nodes to mitigate the effect of stragglers in distributed learning. Again, let $n$ be the number of computing nodes connected to a single PS and $k$ be the number of data partitions. Let $r$ be the average number of data partitions assigned to each node, i.e., the computational load. We would like to have the exact sum of gradients $\sum_{i=1}^{k} g_i$ at the PS with $s$ straggling nodes in the system. Then according to [18], in a single-layer master-worker setup, we must have

$$r \geq \frac{k(s+1)}{n}. \tag{2}$$

The equation (2) can be interpreted as follows. In order to tolerate $s$ stragglers, each data partition should be replicated and allocated to at least $s + 1$ nodes. Thus, the number of redundant data partitions is $k(s+1)$ partitions in total, which are distributed into $n$ nodes. This implies that each node should receive $k(s+1)/n$ data partitions, which coincides with (2). Fig. 1 shows an example with $k = n = 3$ and $s = 1$. The exact sum of gradients $\sum_{i=1}^{3} g_i$ can be obtained at the PS by receiving any two out of three results, tolerating any $s = 1$ straggler. This example also achieves the lower bound on computational load $r$ in (2).

## III. System Model: Coded Distributed Learning at the Wireless Edge

In this article, we also consider a distributed learning problem where the PS has the whole dataset, but in a hierarchical wireless setup. Motivated by the recent proliferation of mobile/IoT devices, we utilize the nodes (e.g., mobile/IoT devices) at the wireless edge as computing resources, where the goal is to obtain the exact value of $\sum_{i=1}^{k} g_i$ at PS located at the cloud. We assume that $L$ APs are connected to a single PS. Each AP forms a cell, serving mobile nodes in its covered area. Here, a cell is defined as a service area (i.e., a geometric region) of each AP. Let $u_i$ be the number of nodes which is connected only to the $i$-th AP and to no others. We also denote $v_{i,j}$ as the number of mobile nodes located in the overlapping cell areas between AP $i$ and AP $j$ for $i \neq j$, connecting to both APs. We assume that a node is connected to at most two APs, although our idea can be generalized to connections to more APs, as shown in Appendix A. Now a specific AP $i$ has access to $f_i$ nodes, where $f_i$ can be written as

$$f_i = u_i + \sum_{j\in[L]\setminus\{i\}} v_{i,j}. \tag{3}$$

The total number of nodes in the system becomes

$$n = \sum_{i=1}^{L} u_i + \sum_{i=1}^{L} \sum_{\substack{j\in[L]\\ j<i}} v_{i,j}. \tag{4}$$

An example system is shown in Fig. 2 with $L = 3$, $u_i = u = 2$ for all $i \in [L]$, $v_{i,j} = v = 1$ for all $i, j \in [L]$, $i \neq j$.

The overall procedure for distributed learning is described as follows: First, datasets are distributed across the computing nodes (e.g., mobile/IoT devices) based on a predefined rule. Each node computes the gradients of assigned data partitions and transmits a particular linear combination of the results to corresponding AP(s). In this process, some of the nodes can be subject to severe channel fading and thus become stragglers. Here, a node shared by two APs transmits its result to both APs. After collecting results from the nodes, each AP sends a specific linear combination of the collected results to PS, with some APs possibly being stragglers. The goal is to obtain $\sum_{i=1}^{k} g_i$ at PS with $s_1$ straggling nodes and $s_2$ straggling APs in the system.

In the following, we describe the general framework of hierarchical broadcast coding, which is built upon the well-established gradient coding concepts and tools of [18]. Before training begins, each AP figures out the current channel condition by receiving feedbacks from the computing nodes, and sends the information on current $u_i$, $v_{i,j}$ values to PS. PS first designs an $L \times k$ encoding matrix $Q$ which allocates $k$ data partitions to $L$ cells such that the sum of all gradients can be recovered despite $s_2$ straggling APs, while also letting some data partitions be shared between adjacent cells. Defining $G = [g_1, g_2, \ldots, g_k]^T$ as the $d$-dimensional gradient of each data partition as its row, $q_i G$, the $i$-th row of $QG$, corresponds to the vector that the $i$-th AP transmits to PS. For the example in Fig. 2, we have

$$Q = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & -1 & -1 & -1 \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}, \tag{5}$$

$$QG = \begin{bmatrix} \frac{1}{2}(g_1^T + g_2^T + g_3^T) + g_4^T + g_5^T + g_6^T \\ g_4^T + g_5^T + g_6^T - (g_7^T + g_8^T + g_9^T) \\ \frac{1}{2}(g_1^T + g_2^T + g_3^T) + g_7^T + g_8^T + g_9^T \end{bmatrix}. \tag{6}$$

Based on $Q$, PS also designs an $E \times L$ decoding matrix $P$, where $E$ denotes the number of possible scenarios straggling APs can arise. With $s_2$ straggling APs in the system, we have $E = \binom{L}{s_2}$. The design purpose of $Q$ and $P$ is to recover the sum of gradients at PS against any $E$ straggling AP scenarios;

Authorized licensed use limited to: Korea Advanced Inst of Science & Tech - KAIST. Downloaded on April 09,2021 at 01:39:55 UTC from IEEE Xplore. Restrictions apply.

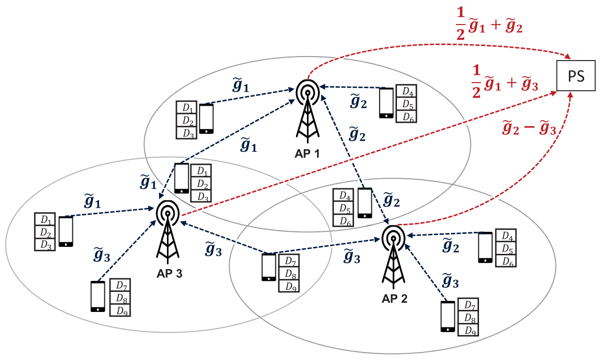


Fig. 2. Example of hierarchical broadcast coding with $L = 3$, $u_i = u = 2$, $v_{i,j} = v = 1$, $s_1 = 1$, $s_2 = 1$. The APs are located at the higher tier of devices, and the parameter server (PS) is located at the higher tier of the APs (i.e., at the cloud). Here, we define $\tilde{g}_1 := g_1 + g_2 + g_3$, $\tilde{g}_2 := g_4 + g_5 + g_6$, $\tilde{g}_3 := g_7 + g_8 + g_9$. By hierarchical coding, the system is able to tolerate $s_1 = 1$ straggling computing node and $s_2 = 1$ straggling AP. By broadcasting, the nodes in the overlapping area can transmit the result to both APs with only one computation and one communication round. The lower bound on the computational load in Theorem 1 is achieved.

this goal can be written as $PQG = \mathbf{1}G$, or equivalently, $PQ = \mathbf{1}_{E\times k}$.

Now we focus on the specific $i$-th cell, where its AP needs to recover the vector $q_iG$ in the presence of $s_1$ straggling nodes. We define a $f_i \times k$ encoding matrix $B^{(i)}$ for allocating data partitions to $f_i$ nodes within the $i$-th cell. The $j$-th row of $B^{(i)}$ is an encoding vector for node $j$ in cell $i$, which linearly combines the corresponding gradients. More specifically, $B^{(i)}G$ becomes a $f_i \times d$ matrix whose $j$-th row corresponds to the result that node $j$ in cell $i$ transmits to the $i$-th AP. For example, considering AP 1 in Fig. 2, we have

$$B^{(1)} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}, \tag{7}$$

$$B^{(1)}G = \begin{bmatrix} g_1^T + g_2^T + g_3^T \\ g_1^T + g_2^T + g_3^T \\ g_4^T + g_5^T + g_6^T \\ g_4^T + g_5^T + g_6^T \end{bmatrix}. \tag{8}$$

Here, we note that the nodes in the overlapping areas between cells would be given two encoded vectors from both APs. When the encoded matrices are precisely designed so that a node has the same encoded vectors for both cells, it can simply broadcast the result to the corresponding APs after computation. This requires one round of computing gradients and one round of communication by *broadcasting*. When a node is assigned with the same data partitions but different encoding vectors from both cells, one round of computation and two rounds of communication are required. When the assigned data partitions from both cells are different, two rounds of computation and communication are required, which is the worst case. Minimizing the computation and communication round at the nodes is an important design criterion for $B^{(i)}$.

Now consider an $F_i \times f_i$ decoding matrix $A^{(i)}$ which aggregates the computational results $B^{(i)}G$ of $f_i$ nodes in cell $i$ and recovers the desired vector $q_iG$. Here, $F_i$ is the number of possible straggling scenarios for the nodes in cell $i$; since we consider combating arbitrary $s_1$ straggling nodes in each cell, we have $F_i = \binom{f_i}{s_1}$. Each row of $A^{(i)}$ corresponds to a specific straggling scenario we want to combat for obtaining $q_iG$. Since we want to recover $q_iG$ for any of the $F_i$ scenarios, the desired condition at the $i$-th AP reduces to $A^{(i)}B^{(i)}G = [q_i; q_i; \ldots; q_i]G$, or equivalently, $A^{(i)}B^{(i)} = [q_i; q_i; \ldots; q_i]$. For example, considering the $F_1 = 4$ straggling scenarios with $s_1 = 1$ in AP 1 of Fig. 2, we can write

$$A^{(1)} = \begin{bmatrix} 0 & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & 0 & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{4} & \frac{1}{4} & 0 & 1 \\ \frac{1}{4} & \frac{1}{4} & 1 & 0 \end{bmatrix}, \tag{9}$$

$$A^{(1)}B^{(1)} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & 1 & 1 & 1 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & 1 & 1 & 1 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & 1 & 1 & 1 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} q_1 \\ q_1 \\ q_1 \\ q_1 \end{bmatrix}. \tag{10}$$

Throughout the paper, we denote $Q$ and $P$ as encoding and decoding matrices of the outer code, respectively. Likewise, we denote $\{B^{(i)}\} := \{B^{(1)}, \ldots, B^{(L)}\}$ and $\{A^{(i)}\} := \{A^{(1)}, \ldots, A^{(L)}\}$ as the encoding and decoding matrices of the inner codes. To sum up, for a given $L$, $u_i$, $v_{i,j}$ and a cell geometry, our goal is to design a scheme $\left(P, Q, \{A^{(i)}\}, \{B^{(i)}\}\right)$ such that $PQ = \mathbf{1}_{E\times k}$ and $A^{(i)}B^{(i)} = [q_i; q_i; \ldots; q_i]$ hold for all $i \in [L]$, which is robust to any $s_1$ straggling nodes and any $s_2$ straggling APs in the system.

## IV. Main Result: Hierarchical Broadcast Coding

In this section, we present our main results for the coding scheme. Our goal is to exactly recover the true sum of gradients when $s_1$, $s_2$, $u_i$, $v_{i,j}$ and $L$ are given. By defining computational load $r$ as the average number of data partitions assigned to each node, we first state the following theorem which provides the lower bound on computational load to combat stragglers.

*Theorem 1: Suppose that a scheme* $\left(P, Q, \{A^{(i)}\}, \{B^{(i)}\}\right)$ *is robust to any $s_1$ straggling computing nodes*[1] *and $s_2$ straggling APs in the system with given $u_i$, $v_{i,j}$, $L$ and $k$. With every node computing the same amount of data, the computational load must satisfy*

$$r \geq \frac{k(s_1+1)(s_2+1)}{\sum_{i=1}^{L} f_i}, \tag{11}$$

*where $f_i$ is the number of computing nodes connected to the $i$-th AP as defined in (3).*

[1]Our hierarchical coding is designed to combat any $s_1$ straggling computing nodes for every cell, which surely guarantees tolerance against $s_1$ straggling nodes in the system.

Authorized licensed use limited to: Korea Advanced Inst of Science & Tech - KAIST. Downloaded on April 09,2021 at 01:39:55 UTC from IEEE Xplore. Restrictions apply.

*Proof:* Suppose that a scheme $\left(P, Q, \{A^{(i)}\}, \{B^{(i)}\}\right)$ is robust to any $s_2$ straggling APs in the system. Then, each data partition $D_j \in D = \{D_1, D_2, \ldots, D_k\}$ should be allocated to at least $s_2+1$ cells. Letting $\mu_i$ be the number of data partitions allocated to cell $i$, we should have

$$\sum_{i=1}^{L} \mu_i \geq k(s_2+1). \tag{12}$$

Now focus on a specific cell $i$. Given $s_1$ straggling nodes in the entire system, there can be maximally $s_1$ straggling nodes in a single cell. Hence, any data partition in a specific cell should be replicated to at least $s_1+1$ computing nodes. Let $r_{i,q}$ be the number of data partitions allocated to node $q$ in cell $i$. Since there are $f_i$ computing nodes in cell $i$, the total number of allocations for all nodes in cell $i$ is $\sum_{q=1}^{f_i} r_{i,q}$. Hence, we can write

$$\sum_{q=1}^{f_i} r_{i,q} \geq \mu_i(s_1+1) \tag{13}$$

for all $i \in [L]$. By summing up for all $i \in [L]$, the following holds:

$$\sum_{i=1}^{L}\sum_{q=1}^{f_i} r_{i,q} \geq \sum_{i=1}^{L} \mu_i(s_1+1) \geq k(s_1+1)(s_2+1). \tag{14}$$

Now we assign the same amount of data $r$ to all nodes, i.e., $r_{i,q} = r$ for all $q \in [f_i]$, $i \in [L]$, which leads to

$$r\sum_{i=1}^{L} f_i \geq k(s_1+1)(s_2+1). \tag{15}$$

By dividing both sides by $\sum_{i=1}^{L} f_i$, we have $r \geq \frac{k(s_1+1)(s_2+1)}{\sum_{i=1}^{L} f_i}$ which completes the proof. ■

This bound is fundamental in that no code exists that computes less than $\frac{(s_1+1)(s_2+1)}{\sum_{i=1}^{L} f_i}$ fraction of the entire data in successfully combating $s_1$ straggling nodes and $s_2$ straggling APs. An important observation is that the bound decreases as $\sum_{i=1}^{L} f_i$ increases, i.e., as the number of nodes in the overlapping cell region increases. This bound extends and generalizes Theorem 1 of [18] for our hierarchical setup with nodes in the overlapping cell areas. Next, making use of the cyclic and fractional repetition codes of [18] as component codes, we provide two hierarchical coding schemes which precisely utilize the computing nodes in the overlapping areas to speed up training while achieving this fundamental bound.

## A. Cyclic Outer Code With Fractional Repetition Inner Codes

*Motivating Example:* We first revisit the example in Fig. 2. It can be seen that this example achieves the lower bound on the computational load in (11), while combating any $s_1 = 1$ straggling node and $s_2 = 1$ straggling AP. Since the nodes in the overlapping areas compute the gradients for two APs and broadcast the result to both of them, the overall training time can be reduced.

*Outer Code:* We first provide construction on the encoding matrix of the outer code $Q$, which allocates $k$ data partitions over $L$ cells to tolerate $s_2$ straggling APs. The overall procedure is divided into the following three steps.

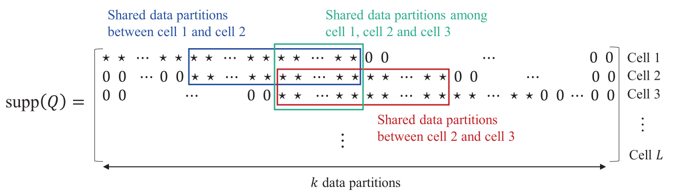

Shared data partitions between cell 1 and cell 2
Shared data partitions among cell 1, cell 2 and cell 3
Shared data partitions between cell 2 and cell 3
supp(Q) =
Cell 1
Cell 2
Cell 3
Cell L
k data partitions

Fig. 3. Support of $Q$, where $\star$ represents non-zero values. Each row has $\frac{k(s_2+1)}{L}$ non-zero values. The data partitions that are overlapped by two rows can be allocated to the nodes in the overlapped cell region.

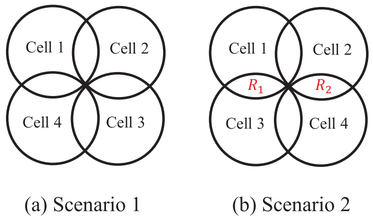


Fig. 4. Examples of cell indexing with $L = 4$, $s_2 = 1$. Here, we have $N(2) = 0$. Since cell 1 does not share data partitions with cell 3, and cell 2 does not share data with cell 4, we cannot allocate data partitions to the nodes in regions $R_1$ and $R_2$ in scenario 2.

*1) Support Construction:* The support construction of $Q$ is based on the cyclic code [18], as illustrated in Fig. 3. Here, we assume $f_i = f$ for all $i \in [L]$ (i.e., each AP selects the same number of nodes for participation) for code construction. Each row of $Q$ has $\frac{k(s_2+1)}{L}$ non-zero elements. For the first row, non-zero values are assigned to the first $\frac{k(s_2+1)}{L}$ elements. For the next row, the non-zero values are shifted $\frac{k}{L}$ steps to the right, and so on. As can be seen in Fig. 3, this code is suitable for the outer code since it allows each cell to share data partitions with not only the adjacent cells but also the cells with larger index differences.

*2) Cell Indexing:* Now let $N(i)$ be the number of data partitions shared by the cells with index difference $i$, i.e., the number of data partitions shared by cell $p$ and cell $q$, where $|p-q| = i$. Then for $L \leq 2s_2+2$, a visual inspection of matrix $Q$ leads to

$$N(i) = \begin{cases} k(s_2+i-L+1)/L, & i \geq s_2+1, \\ k(2s_2-L+2)/L, & L-s_2-1 \leq i < s_2+1, \\ k(s_2-i+1)/L, & i < L-s_2-1. \end{cases} \tag{16}$$

Note that $N(i) = N(L-i)$ due to the cyclic structure of matrix $Q$. It can be seen from (16) that unless $L = 2s_2+2$, we have $N(i) > 0$ for all $i \in [L]$, i.e., each cell shares data partitions with all other cells. For $L > 2s_2+2$, we have

$$N(i) = \begin{cases} k(s_2+i-L+1)/L, & i \geq L-s_2-1, \\ 0, & s_2+1 \leq i < L-s_2-1, \\ k(s_2-i+1)/L, & i < s_2+1. \end{cases} \tag{17}$$

For a given cell geometry specified as $L$, $N(i)$, and a given straggling parameter $s_2$, the goal of this step is to find an appropriate cell indexing solution. More specifically, we would

Authorized licensed use limited to: Korea Advanced Inst of Science & Tech - KAIST. Downloaded on April 09,2021 at 01:39:55 UTC from IEEE Xplore. Restrictions apply.

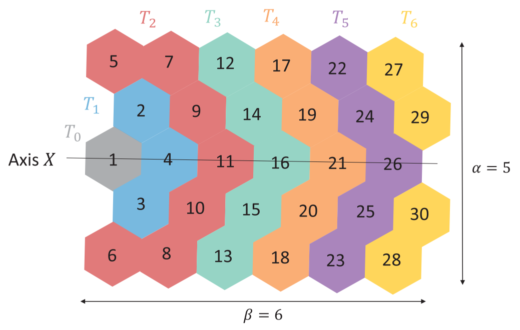


Fig. 5. Illustration of Algorithm 1.

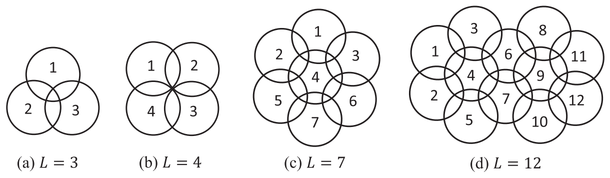


Fig. 6. Cell indexing for various cell geometry.

like to determine which row of $Q$ should be assigned to each AP for a given cellular network. Fig. 4 shows an example with $L = 4$, $s_2 = 1$. Each cell shares nodes with the adjacent cells. From (17), we have $N(2) = 0$. Hence, for a given cell geometry in Fig. 4, it can be seen that cell indexing of Fig. 4a is better than that of in Fig. 4b, since cell 1 does not share data partitions with cell 3, and cell 2 does not share data partitions with cell 4. More specifically, we cannot allocate data partitions to the nodes in regions $R_1$ and $R_2$ in Fig. 4b.

Now the question is how to find a valid cell indexing for a given cellular network? Here, we say that a cell indexing is valid if it allows every cell to share data partitions with its geometrically adjacent cells. We provide Algorithm 1 which finds a cell indexing solution for arbitrary 2-dimensional cell geometry. Fig. 5 illustrates Algorithm 1. First, select an initial cell (with cell index 1) and choose an axis $X$ which contains the initial cell. Let $T_0$ be a set that contains only the initial cell. Now define a set $T_b$ which contains all cells that are adjacent to the cells in $T_{b-1}$, i.e., each cell in $T_b$ shares some nodes with at least one cell in $T_{b-1}$. We first index the cells along the axis $X$, and then iteratively index the cells located left and right side of the axis $X$. In each set $T_b$, the number of iterations for cell indexing is maximally $|T_b|$ where the total number of iterations becomes maximally $L$. Hence, the complexity of the algorithm becomes $O(L)$. This ensures the practical applicability of the algorithm which scales linearly with the number of APs.

In Fig. 6, we provide valid cell indexing solutions depending on the cell geometry. For the cases in Figs. 6a, 6b, regardless of $s_2$, we can always find a valid cell indexing solution. For Figs. 6c and 6d, we can find a valid cell indexing if and only if $s_2 \geq 3$, which can be seen from Fig. 6 and $N(i)$ obtained in (16) and (17). In the following, we provide a condition on finding a valid cell indexing solution by Algorithm 1.

**Algorithm 1** Cell Indexing for Arbitrary 2-Dimensional Structure

```
Input: $\mathcal{C} = \{C_1, C_2, \cdots, C_L\}$, set of $L$ hexagonal cells, arbitrarily indexed at the beginning
Output: $\pi = [\pi_1, \pi_2, \cdots, \pi_L]$, indices of $n$ cells. Here, $\pi_k$ is the index of cell $C_k$. We have $\{\pi_1, \pi_2, \cdots, \pi_L\} = \{1, 2, \cdots, L\}$.
  Initialize: Set $\pi_k = 0$ for $k \in [L]$, and define $b = 0$.
  Step 1. Select an initial cell $s \in \mathcal{C}$ such that the number of adjacent cells of $s$ is less than six.
  Select an axis $X \subseteq \mathcal{C}$ satisfying two conditions: 1) $s \in X$ and 2) there exists a line $l$ containing the centers of all cells $x \in X$.
  Define $T_0 = \{s\}$. Set $\pi_s = 1$.
  Step 2. Specify $\pi_k$ values for $k \neq s$:
  while $\pi_k = 0$ for some $k \in [L]$ do
    Set $b = b + 1$ and define $t = \sum_{i=0}^{b-1} |T_i|$.
    Define $T_b = \{c \in \mathcal{C}$ : cell $c$ shares an overlapping area with another cell $c' \in T_{b-1}\}$
    Let $s'$ be the unique cell satisfying $s' \in X \cap T_b$. Set $\pi_{s'} = \sum_{i=0}^{b} |T_i|$.
    Define $\mathcal{C}_{\text{left}} = \{C_{i_1}, C_{i_2}, \cdots, C_{i_p}\} \subseteq T_b$, the set of cells in $T_b$ which are in the left side of axis $X$. Order the elements within the set such that $d(C_{i_l}, s') \geq d(C_{i_m}, s')$ for all $l \leq m$. Here, $d(C_i, C_j)$ is the distance between the centers of cells $C_i$ and $C_j$.
    Similarly, define $\mathcal{C}_{\text{right}} = \{C_{j_1}, C_{j_2}, \cdots, C_{j_{\{|T_b|-1-p\}}}\} \subseteq T_b$, the set of cells in $T_b$ which are in the right side of axis $X$.
    while $\exists\ C_k \in T_u$ such that $\pi_k = 0$ (i.e., an un-indexed cell exists in $T_b$) do
      Select $C_{j_l} \in \mathcal{C}_{\text{right}}$ satisfying $\pi_{j_l} = 0$ (choose the smallest $l$). If such $C_{j_l}$ exists, set $t = t + 1$ and $\pi_{j_l} = t$.
      Select $C_{i_l} \in \mathcal{C}_{\text{left}}$ satisfying $\pi_{i_l} = 0$ (choose the smallest $l$). If such $C_{i_l}$ exists, set $t = t + 1$ and $\pi_{i_l} = t$.
    end while
  end while
```

This guarantees the convergence of Algorithm 1, where the proof can be found in Appendix B.

*Proposition 1: Consider an $\alpha \times \beta$ cell geometry as in Fig. 5, where the total number of cells is $L = \alpha\beta$. Assume that $\alpha$ is an odd number. Then if $\frac{s_2}{L} \geq \frac{3}{2\beta}$, we can always obtain a valid cell indexing solution by Algorithm 1.*

Proposition 1 implies that as $\beta$ becomes larger, we can always construct valid cell indexing with a relaxed constraint on the ratio of straggling APs to $L$, i.e., $\frac{s_2}{L}$.

*3) Determining the Matrix Elements:* Now given an appropriate cell indexing with the support of $Q$ in Fig. 3, how do we determine the actual values of non-zero elements in matrix $Q$? As in Algorithm 2 below, which generalizes Algorithm 2 in [18], a random design of $Q$ allows PS to exactly recover the sum of gradients against $s_2$ straggling APs. The key idea is to design a cyclic matrix $Q$ where any $L - s_2$ rows are linearly independent and their span contains $\mathbf{1}_{L\times1}$.

Authorized licensed use limited to: Korea Advanced Inst of Science & Tech - KAIST. Downloaded on April 09,2021 at 01:39:55 UTC from IEEE Xplore. Restrictions apply.

The complexity of Algorithm 2 is approximately $O(L)$, which comes from $L$ iterations in constructing each row of $L \times L$ matrix $Q'$.

**Algorithm 2** Algorithm to Construct Outer Encoding Matrix $Q$

```
Stage 1: Generate random matrix $H \in \mathbb{R}^{s_2 \times L}$ to satisfy:
    1. Any $s_2$ columns of $H$ are linearly independent and dim(null $H$) $= L - s_2$
    2. $\mathbf{1}_{L\times 1} \in$ null $H$
Stage 2: Based on $H$, construct matrix $Q' \in \mathbb{R}^{L\times L}$ to satisfy:
    1. Any $L - s_2$ rows of $Q'$ are linearly independent and their span contains $\mathbf{1}_{L\times 1}$
    2. The support of $Q'$ has a cyclic structure
Stage 3: By replicating each element of $Q'$ by $\frac{k}{L}$ times row-wise, construct $Q$ where the support of $Q$ has the structure as in Fig. 3.
```

*Inner codes:* Based on the outer code designed above, each AP has access to $\frac{k(s_2+1)}{L}$ data partitions. Now we design inner codes, i.e., $\{B^{(i)}\}$ and $\{A^{(i)}\}$. Our main contribution is to allow a precise utilization of the shared data partitions between cells for designing $\{B^{(i)}\}$, to reduce computation and communication time at the nodes in the overlapping areas.

*a) Construction for cell $i$:* We first focus on the specific $i$-th AP connected with $f_i = f$ computing nodes (i.e., each AP selects the same number of nodes for participation). Our goal is to combat $s_1$ straggling computing nodes in each cell. We utilize the fractional repetition codes of [18] to construct $B^{(i)}$, which works as follows. Suppose $s_1 + 1$ divides $f$. We first divide the nodes into $\frac{f}{s_1+1}$ groups with equal size, having $s_1+1$ nodes in each group. The data partitions assigned to cell $i$ are also divided into $\frac{f}{s_1+1}$ *blocks* (a set of $D_i$s) with equal size, having $\frac{k(s_1+1)(s_2+1)}{fL}$ data partitions in each block. Then, we arrange the nodes in the same group to compute the gradients of the same data block. This guarantees cell $i$ to tolerate any $s_1$ straggling nodes among $f$ nodes.

*b) Condition for broadcasting:* Now there are two important problems to address: 1) How to jointly design $\{B^{(i)}\}$ to combat $s_1$ straggling nodes for every cell, while achieving the fundamental bound on computational load (by receiving the same data partitions from both APs) and having one communication round (by broadcasting) at the nodes in the overlapping areas? 2) Among the candidates of $\{B^{(i)}\}$ satisfying these conditions, which one is the best solution? Which data should be allocated to the nodes in the overlapping cell areas to achieve this best solution? We begin with the first problem, by observing the following lemma which provides a guideline for having minimum communication and computation load using $\{B^{(i)}\}$.

*Lemma 1 (Broadcasting): Suppose the encoding matrix of outer code $Q$ is constructed by Algorithm 2 and encoding matrices of inner codes $\{B^{(i)}\}$ are based on fractional repetition codes. Assume that $L = 3$ and $u_i = u$ for all $i \in [L]$ and $v_{i,j} = v$ for all $i, j \in [L]$. Then the necessary and sufficient condition for achieving the lower bound on computational load in (11) and for having one communication round for all nodes is $u \geq 2\left(\lceil \frac{v}{s_1+1} \rceil (s_1 + 1) - v\right)$.*

*Proof:* See Appendix C. ■

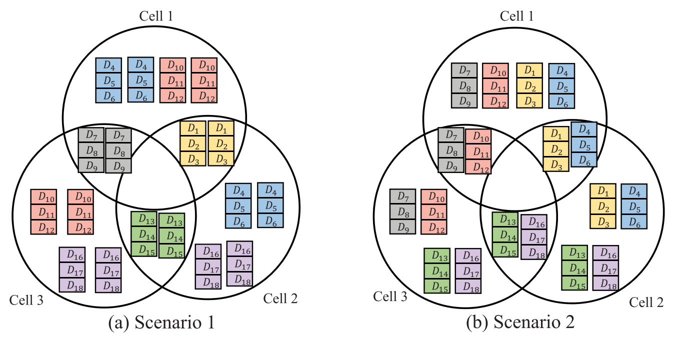


Fig. 7. Examples of different data allocation methods. The parameters are $L = 3$, $u_i = u = 4$, $v_{i,j} = v = 2$, $s_1 = 1$, $s_2 = 1$. The recovery threshold of scenario 1 and scenario 2 are 17 and 16, respectively.

If the condition in Lemma 1 holds, the nodes in the overlapping cell areas can broadcast the result to both APs, which further reduces the finishing time. Depending on straggler scenarios, we note that one can appropriately select $u$, $v$ by connecting the nodes in the overlapping cell areas to either one or two APs[2] to satisfy the condition in Lemma 1.

*3) Data allocation to the overlapping areas:* Now we provide an answer to the next question: Among the solutions that satisfy the condition in Lemma 1, which one is the best solution? How should we allocate data partitions to the nodes in the overlapped regions in order to achieve this solution? We provide an allocation method to minimize the recovery threshold [13], which is defined as the minimum number of successful nodes required at PS to always guarantee the exact sum of gradients. We provide an example in Fig. 7. Here, the overall computation is finished when PS receives the results of two out of three cells. If each AP can detect where each result is sent from, the recovery thresholds for Fig. 7a and Fig. 7b become $K = 17$ and $K = 16$, respectively. Motivated by this example, we provide Algorithm 3 to construct $\{B^{(i)}\}$ based on fractional repetition codes which allocate data blocks as uniformly as possible within the overlapping area, and also across the overlapping areas. More specifically, we allocate data blocks to minimize the maximum element $t_{max}$ of a set $T$, where $T$ contains the number of each data block located in the overlapping areas. For example, we have $T = \{2, 2, 2\}$ and $T = \{1, 1, 1, 1, 1, 1\}$ for Fig. 7a and Fig. 7b, respectively. To summarize, the goal of Algorithm 3 is to precisely allocate data blocks to achieve $T^* = \text{argmin}_T \; t_{max}$. The complexity of the algorithm (number of iterations for allocating data) becomes $O(n)$, scaling linearly with the number of participating devices. We first allocate data partitions to the nodes in the overlapping areas (which requires $3v$ iterations for allocating data), and arbitrarily assign the remaining partitions to the

[2]This is especially reasonable when a large number of nodes are in the overlapping areas between cells. In such cases, we can flexibly control $u$, $v$ by connecting a specific node in the overlapping areas to only one AP.

Authorized licensed use limited to: Korea Advanced Inst of Science & Tech - KAIST. Downloaded on April 09,2021 at 01:39:55 UTC from IEEE Xplore. Restrictions apply.

nodes in the non-overlapped regions (which requires $n-3v$ iterations of data allocation). Now we have the following theorem, with the proof given in Appendix D.

**Algorithm 3** Algorithm That Allocates Data Partitions to the Overlapping Cell Areas to Minimize the Recovery Threshold

**Stage 1**: Based on the fractional repetition scheme, construct matrix $M$ containing all possible data blocks to be assigned to the nodes in the system.
**Stage 2**: Construct the sets $I_1$, $I_2$, $I_3$ by taking the row indices of $M$ that should be assigned to $B^{(1)}$, $B^{(2)}$, $B^{(3)}$, respectively.
**Stage 3**: Allocate the first and last $v$ rows of $B^{(1)}$, $B^{(2)}$, $B^{(3)}$, corresponding to the overlapping areas between cells. Allocate data blocks to achieve $T^* = \text{argmin}_T\, t_{max}$. Subtract the allocated row indices from $I_1$, $I_2$, $I_3$.
**Stage 4**: Based on $I_1$, $I_2$, $I_3$, arbitrarily fill the remaining parts of $B^{(1)}$, $B^{(2)}$, $B^{(3)}$.

*Theorem 2 (Optimality of Algorithm 3): Suppose Q is constructed by Algorithm 2 and $\{B^{(i)}\}$ are based on fractional repetition codes. Then among the candidates of $\{B^{(i)}\}$ which can tolerate any $s_1$ straggling nodes and any $s_2$ straggling APs, the construction of $\{B^{(i)}\}$ based on Algorithm 3 achieves the minimum recovery threshold.*

Based on $\{B^{(i)}\}$, the corresponding $\{A^{(i)}\}$ can be easily constructed by $A^{(i)}B^{(i)} = [q_i; q_i; \ldots; q_i]$.

## B. Cyclic Codes for Both Outer and Inner Codes

In this subsection, we provide an alternative hierarchical coding scheme achieving the lower bound in (11), which can be used when the condition in Lemma 1 does not hold. We assume $f_i = f$ for all $i \in [L]$. The construction for both $Q$ and $\{B^{(i)}\}$ are based on cyclic codes, where $Q$ can be constructed by Algorithm 2. For constructing $\{B^{(i)}\}$, we only need to fill $\frac{k(s_2+1)}{L}$ columns of $B^{(i)}$ where the elements of $q_i$ are non-zero. Define a matrix $C^{(i)}$, by taking these $\frac{k(s_2+1)}{L}$ columns. We fill each row of $C^{(i)}$ with $\frac{k(s_1+1)(s_2+1)}{fL}$ non-zero elements. For the first row, the non-zero elements are assigned to the first $\frac{k(s_1+1)(s_2+1)}{fL}$ elements. For the next row, the non-zero values are shifted $\frac{k(s_2+1)}{fL}$ steps to the right. An appropriate random generation similar to Algorithm 2 can be used for construction. Assuming $L = 3$, there are two sufficient conditions for achieving the fundamental bound in (11), $u+2v \mid s_2+1$ and $v \le 1 + \lfloor \frac{u+2v}{s_2+1} - (s_1+1) \rfloor$, which can be obtained by using similar techniques to the proof of Lemma 1. Note that the coding schemes in subsections IV-A and IV-B have different conditions for constructions. One can decide which coding scheme to use depending on the current system parameters $u$, $v$.

## C. Trade-Off in Code Design: Computational Load Versus Recovery Threshold

Thus far, we provided results on designing the encoding and decoding matrices as functions of $u$, $v$. In this subsection, we explore trade-offs in selecting different $u$ and $v$. Note that $u + v = n/L$ is a constant for $L = 3$. It can be seen from Theorem 1 that when $v$ increases, it is possible to combat the same number of stragglers with a lower computational load. However, the following lemma shows that the recovery threshold is a monotonic increasing function of $v$ when $s_1$, $s_2$ are given. The proof can be found in Appendix E.

*Lemma 2: Assuming $L = 3$ and considering both codes in subsections IV-A and IV-B in the manuscript, the recovery threshold $K$ of hierarchical broadcast coding becomes*

$$K = \begin{cases} n - s_1, & if\ v > s_1 \\ n + v - 2s_1 - 1, & otherwise. \end{cases} \tag{18}$$

*for $s_2 = 1$. If $s_2 = 2$, we have*

$$K = \begin{cases} n + 3v - 3s_1 - 2, & if\ \ v \le \frac{s_1}{2} \\ n - \frac{3}{2}s_1 - 1, & if\ \ v > \frac{s_1}{2}\ \ and\ \ s_1|2 \\ n - \frac{3}{2}s_1 - \frac{1}{2} & otherwise. \end{cases} \tag{19}$$

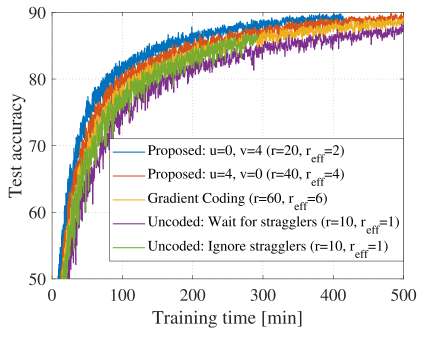
Equation (18) shows that for $v > s_1$, increasing $v$ is always beneficial since a lower computational load can be achieved with the same recovery threshold. For $v \le s_1$, there exists a trade-off. Again from (19), increasing $v$ is always beneficial for $v > \frac{s_1}{2}$, while there is a trade-off for $v \le \frac{s_1}{2}$.

## V. Experiments on Amazon EC2

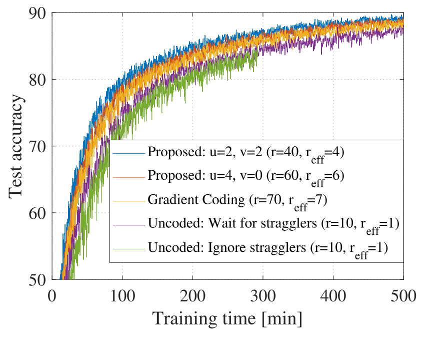
We provide extensive experimental results by implementing our methods on Amazon EC2. We assume $u_i = u$ for all $i \in [L]$ and $v_{i,j} = v$ for all $i, j \in [L]$. Note that $u = \frac{n}{L}$, $v = 0$ corresponds to the case where hierarchical coding is applied without considering the shared nodes between *APs*. We consider two *uncoded* schemes for comparison, where the data is divided uniformly across all the *workers* (i.e., computing devices) without replication. The first uncoded scheme is to wait until all stragglers to arrive. Each AP is connected to $\frac{n}{L}$ workers. To obtain the sum of local gradients, each AP waits for all results from the corresponding workers. The PS waits for all the results from the APs to recover the exact sum of total gradients. Another uncoded scheme is *ignoring stragglers* approach, where the PS ignores $s_2$ APs when updating the model in each iteration. Each AP ignores $s_1$ workers before sending the aggregated gradient to the PS. Finally, we consider *naive* coding scheme for comparison, which does not reflect the hierarchical/broadcast nature of the network; here PS directly codes across the workers by gradient coding [18]. Considering both $s_1$ and $s_2$, gradient coding is designed targeting $s_1 + \frac{n}{L}s_2$ straggling workers. In this case there is no general rule for the AP. Hence, each AP sends the result of each worker as soon as it is received, which requires $\frac{n}{L}$ times more communication burden between APs and the PS compared to other schemes. We define the effective redundancy as $r_{\text{eff}} = \frac{r}{r_{\text{uncoded}}}$, which shows the relative redundancy of each scheme compared to the method without coding (i.e., uncoded scheme).

We used PyTorch with MPI4py package for implementation. Stochastic gradient descent with momentum is used for training, where we adopted momentum term of $\eta = 0.9$. We utilized one PS, $L = 3$ APs and $n = 12$ workers. For all proposed schemes, coding scheme of Section IV-B is utilized

Authorized licensed use limited to: Korea Advanced Inst of Science & Tech - KAIST. Downloaded on April 09,2021 at 01:39:55 UTC from IEEE Xplore. Restrictions apply.



(a) Target design: $s_1 = 1$, $s_2 = 1$.



(b) Target design: $s_1 = 2$, $s_2 = 1$.

Fig. 8. Test accuracy versus training time for ResNet18 with CIFAR-10. We artificially delayed $s_1 = 1$ worker and $s_2 = 1$ AP.

for the case $u = 4$, $v = 0$, $s_1 = 2$ where the scheme of Section IV-A is utilized for other scenarios. Here, $s_1$ is a design parameter which depends on the signal-to-interference-plus-noise ratio (SINR) at the APs. If the SINR is below a certain threshold (i.e., outage case), packet retransmission is required at the devices which results in a server form of straggling. Hence, if the channel condition between the device and the AP is bad, we expect a large number of straggling devices and thus set $s_1$ to be large when designing our coding scheme. Similarly, if the frequency reuse factor is large (i.e., if frequency reuse between cells is frequent), we can expect a larger co-channel interference and thus set $s_1$ to be large. To sum up, both channel fading and co-channel interference are the factors that can be simply parameterized as $s_1$. We compare the performances with various $s_1$ values ($s_1 = 1, 2, 3$), which correspond to scenarios with different channel conditions and frequency reuse factors (co-channel interferences).

## A. Experiments for Deep Neural Networks

We trained ResNet18 using CIFAR-10 dataset, where the number of parameters of model is 11,173,962 and the number of training samples and test samples are 50,000 and 10,000, respectively. Mini-batch stochastic gradient descent is used with a batch size of 120. `c4.2xlarge` instances are used for PS, APs and workers.

Fig. 8 shows test accuracy versus training time for different schemes. Here, the test accuracy is averaged by performing 5 independent trainings. The coding schemes are designed targeting $s_1 = 1$, $s_2 = 1$ for Fig. 8a and $s_1 = 2$, $s_2 = 1$ for Fig. 8b, where we considered two proposed schemes depending on $u$, $v$ values for each figure. Note that even when some nodes are located in the overlapping cell areas, we can design a code for $u = 4$, $v = 0$ (Figs. 8a, 8b) by connecting all nodes only to one AP. Similarly, we can construct a code for $u = 2$, $v = 2$ (Fig. 8b) even when more than 2 nodes are in the overlapping cell areas by connecting specific nodes only to one AP. To see the effect of stragglers, we artificially delayed $s_1 = 1$ worker for 2 seconds $s_2 = 1$ AP for 2 seconds at each iteration. While the uncoded scheme (ignore straggles) should have the smallest average time per iteration by ignoring stragglers, it has lower test accuracy compared to the coded schemes in Fig. 8. This is because this uncoded scheme loses data by ignoring stragglers at each iteration. As $s_1$ increases, the performance of ignoring stragglers approach gets worse, since more data is lost at each iteration. For each plot, it is observed that the proposed schemes provide better test accuracy compared to the naive coding scheme. It can be also seen that the proposed scheme with $v = 0$ has lower test accuracy compared to $v > 0$, i.e., one or more nodes in the overlapping cell areas. Table I shows the training time of each scheme in Fig. 8a, to achieve the target test accuracy. As the target accuracy becomes larger, there is a larger gap between the proposed coding idea and other comparison schemes.

We provide a more detailed result in Fig. 9, which shows the average running time per iteration corresponding to the plots in Fig. 8. We randomly delayed $s_1 = 1$ worker and $s_2 = 1$ AP at each iteration. The performance of the uncoded scheme (wait for stragglers) becomes poorer as delay increases, since it has to wait for all the workers and APs. As expected, the second uncoded scheme (ignore stragglers) has the lowest average time since it has the minimum computational load and is not largely affected by the delays. The coded schemes are also not largely affected by the delays. Similar to the results in test accuracy plot, it can be seen that the coding scheme which utilizes the nodes in the overlapping cell areas have better performance compared to other coding schemes, confirming the advantage of the proposed design.

Now what happens if there are more stragglers than expected, giving rise to a mismatch situation? We address this question in Fig. 10, where the target stragglers are the same as in Figs. 8, 9. Here, we artificially delayed $s_1 = 3$ workers and $s_2 = 1$ AP for 2 seconds, which is beyond the target design. The test accuracy is obtained by averaging the results of 5 independent trainings. It can be seen that the scheme with larger $v$ tend to have more performance degradation in this mismatch scenario. This is because with a larger $v$, the system has a higher probability to have stragglers in the overlapping cell areas, which slows down both APs. The mismatch sensitivity varies depending on $u$, $v$ values but in general proposed coding still gives advantage compared to naive application of gradient coding and uncoded schemes.

A more detailed observation on mismatch scenarios can be found in Fig. 11, which shows the average running time per iteration for different schemes. The setup is the same as in Fig.10b, where the codes are designed targeting $s_1 = 2$ and $s_2 = 1$. Fig. 11a is for no actual worker delay (i.e., $s_1 = 0$,

Authorized licensed use limited to: Korea Advanced Inst of Science & Tech - KAIST. Downloaded on April 09,2021 at 01:39:55 UTC from IEEE Xplore. Restrictions apply.

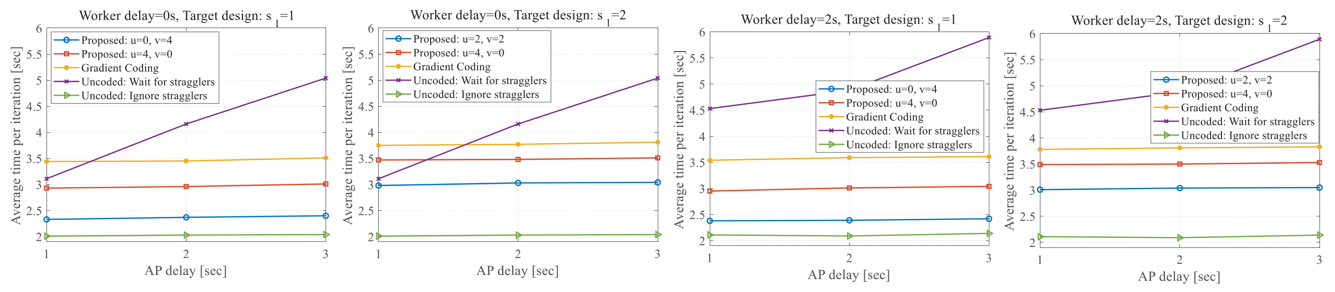
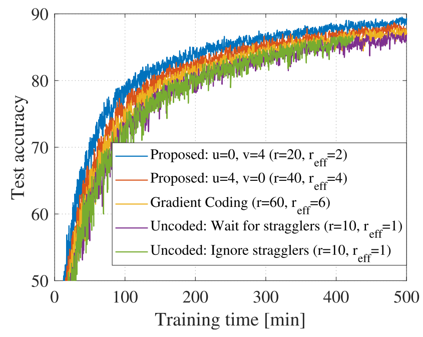
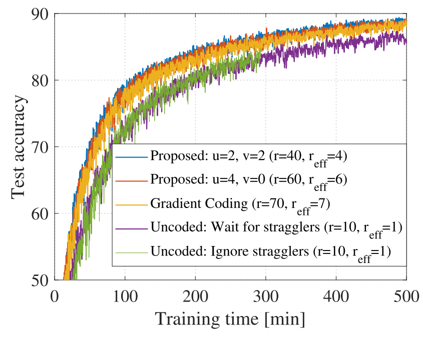


Fig. 9. Average time per iteration where the codes are designed targeting $s_2 = 1$ and different $s_1$ values with $n = 12$ workers. We artificially delayed $s_1 = 1$ worker and $s_2 = 1$ AP.



(a) Target design: $s_1 = 1$, $s_2 = 1$.



(b) Target design: $s_1 = 2$, $s_2 = 1$.

Fig. 10. Test accuracy versus training time for ResNet18 with CIFAR-10. We artificially delayed $s_1 = 3$ workers and $s_2 = 1$ AP.

an under-match), Fig. 11b is for actual 2 worker delays (i.e., $s_1 = 2$, the exact match) and Fig. 11c is for actual 3 worker delays (i.e., $s_1 = 3$, an over-match). If we artificially delay more workers than that we targeted, the schemes with larger $v$ tend to have more performance degradation since it is more likely to have stragglers in the overlapping cell areas. This effect can be seen from Fig. 11c with $u = v = 2$. If only the workers in the non-overlapping areas are delayed, the average time is not largely increased compared to the cases in Fig. 11a and Fig. 11b. However, if we only delay the workers in the overlapping areas, the average time increases substantially. When the number of stragglers is within the range of target design, the trend is consistent with the results in Fig. 8.

TABLE I

TRAINING TIME (MIN) ACHIEVING THE DESIRED ACCURACY FOR TRAINING RESNET18 IN FIG. 8a

| Target test accuracy | 80% | 85% | 88% |
|---|---|---|---|
| Proposed: $u = 0$, $v = 4$ | 72.3 | 138.4 | 242.3 |
| Proposed: $u = 4$, $v = 0$ | 89.2 | 170.8 | 311.4 |
| Gradient Coding [18] | 108.8 | 206.2 | 367.7 |
| Uncoded: Wait for stragglers | 146.4 | 268.8 | 497.2 |
| Uncoded: Ignore stragglers | 117.7 | 212.5 | – |

## B. Experiments for Logistic Regression Models

Now we provide experimental results for training a logistic regression model by using the Amazon Employee Access dataset from Kaggle.[3] After applying one-hot encoding as a preprocessing step, which converts the categorical features to binary features, the dimension of the model we used for experiment becomes $d = 263,500$. `t2.micro` instances are used for workers and subamsters, and `c4.2xlarge` for the PS.

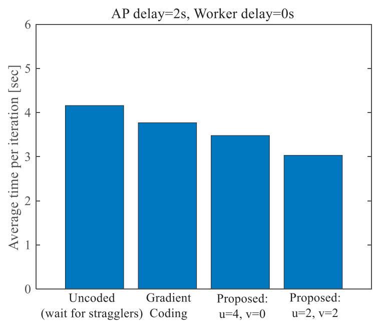
We provide Generalization AUC (area under the curve) versus training time in Fig. 12, where $26,220$ training samples are used with $n = 12$ workers. Codes are designed targeting $s_2 = 1$ and various $s_1$ values. We artificially delayed $s_2 = 1$ AP for 0.5 seconds. It can be again seen that the scheme utilizing the nodes in the overlapping cell areas gives the best performance. The results are consistent with the results on ResNet18, which shows the advantage of proposed schemes over naive gradient coding and uncoded schemes.

## VI. CONCLUSION

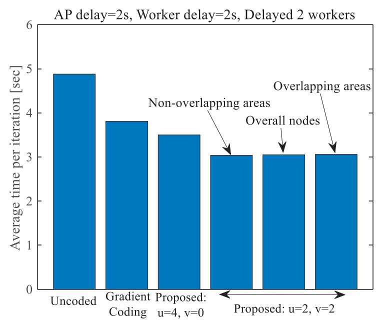
In this article, we provided a provable coding technique with three features for speeding up wireless distributed learning in edge computing scenarios: combats stragglers in both hierarchical layers, achieves the fundamental lower bound on computational load, and fully utilizes the broadcast nature of wireless networks. Experimental results on Amazon EC2 confirm the advantage of our scheme for both deep neural networks and logistic regression models. Our result offers a promising solution to expedite learning at the wireless edge, where minimizing the training time of learning models is of paramount importance.

[3]https://www.kaggle.com/c/amazon-employee-access-challenge

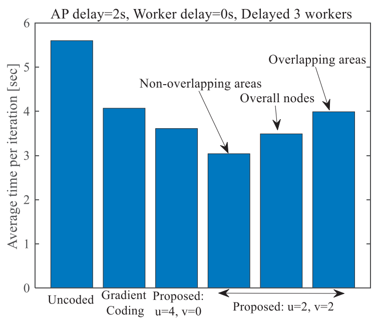
Authorized licensed use limited to: Korea Advanced Inst of Science & Tech - KAIST. Downloaded on April 09,2021 at 01:39:55 UTC from IEEE Xplore. Restrictions apply.



(a) No workers delayed

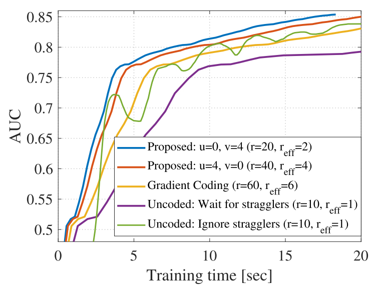


(b) Two random workers delayed

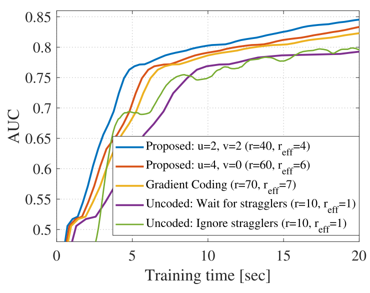


(c) Three random workers delayed

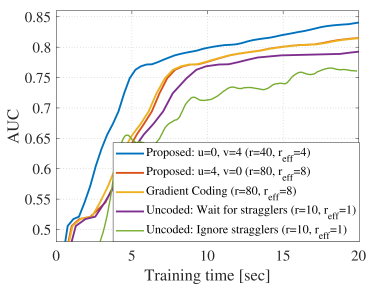
Fig. 11. Average time per iteration where the codes are designed targeting $s_1 = 2$, $s_2 = 1$ with $n = 12$ workers. We artificially delayed $s_2 = 1$ AP and $s_1 = 0, 2, 3$ workers. Especially for the case with $u = v = 2$, we consider three cases: the workers to be delayed are randomly selected from 1) overlapping areas only, 2) non-overlapping areas only, and 3) overall system.



(a) Target design: $s_1 = 1$, $s_2 = 1$.



(b) Target design: $s_1 = 2$, $s_2 = 1$.



(c) Target design: $s_1 = 3$, $s_2 = 1$.

Fig. 12. AUC versus training time with $n = 12$ workers. Each plot is designed targeting $s_2 = 1$ and different $s_1 = 1, 2, 3$. At each iteration, we artificially delayed $s_2 = 1$ AP for 0.5 seconds.

## Appendix

### A. Generalization to More Than Two Overlapping APs

Our idea can be generalized to connections to more than two APs following the similar procedures in Sections III and IV. Let $w_{i,j,l}$ be the number of nodes located in the overlapping region among cell $i$, cell $j$ and cell $l$. Then by simply rewriting $f_i$ of (3) $f_i = u_i + \sum_{j\in[L]\setminus\{i\}} v_{i,j} + \sum_{j\in[L]\setminus\{i\}} \sum_{l\in[L]\setminus\{i,j\}, j<l} w_{i,j,l}$, the fundamental bound in Theorem 1 remains exactly the same. Similarly, if we consider $N$-overlapping APs, we just need to rewrite $f_i$ and the fundamental bound in Theorem 1 remains the same. The coding idea can be also extended to achieve this bound. We first construct the encoding matrix $Q$ of the outer code by using Algorithm 2. When allocating data partitions to each computing node based on Algorithm 3, the key idea is to allocate data partitions that is shared by *multiple* APs to the nodes in the overlapping areas. As an example, Fig. 13 shows the case with $n = k = 32$, $L = 4$, $s_1 = 3$, $s_2 = 2$. Here, the nodes in the overlapping areas among 3 APs send their results to all corresponding 3 APs. It can be seen that the fundamental bound in Theorem 1 $r = k(s_1+1)(s_2+1)/\sum_{i=1}^{L} f_i = 32\cdot 4\cdot 3/32 = 8$ is achieved.

### B. Proof of Proposition 1

Take an axis parallel to the y-axis that contains cells located in the center of the row (see Fig. 5). It is easy to see then that if we apply Algorithm 1, the largest index difference between cells becomes $\max_b |T_b|$. Hence from $N(i)$ in (17), if $s_2 \geq \max_b |T_b|$, we can always find a valid cell indexing solution since it allows each node to share data partitions with cell index difference less than $\max_b |T_b|$. With a visual inspection of Fig. 5, we have

$$|T_b| = \begin{cases} 1 + 3b, & b \leq \dfrac{\alpha-1}{2},\ b \mid 2, \\ 3b, & b \leq \dfrac{\alpha-1}{2},\ b \nmid 2, \\ \alpha, & \dfrac{\alpha-1}{2} < b < \beta \end{cases} \tag{20}$$

which leads to $\max_b |T_b| = \frac{3\alpha-1}{2}$ if $\frac{\alpha-1}{2} \mid 2$, $\max_b |T_b| = \frac{3\alpha-3}{2}$ otherwise. Hence if $s_2 \geq \frac{3\alpha}{2}$, i.e., $\frac{s_2}{L} \geq \frac{3}{2\beta}$, we can always find a valid cell indexing solution by Algorithm 1.

### C. Proof of Lemma 1

For $L = 3$, there exist three straggling AP scenarios described as follows.

**Case 1** ($s_2 = 0$): Since this case does not consider any straggling APs, the overall dataset is uniformly divided into $L$ cells without repetition. No data partitions are shared by the cells, and there are no broadcasting nodes $(v = 0)$ and each AP is connected to $\frac{n}{L}$ nodes $(u = \frac{n}{L})$.

Authorized licensed use limited to: Korea Advanced Inst of Science & Tech - KAIST. Downloaded on April 09,2021 at 01:39:55 UTC from IEEE Xplore. Restrictions apply.

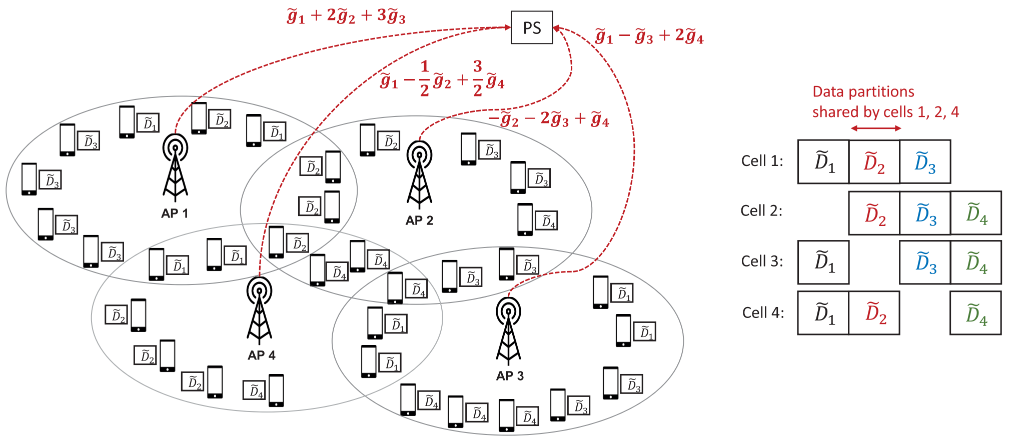

PS
$\tilde{g}_1 + 2\tilde{g}_2 + 3\tilde{g}_3$
$\tilde{g}_1 - \tilde{g}_3 + 2\tilde{g}_4$
$\tilde{g}_1 - \frac{1}{2}\tilde{g}_2 + \frac{3}{2}\tilde{g}_4$
$-\tilde{g}_2 - 2\tilde{g}_3 + \tilde{g}_4$
AP 1
AP 2
AP 3
AP 4
Data partitions shared by cells 1, 2, 4
Cell 1: $\tilde{D}_1$ $\tilde{D}_2$ $\tilde{D}_3$
Cell 2: $\tilde{D}_2$ $\tilde{D}_3$ $\tilde{D}_4$
Cell 3: $\tilde{D}_1$ $\tilde{D}_3$ $\tilde{D}_4$
Cell 4: $\tilde{D}_1$ $\tilde{D}_2$ $\tilde{D}_4$

Fig. 13. Generalization to more than two overlapping APs with $n = k = 32$, $L = 4$, $s_1 = 3$, $s_2 = 2$. Here, we have $\tilde{D}_1 := \{D_1, D_2, \dots D_8\}$, $\tilde{D}_2 := \{D_9, D_{10}, \dots D_{16}\}$, $\tilde{D}_3 := \{D_{17}, D_{18}, \dots D_{24}\}$, $\tilde{D}_4 := \{D_{25}, D_{26}, \dots D_{32}\}$ and $\tilde{g}_1 := g_1 + g_2 + \dots + g_8$, $\tilde{g}_2 := g_9 + g_{10} + \dots + g_{16}$, $\tilde{g}_3 := g_{17} + g_{18} + \dots + g_{24}$, $\tilde{g}_4 := g_{25} + g_{26} + \dots + g_{32}$.

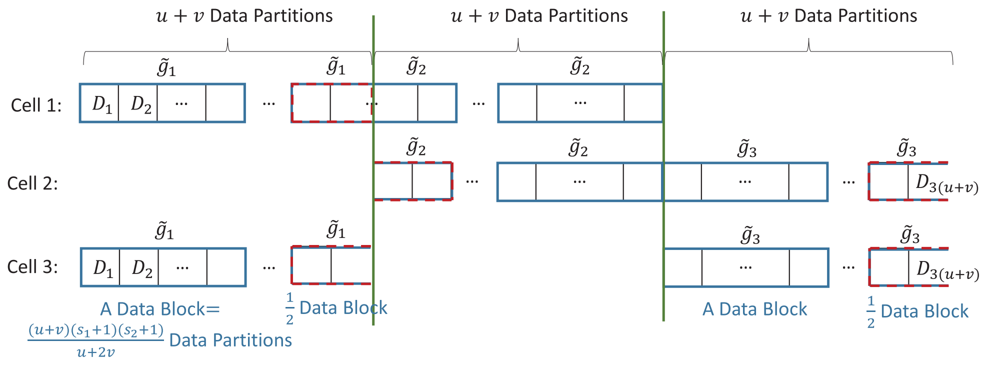


Fig. 14. The case for $s_2 = 1$. Each cell shares $u + v$ data partitions with both adjacent cells. Within each cell, there are $\frac{u+2v}{s_1+1}$ types of data blocks which are replicated $s_1 + 1$ times.

**Case 2** ($s_2 = 1$): Define $\tilde{g}_i := \sum_{j=(u+v)(i-1)+1}^{(u+v)i} g_i$ for all $i \in \{1, 2, 3\}$. Then, based on $Q$ constructed by Algorithm 1, we can always write

$$q_1 G = \alpha_{1,1} \sum_{i=1}^{u+v} g_i + \alpha_{1,2} \sum_{i=u+v+1}^{2(u+v)} g_i = \alpha_{1,1}\tilde{g}_1 + \alpha_{2,1}\tilde{g}_2 \quad (21)$$

$$q_2 G = \alpha_{2,1} \sum_{i=u+v+1}^{2(u+v)} g_i + \alpha_{2,2} \sum_{i=2(u+v)+1}^{3(u+v)} g_i = \alpha_{2,1}\tilde{g}_2 + \alpha_{2,2}\tilde{g}_3 \quad (22)$$

$$q_3 G = \alpha_{3,1} \sum_{i=2(u+v)+1}^{3(u+v)} g_i + \alpha_{3,2} \sum_{i=1}^{u+v} g_i = \alpha_{3,1}\tilde{g}_3 + \alpha_{3,2}\tilde{g}_1 \quad (23)$$

where $\alpha_{i,j}$ is the $[(u + v)(j - 1) + 1]$-th element of vector $q_i$ for all $i \in \{1, 2, 3\}$, $j \in \{1, 2\}$. This indicates that each cell shares one half of its allocated data partitions with one adjacent cell and the other half with another adjacent cell (see Fig. 14). Now we prove both directions:

(i) Suppose there exists a scheme achieving the lower bound on redundancy with only one round of gradient computation (with computational load $\frac{(u+v)(s_1+1)(s_2+1)}{u+2v}$) and one round of communication (by broadcasting) for the nodes in the overlapping area. Since all the non-zero elements of $\{B^{(i)}\}$ are one, only one round of communication is required if one round of computation is possible. Hence, it is enough to focus on one round of computation for the proof.

By applying fractional repetition codes as the inner code, there exist $\frac{u+2v}{s_1+1}$ different types of data blocks in each cell, where each data block contains $\frac{(u+v)(s_1+1)(s_2+1)}{u+2v}$ different data partitions (see Fig. 14). By a fractional repetition code, each data block is replicated $s_1 + 1$ times in each cell. Hence, $\lceil \frac{v}{s_1+1} \rceil$ is the minimum number of data block types that should be allocated in each overlapping area. This leads to:

$$\underbrace{\frac{(u+v)(s_1+1)(s_2+1)}{u+2v} \left\lceil \frac{v}{s_1+1} \right\rceil}_{\text{Minimum number of data partition types in each overlapping area}} \quad (24)$$

$$\leq \underbrace{(u+v)s_2}_{\text{Number of types of data partitions shared by each adjacent cell}} \quad (25)$$

$$\frac{(u+v)(s_1+1)(s_2+1)}{u+2v} \left\lceil \frac{v}{s_1+1} \right\rceil \quad (26)$$

$$\leq \underbrace{\frac{1}{2}(u+v)(s_2+1)}_{\text{one half of data partition types in a cell}}. \quad (27)$$

Authorized licensed use limited to: Korea Advanced Inst of Science & Tech - KAIST. Downloaded on April 09,2021 at 01:39:55 UTC from IEEE Xplore. Restrictions apply.

Equation $(24)$ comes from the fact that the minimum number of types of data partitions in each overlapping area should not exceed the number of types of data partitions that are shared by two APs. It should also not exceed one half of the data partition types in a cell, which is given by the right hand side of $(26)$. If not, it is not possible to assign data blocks in another overlapping area which also requires more than one half of data partitions in the cell. For $s_2 \geq 1$, combining these two leads to $u \geq 2\left(\lceil \frac{v}{s_1+1} \rceil (s_1+1) - v\right)$.

(ii) Suppose $u \geq 2\left(\lceil \frac{v}{s_1+1} \rceil (s_1+1) - v\right)$. We prove that we can always construct a fractional repetition scheme achieving the lower bound on redundancy with only one computation round in the overlapping area. We first divide the overall data partitions into blocks and allocate them to each cell as in Fig. 14. Note that $\frac{(u+v)(s_1+1)(s_2+1)}{u+2v}$ divides $(u+v)(s_2+1)$, with quotient $\frac{u+2v}{s_1+1}$. Hence, we can write

$$(u+v)(s_2+1) = \frac{(u+v)(s_1+1)(s_2+1)}{u+2v}\alpha \tag{28}$$

where $\alpha$ is an integer. Now we observe how many types of data blocks can be shared by the cells. If $\frac{(u+v)(s_1+1)(s_2+1)}{u+2v}$ divides $(u+v)$, the number of types becomes $\frac{u+2v}{(s_1+1)(s_2+1)}$. If not, we can write

$$u+v = \frac{(u+v)(s_1+1)(s_2+1)}{u+2v}\alpha' + \gamma, \tag{29}$$

where $\alpha'$ is the quotient and $\gamma$ is the remainder. By combining (28) and (29) we have

$$\alpha - (s_2+1)\alpha' = \frac{(s_2+1)\gamma}{\frac{(u+v)(s_1+1)(s_2+1)}{u+2v}}. \tag{30}$$

Since the left hand side is an integer, the right hand side should also be an integer. Note that $(s_2+1)\gamma < (s_2+1)\frac{(u+v)(s_1+1)(s_2+1)}{u+2v}$ since $\gamma < \frac{(u+v)(s_1+1)(s_2+1)}{u+2v}$. This leads to $\frac{(s_2+1)\gamma}{\frac{(u+v)(s_1+1)(s_2+1)}{u+2v}} < s_2+1$. If $s_2 = 1$, we have $\frac{2\gamma}{\frac{(u+v)(s_1+1)(s_2+1)}{u+2v}} < 2$, which indicates $\gamma = \frac{1}{2}\frac{(u+v)(s_1+1)(s_2+1)}{u+2v}$. Therefore, if $\frac{(u+v)(s_1+1)(s_2+1)}{u+2v}$ does not divide $(u+v)(s_2+1)$, the remainder's size is always one half of the block size as can be seen in Fig. 14. This always guarantees the construction of $\frac{u+2v}{s_1+1}$ data blocks in each cell as in Fig. 14. Since $u \geq 2\left(\lceil \frac{v}{s_1+1} \rceil (s_1+1) - v\right)$, which satisfies both (24) and (26), we can always allocate $\lceil \frac{v}{s_1+1} \rceil$ types of data blocks in the overlapping area. After allocating the blocks in the overlapping area, the remaining blocks can be allocated to $u$ nodes in the cell. It can be seen that this construction always guarantees redundancy of $\frac{(u+v)(s_1+1)(s_2+1)}{u+2v}$ for all nodes. Moreover, since the nodes in the overlapping areas are given the same encoded vectors from both APs, the proposed code guarantees a single computation round as well as a single communication round at all nodes.

**Case 3** ($s_2 = 2$): Since all the gradients should be recovered with only one AP, the overall data partitions are replicated and assigned to each AP. The overall data partitions $(u+v)L$ is grouped into $\frac{u+2v}{s_1+1}$ different types of data blocks, where each data block contains $\frac{(u+v)(s_1+1)L}{u+2v}$ data partitions. Since $s_2 + 1 = 3 = L$, $\frac{(u+v)(s_1+1)(s_2+1)}{u+2v}$ data partitions are in each block.

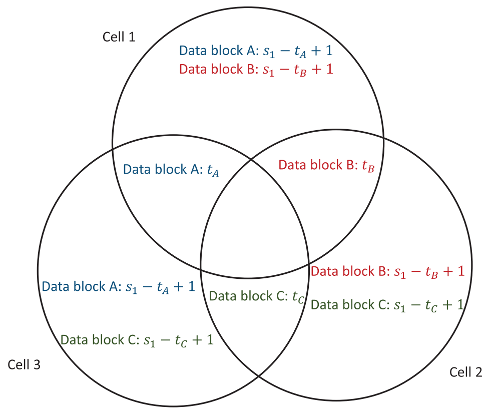

Cell 1
Data block A: $s_1 - t_A + 1$
Data block B: $s_1 - t_B + 1$
Data block A: $t_A$
Data block B: $t_B$
Data block B: $s_1 - t_B + 1$
Data block A: $s_1 - t_A + 1$
Data block C: $t_C$
Data block C: $s_1 - t_C + 1$
Data block C: $s_1 - t_C + 1$
Cell 3
Cell 2

Fig. 15. Basic idea for proof of Theorem 2 with $s_2 = 1$. When data block $A$ is allocated $t_A$ times in the overlapping area between cell 1 and cell 3, it is allocated $s_1 - t_1 + 1$ times in each non-overlapping area in cell 1 and cell 3. Recovery threshold is minimized when we properly design $t_A, t_B, t_C \ldots$ to minimize the maximum of $\{t_A, t_B, t_C \ldots\}$, i.e., allocate each data block as uniformly as possible.

Moreover, since all cells share all the blocks, we can always construct the codes achieving both lower bound on redundancy and one round of computation and communication by filling each overlapping area one by one. Hence, no conditions on $u$,$v$ are required.

## D. Proof of Theorem 2

**Case 1** ($s_2 = 0$): Since no data partitions are shared by the cells, there are no broadcasting nodes $(v = 0)$ and each AP is connected to $\frac{n}{L}$ nodes $(u = \frac{n}{L})$.

**Case 2** ($s_2 = 1$): Our basic idea is illustrated in Fig. 15. Since $\{B^{(i)}\}$ are based on fractional repetition codes, if data block $A$ in Fig. 15 is allocated $t_A$ times in the overlapping area between cell 1 and cell 3, it should be also allocated $s_1 - t_A + 1$ times in each non-overlapping area of both cells. Note that data block $A$ is not allocated in cell 2 since $s_2 = 1$ (see Fig. 14). We use the definition of $T$ in the main paper, containing the number of each data block in the the overlapping areas, as $T = \{t_A, t_B, t_C, \ldots\}$. Note that if $t \in T$, $t > 0$.

Consider an arbitrary data block that is located in the overlapping area $t$ times, $t \in T$. Consider the case where all nodes except the nodes having that specific block are finished. We can see that PS cannot have the exact $\sum_{i=1}^{k} g_i$, since the computation of only one cell is completed. For example, if we select data block $A$ not to be finished, we have $t = t_A$ and computation at cell 1 and cell 3 are not completed. After one more node finishes its work, PS can obtain the exact sum of gradients. The number of nodes that is finished can be written as $K_t = n - [2(s_1 - t + 1) + t] + 1 = n - 2s_1 + t - 1$ which is an increasing function of $t$. When the set $T$ is given, it can be easily seen that recovery threshold $K$ becomes

$$K = \max_{t \in T} K_t. \tag{31}$$

Now the minimum recovery threshold $K^*$ can be written as follows:

$$K^* = \min_{T} \max_{t \in T} n - 2s_1 + t - 1 \tag{32}$$

Authorized licensed use limited to: Korea Advanced Inst of Science & Tech - KAIST. Downloaded on April 09,2021 at 01:39:55 UTC from IEEE Xplore. Restrictions apply.

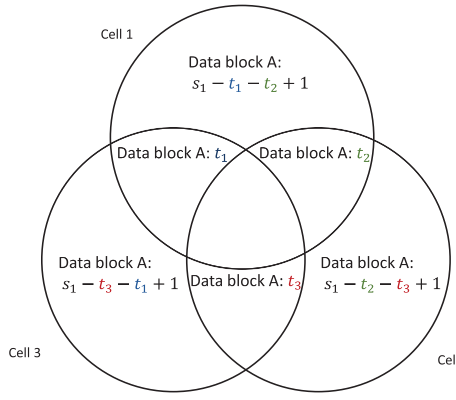

Cell 1
Data block A: $s_1 - t_1 - t_2 + 1$
Data block A: $t_1$
Data block A: $t_2$
Data block A: $s_1 - t_3 - t_1 + 1$
Data block A: $t_3$
Data block A: $s_1 - t_2 - t_3 + 1$
Cell 3
Cell 2

Fig. 16. Basic idea for proof of Theorem 2 with $s_2 = 2$.

Since $n - 2s_1 + t - 1$ is an increasing function of $t$, minimum recovery threshold is obtained when the maximum element of $T$ is minimized. More specifically, we would like to construct $T^*$ such that

$$T^* = \text{argmin}_T \ t_{max} \tag{33}$$

where $t_{max}$ is the maximum element of a given $T$. Hence, it is enough to allocate each data block as uniformly as possible within each overlapping area, which exactly stage 3 in Algorithm 2 does.

**Case 3** ($s_2 = 2$): For this case, since all the cells share the whole data partitions, a specific data block can be located in any area (see Fig. 16). Similar to the case $s_2 = 1$, we define a set $T$ having the number of each data block located in the overlapping area. From Fig. 16, we have $T = \{t_1+t_2+t_3, \ldots\}$. Then consider an arbitrary data block corresponding to $t \in T$, where $t = t_1 + t_2 + t_3$ is the number of that specific data block located in the overlapping areas (see Fig. 16). Consider the case where all nodes except for the nodes having that specific block are finished. The overall computation is still not finished, since none of the cells are finished. After one more node finishes, PS can guarantee the exact sum of gradients. Now the number of nodes that is finished becomes $K_t = n - [(s_1 - t_1 - t_2 + 1) + (s_1 - t_2 - t_3 + 1) + (s_1 - t_3 - t_1 + 1) + t_1 + t_2 + t_3] + 1 = n - 3s_1 + (t_1 + t_2 + t_3) - 2 = n - 3s_1 + t - 2$. When the set $T$ is given, it can be easily seen that recovery threshold $K$ becomes

$$K = \max_{t \in T} K_t. \tag{34}$$

The minimum recovery threshold $K^*$ becomes

$$K^* = \min_T \max_{t \in T} n - 3s_1 + t - 2, \tag{35}$$

which is again an increasing function of $t$. We can achieve $K^*$ by minimizing the maximum element of $T$. This means that we should allocate the data blocks as uniformly as possible across the overlapping areas. For example, if we allocated a specific data block $A$ in any of the overlapping area, the next data block that is allocated to the overlapping areas should not be data block $A$. This completes the proof.

## *E. Proof of Lemma 3*

Note that work for each cell is finished when $u + 2v - s_1$ out of $u + 2v$ nodes finish its work.

**Case 1** ($s_2 = 1$): Each iteration is finished when two out of three cells successfully transmit the result to PS. We first consider the case where $v > s_1$. Then, the iteration is not finished even when $n - s_1 - 1$ nodes are finished except for the $s_1 + 1$ nodes in a single overlapping area. After one of them is completed, the recovery threshold is achieved. Hence,

$$K = n - s_1. \tag{36}$$

Now consider the case $v \le s_1$. Denote the three cells as cell $A$, cell $B$ and cell $C$. We first select $u$ nodes in the non-overlapping area of a specific cell $A$. Then we alternately select nodes from the overlapping area between cell $A$, $B$ and the overlapping area between cell $A$, $C$. If $u + 2v - s_1$ nodes are selected from cell $B$ or $C$, we stop selecting and count the number of selected nodes which becomes the recovery threshold. If all the nodes in the corresponding overlapping areas are selected but the condition is not satisfied, we alternately select nodes in the non-overlapping areas in cell $B$ and $C$. Again if $u + 2v - s_1$ nodes are selected from cell $B$ or $C$, we stop selecting and count the number of nodes. Hence, the recovery threshold is written as $u + 2x - 1$ where $x = u + 2v - s_1$. Hence, we have

$$K = 3(u + v) + v - 2s_1 - 1 = n + v - 2s_1 - 1. \tag{37}$$

**Case 2** ($s_2 = 2$): For this case, each iteration is finished if only one cell is successfully finished. We first consider the case $s_1 \ge 2v$. Then, we alternately select the nodes in the non-overlapping areas for each cell. If $u + 2v - s_1$ nodes are selected from a single cell, the recovery threshold is achieved. Therefore, the recovery threshold becomes

$$\begin{align} K &= 3(u + 2v - s_1 - 1) + 1 \tag{38} \\ &= 3(u + v) + 3v - 3s_1 - 2 \tag{39} \\ &= n + 3v - 3s_1 - 2. \tag{40} \end{align}$$

Now we consider $s_1 < 2v$. We first select $u$ nodes from each cell in the non-overlapping areas. By alternately selecting the nodes in each overlapping area, the recovery threshold is achieved if $u + 2v - s_1$ nodes are selected from any single cell.

(i) $s_1 | 2$: $K$ becomes $3u + 3x - 1$ where $2x = 2v - s_1$. Hence, we have

$$K = 3u + 3\Big(\frac{2v - s_1}{2}\Big) - 1 = n - \frac{3}{2}s_1 - 1 \tag{41}$$

(ii) $s_1 \nmid 2$: $K$ becomes $3u + 3x - 2$ where $2x - 1 = 2v - s_1$. Then, we have

$$K = 3u + 3\Big(\frac{2v - s_1 + 1}{2}\Big) - 2 = n - \frac{3}{2}s_1 - \frac{1}{2} \tag{42}$$

## References

[1] J. Dean *et al.*, "Large scale distributed deep networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1223–1231.

[2] M. Li, "Scaling distributed machine learning with the parameter server," in *Proc. Int. Conf. Big Data Sci. Comput. BigDataScience*, 2014, pp. 583–598.

Authorized licensed use limited to: Korea Advanced Inst of Science & Tech - KAIST. Downloaded on April 09,2021 at 01:39:55 UTC from IEEE Xplore. Restrictions apply.

[3] M. Li, D. G. Andersen, A. J. Smola, and K. Yu, "Communication efficient distributed machine learning with the parameter server," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 19–27.
[4] B. Recht, C. Re, S. Wright, and F. Niu, "Hogwild: A lock-free approach to parallelizing stochastic gradient descent," in *Proc. Adv. Neural Inf. Process. Syst.*, 2011, pp. 693–701.
[5] X. Lian, C. Zhang, H. Zhang, C.-J. Hsieh, W. Zhang, and J. Liu, "Can decentralized algorithms outperform centralized algorithms? A case study for decentralized parallel stochastic gradient descent," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 5330–5340.
[6] D. Datla *et al.*, "Wireless distributed computing: A survey of research challenges," *IEEE Commun. Mag.*, vol. 50, no. 1, pp. 144–152, Jan. 2012.
[7] K. Yang, T. Jiang, Y. Shi, and Z. Ding, "Federated learning via over-the-air computation," *IEEE Trans. Wireless Commun.*, vol. 19, no. 3, pp. 2022–2035, Mar. 2020.
[8] Y. He, J. Ren, G. Yu, and Y. Cai, "D2D communications meet mobile edge computing for enhanced computation capacity in cellular networks," *IEEE Trans. Wireless Commun.*, vol. 18, no. 3, pp. 1750–1763, Mar. 2019.
[9] F. Wang, J. Xu, X. Wang, and S. Cui, "Joint offloading and computing optimization in wireless powered mobile-edge computing systems," *IEEE Trans. Wireless Commun.*, vol. 17, no. 3, pp. 1784–1797, Mar. 2018.
[10] J. Ren, G. Yu, Y. He, and G. Y. Li, "Collaborative cloud and edge computing for latency minimization," *IEEE Trans. Veh. Technol.*, vol. 68, no. 5, pp. 5031–5044, May 2019.
[11] D. Tse and P. Viswanath, *Fundamentals of Wireless Communication*. Cambridge, U.K.: Cambridge Univ. Press, 2005.
[12] 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.
[13] Q. Yu, M. Maddah-Ali, and S. Avestimehr, "Polynomial codes: An optimal design for high-dimensional coded matrix multiplication," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 4403–4413.
[14] A. Reisizadeh, S. Prakash, R. Pedarsani, and A. S. Avestimehr, "Coded computation over heterogeneous clusters," *IEEE Trans. Inf. Theory*, vol. 65, no. 7, pp. 4227–4242, Jul. 2019.
[15] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Coded Fourier transform," in *Proc. 55th Annu. Allerton Conf. Commun., Control, Comput. (Allerton)*, Oct. 2017, pp. 494–501.
[16] S. Dutta, V. Cadambe, and P. Grover, "Coded convolution for parallel and distributed computing within a deadline," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jun. 2017, pp. 2403–2407.
[17] S. Dutta, V. Cadambe, and P. Grover, "Short-dot: Computing large linear transforms distributedly using coded short dot products," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 2100–2108.
[18] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient coding: Avoiding stragglers in distributed learning," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2017, pp. 3368–3376.
[19] W. Halbawi, N. Azizan, F. Salehi, and B. Hassibi, "Improving distributed gradient descent using Reed–Solomon codes," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jun. 2018, pp. 2027–2031.
[20] S. Li, S. M. Mousavi Kalan, A. S. Avestimehr, and M. Soltanolkotabi, "Near-optimal straggler mitigation for distributed gradient methods," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops (IPDPSW)*, May 2018, pp. 857–866.
[21] N. Ferdinand, B. Gharachorloo, and S. C. Draper, "Anytime exploitation of stragglers in synchronous stochastic gradient descent," in *Proc. 16th IEEE Int. Conf. Mach. Learn. Appl. (ICMLA)*, Dec. 2017.
[22] E. Ozfatura, D. Gunduz, and S. Ulukus, "Speeding up distributed gradient descent by utilizing non-persistent stragglers," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jul. 2019, pp. 2729–2733.
[23] S. Kadhe, O. O. Koyluoglu, and K. Ramchandran, "Gradient coding based on block designs for mitigating adversarial stragglers," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jul. 2019, pp. 2813–2817.
[24] Q. Yu, S. Li, N. Raviv, S. M. M. Kalan, M. Soltanolkotabi, and S. Avestimehr, "Lagrange coded computing: Optimal design for resiliency, security and privacy," 2018, *arXiv:1806.00939*. [Online]. Available: https://arxiv.org/abs/1806.00939
[25] M. Ye and E. Abbe, "Communication-computation efficient gradient coding," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2018, pp. 5610–5619.
[26] N. Raviv, I. Tamo, R. Tandon, and A. G. Dimakis, "Gradient coding from cyclic mds codes and expander graphs," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2018, pp. 1–18.
[27] Z. Charles and D. Papailiopoulos, "Gradient coding using the stochastic block model," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jun. 2018, pp. 1–8.
[28] Z. Charles, D. Papailiopoulos, and J. Ellenberg, "Approximate gradient coding via sparse random graphs," 2017, *arXiv:1711.06771*. [Online]. Available: http://arxiv.org/abs/1711.06771
[29] L. Chen, H. Wang, Z. Charles, and D. Papailiopoulos, "Draco: Byzantine-resilient distributed training via redundant gradients," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2018, pp. 1–23.
[30] A. Reisizadeh, S. Prakash, R. Pedarsani, and A. S. Avestimehr, "Tree gradient coding," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jul. 2019, pp. 2808–2812.
[31] H. Park, K. Lee, J.-Y. Sohn, C. Suh, and J. Moon, "Hierarchical coding for distributed computing," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jun. 2018, pp. 1630–1634.
[32] A. Reisizadeh and R. Pedarsani, "Latency analysis of coded computation schemes over wireless networks," in *Proc. 55th Annu. Allerton Conf. Commun., Control, Comput. (Allerton)*, Oct. 2017, pp. 1256–1263.
[33] D.-J. Han, J.-Y. Sohn, and J. Moon, "Coded distributed computing over packet erasure channels," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jul. 2019, pp. 717–721.
[34] S. Li, Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Edge-facilitated wireless distributed computing," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2016, pp. 1–7.
[35] S. Li, Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "A scalable framework for wireless distributed computing," *IEEE/ACM Trans. Netw.*, vol. 25, no. 5, pp. 2643–2654, Oct. 2017.
[36] K. Yang, Y. Shi, and Z. Ding, "Data shuffling in wireless distributed computing via low-rank optimization," *IEEE Trans. Signal Process.*, vol. 67, no. 12, pp. 3087–3099, Apr. 2019.
[37] J. Konecny, H. B. McMahan, F. X. Yu, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," in *Proc. NIPS Workshop Private Multi-Party Mach. Learn.*, 2016, pp. 1–10.
[38] J. Konečný, H. Brendan McMahan, D. Ramage, and P. Richtárik, "Federated optimization: Distributed machine learning for on-device intelligence," 2016, *arXiv:1610.02527*. [Online]. Available: http://arxiv.org/abs/1610.02527
[39] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial Intelligence and Statistics*. PMLR, 2017, pp. 1273–1282.
[40] Y. Du, S. Yang, and K. Huang, "High-dimensional stochastic gradient quantization for communication-efficient edge learning," *IEEE Trans. Signal Process.*, vol. 68, pp. 2128–2142, 2020.
[41] L. Tong, Y. Li, and W. Gao, "A hierarchical edge cloud architecture for mobile computing," in *Proc. 35th Annu. IEEE Int. Conf. Comput. Commun. (INFOCOM)*, Apr. 2016, pp. 1–9.
[42] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1628–1656, 3rd Quart., 2017.
[43] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2322–2358, 4th Quart., 2017.
[44] N. Bhushan *et al.*, "Network densification: The dominant theme for wireless evolution into 5G," *IEEE Commun. Mag.*, vol. 52, no. 2, pp. 82–89, Feb. 2014.
[45] V. Chandrasekhar, J. Andrews, and A. Gatherer, "Femtocell networks: A survey," *IEEE Commun. Mag.*, vol. 46, no. 9, pp. 59–67, Sep. 2008.

**Dong-Jun Han** (Graduate Student Member, IEEE) received the B.S. degree in mathematics and electrical engineering and the M.S. degree in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST) in 2016 and 2018, respectively, where he is currently pursuing the Ph.D. degree. His research interests include distributed machine learning and information theory.

Authorized licensed use limited to: Korea Advanced Inst of Science & Tech - KAIST. Downloaded on April 09,2021 at 01:39:55 UTC from IEEE Xplore. Restrictions apply.

**Jy-Yong Sohn** (Member, IEEE) received the M.S. and Ph.D. degrees in electrical engineering from KAIST in 2016 and 2020, respectively. He is currently a Post-Doctoral Researcher with the Information and Electronics Research Institute, KAIST. His research interests include coding/information theory, distributed/federated learning, and robust machine learning. He received the IEEE International Conference on Communications (ICC) Best Paper Award in 2017, the KAIST EE Best Research Achievement Award in 2018, the KAIST Global Leader Fellowship in 2019, and the Qualcomm Innovation Award in 2015 and 2019.

**Jaekyun Moon** (Fellow, IEEE) received the Ph.D. degree in electrical and computer engineering from Carnegie Mellon University, Pittsburgh, PA, USA. From 1990 to early 2009, he was with the Faculty of the School of Electrical and Computer Engineering, University of Minnesota, Twin Cities. He consulted as the Chief Scientist for DSPG, Inc., from 2004 to 2007. He has also worked as the Chief Technology Officer at Link-A-Media Devices Corporation. He is currently a Professor of electrical engineering with KAIST. His research interests include channel characterization, signal processing and coding for data storage, and digital communication. He received the McKnight Land-Grant Professorship from the University of Minnesota. He received the IBM Faculty Development Awards as well as the IBM Partnership Awards. He was awarded the National Storage Industry Consortium (NSIC) Technical Achievement Award for the invention of the maximum transition run (MTR) code, a widely used error-control/modulation code in commercial storage systems. He has served as the Program Chair for the 1997 IEEE Magnetic Recording Conference. He is also the Past Chair of the Signal Processing for Storage Technical Committee of the IEEE Communications Society. He has served as a Guest Editor for the 2001 IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS Issue on Signal Processing for High Density Recording. He has also served as an Editor for IEEE TRANSACTIONS ON MAGNETICS in the area of signal processing and coding from 2001 to 2006.

Authorized licensed use limited to: Korea Advanced Inst of Science & Tech - KAIST. Downloaded on April 09,2021 at 01:39:55 UTC from IEEE Xplore. Restrictions apply.