Joint Dynamic Grouping and Gradient Coding for Time-critical Distributed Machine Learning in Heterogeneous Edge Networks

Yingchi Mao, Jun Wu, Xiaoming He, Ping Ping, Jiajun Wang, and Jie Wu, *Fellow, IEEE*

Abstract—In edge networks, distributed computing resources have been widely utilized to collaboratively perform a machine learning task by multiple nodes. However, the model training time in heterogeneous edge networks is becoming longer because of excessive computation and delay caused by slow nodes, namely stragglers. The parameter server even abandons stragglers which fail to return outcome within a reasonable deadline, called straggler dropout, decreasing the model accuracy. To optimize the computation cost and maintain the model accuracy, we focus on mitigating the heavy computation of stragglers and preventing straggler dropout. Therefore, we propose a novel scheme named Dynamic Grouping and Heterogeneity-aware Gradient Coding (DGH-GC) to tolerate stragglers by employing dynamic grouping and gradient coding. DGH-GC evenly distributes stragglers in each group and encodes gradients based on their computation capacity to prevent them drop out. However, DGH-GC exacerbates the communication burden by making data duplication to tolerate stragglers. Relying on the scheme, we further propose an algorithm called DGH-(GC)^2 to compress transferred gradients in both upstream communication and downstream communication. Experimental evaluations prove that DGH-(GC) outperforms all state-of-the-art methods and DGH-(GC)^2 further speeds up the convergence time of the trained model and saves about 26% average iteration time compared to the DGH-(GC).

Index Terms—Heterogeneous Edge Networks, Gradient Coding, Dynamic Grouping, Gradient Compression

1 Introduction

With the increasing computation power of edge devices [1], *e.g.*, smartphones [2] and IoT sensors [3], training Deep Neural Network (DNN) models on multiple edge devices becomes feasible. The distributed model training in edge networks takes advantage of the distributed parameter server (PS) architecture, where edge devices work as nodes and the edge server works as the central PS [4].

One of the main challenges for distributed DNN model training in heterogeneous edge networks lies in the heavy communication [5] load caused by excessive computation [6] and straggler dropout [7]. Specifically, some nodes may incur delays in computation or communication, which are called stragglers [8] [9]. As shown in Fig. 1, the PS waits for stragglers to submit their local gradients, and even abandons stragglers with long computing time, which are called straggler dropout [10] [11]. This largely increases the communication cost and reduces the accuracy of the training model [12]. Meanwhile, the attributes of heterogeneous nodes, *i.e.*, the computation capacity [13], memory size [14], and communication mode [15], further exacerbate straggler dropout, resulting in inefficient communication [16]. Besides, stragglers aggregation results in huge time consumption, leading to higher electricity costs and heavier greenhouse gas emissions [17] [18]. Therefore, it is vital to design an efficient solution of straggler dropout for green communication and computing [19] [20].

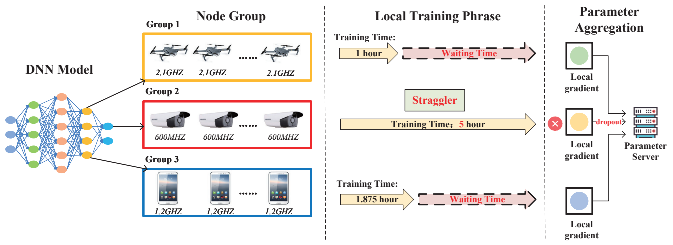


Fig. 1. Heavy communication load caused by stragglers in heterogeneous edge networks

The straggler dropout has been widely studied in distributed computing [21]. From the perspective of parallelism mechanism, the typical Time Asynchronous Parallel (TAP) [22] and Staleness Synchronous Parallel (SSP) [23] were proposed to mitigate the negative impact of stragglers on the communication efficiency. Based on the SSP algorithm, Dynamic Stochastic Gradient Descent (DynSGD) [24] adjusted the learning rate dynamically according to the delay of stragglers in completing gradient computation. Considering that the above parallelism schemes alleviated the model accuracy, Raviv *et al.* [25] proposed gradient Coding (GC) to tolerate stragglers by encoding gradients. However, GC

- *Y. Mao and P. Ping are with the Key Laboratory of Water Big Data Technology of Ministry of Water Resources. Y. Mao, J. Wu, X. He, P. Ping, J. Wang are with the College of Computer and Information, Hohai University, Nanjing, China.*
  E-mail: yingchimao@hhu.edu.cn, 1606010225@hhu.edu.cn, isxmhe@gmail.com, pingpingnjust@163.com, 211307040017@hhu.edu.cn
- *J. Wu is with the Center for Networked Computing, Temple University, 1805 N. Broad Street, Philadelphia, PA 19122.*
  E-mail: jiewu@temple.edu

is only applicable to homogeneous training environments, which does not work in heterogeneous edge networks [26]. Given the heterogeneity of edge devices, Wang *et al.* [27] proposed the Heterogeneity-aware Gradient Coding (HGC) to allocate data of reasonable size to nodes depending on their computation capacity, which tolerates a predetermined number of stragglers to prevent them from dropping out.

Some researchers have also reduced the chance of straggler dropout by grouping stragglers. Tang *et al.* [35] grouped all participating nodes into several clusters using k-Means clustering. Kopparapu *et al.* [38] proposed the Federated Cloning-and-Deletion (FedCD) to group nodes with similar data for addressing the non-IID problem in federated learning. Both of the above approaches employ the characteristics of nodes or data features carried by nodes to achieve the static grouping of stragglers. However, they fail to consider the fact that stragglers may drop out after the static grouping.

To address the shortcomings of the static grouping for stragglers, Kopparapu *et al.* [39] improved FedCD and used a dynamic grouping approach called Fork-Merge-Consolidate (FedFMC) that first dynamically forked nodes into updating different global models, then merged and consolidated separate models into one. Buyukates *et al.* [28] proposed Dynamic Clustering with Gradient Coding (DCGC) with the goal of dynamically adjusting the number of stragglers in each cluster, based on the straggler dropout in the previous iteration, making stragglers uniformly dispersed into each cluster to the maximum extent. However, FedFMC and DCGC ignores the heterogeneity of edge devices and fails to fully utilize the computation power of nodes for data allocation.

Considering the strong heterogeneity of nodes and the inappropriate groupings for stragglers that leads to straggler dropout in edge networks, we adopt a novel grouping for stragglers. It firstly employs the static grouping and then utilizes the dynamic grouping to improve the rationality of the static grouping for stragglers. Besides, a heterogeneity-aware gradient coding is applied in both grouping phrases. We propose a scheme termed Dynamic Grouping and Heterogeneity-aware Gradient Coding (DGH-GC). Firstly, DGH-GC initializes the static grouping based on the computation capacity of nodes and adjusts stragglers dynamically in each group according to the dropout frequency of stragglers, to evenly distribute stragglers in each group and tolerate more stragglers to mitigate excessive computation in heterogeneous edge networks. Secondly, DGH-GC fully exploits the computation resource of heterogeneous nodes for data allocation and encodes gradients that adapt to the computation power of stragglers, reducing delays caused by straggler dropout. Generally speaking, DGH-GC solves heavy computation of stragglers and reduces straggler dropout in heterogeneous edge networks.

However, DGH-GC increases the amount of data transferred between nodes and the parameter server by making data duplication for tolerating stragglers, which exacerbates the communication burden. To achieve the goal of optimizing the communication time in edge computing, we further propose a new algorithm, namely DGH-$(GC)^2$, to compress transferred gradients during the DNN training. Specifically, DGH-$(GC)^2$ employs a combination of gradient sparsification and ternary quantization to compress both upstream communication and downstream communication. Through this way, DGH-$(GC)^2$ can optimize the communication volume caused by tolerating more stragglers, which achieves both computation efficiency and communication efficiency. Extensive experimental evaluations are conducted on multiple popular machine learning tasks on nodes with different sizes and computation. Results prove that DGH-(GC) can decrease the model training time while maintaining the model accuracy compared to state-of-the-art methods. In addition, DGH-$(GC)^2$ further speeds up the convergence time of the trained model and saves about 26% average iteration time compared to the DGH-(GC). This paper is an extension of our previous work. The main contributions in this work include:

- Two static groupings based on a greedy algorithm and Karmarkar-Karp (KK) algorithm [29] are used to evenly distribute stragglers in each group, mitigating inherent heterogeneity gaps in edge networks. Given the fact that the static grouping ignores dynamic straggler dropout during the actual training, a dynamic grouping based on the frequency of straggler dropout is proposed.
- Considering the heterogeneity of nodes in the dynamic grouping, we propose a novel scheme termed DGH-GC. Specifically, DGH-GC allocates reasonable data to stragglers and encodes gradients depending on their computation capacity, which tolerates a predetermined number of stragglers to prevent them from dropping out and mitigates delay caused by straggler dropout.
- Furthermore, we propose a novel algorithm DGH-$(GC)^2$ based on DGH-GC with a combination of gradient sparsification and ternary quantization to compress gradients, which reduces the transferred data both in the upstream and downstream communication.
- Evaluations demonstrate that DGH-GC outperforms all baselines on nodes with different scales and computation capacity and DGH-$(GC)^2$ further accelerates the convergence time of the trained model and optimizes the communication time compared to DGH-GC.

The rest of this paper is organized as follows. Section 2 presents the related work. After that, a system framework and the problem formulation are proposed in Section 3. In Section 4, the algorithm design of DGH-GC is discussed in detail. Then, we show an improved algorithm DGH-$(GC)^2$ based on DGH-GC in Section 5. Next, experiments are performed to evaluate the accuracy and communication efficiency in Section 6. Finally, conclusions are drawn in Section 7.

2 Related Work

Mitigating the impact of stragglers on the communication efficiency in distributed systems: three approaches can be classified into the necessity of non-coding, coding, and grouping.

2.1 Non-coding approach

Considering the parallel mechanism of distributed computing, some scholars adopted non-coding approaches of asynchronous communication [30] to solve the problem of stragglers communication. SSP [23] aimed at a homogeneous cluster. In this case, if each node is executed asynchronously and added with certain control logic, SSP can mitigate the communication impact caused by stragglers in each iteration at the cost of a certain model convergence accuracy. SSP leads to a research boom of straggler communication problems in the field of non-coding mode. However, for the algorithm itself, SSP is not suitable for a heterogeneous cluster, where SSP can prolong the communication waiting time and even lead to the non-convergence of the DNN training model. Considering the lagging nature of the local param update in the heterogeneous environment, DynSGD [24] made an improvement on the SSP, that is, DynSGD assigns different learning rates to the param update of each node. The learning rate is small for the updates generated by stragglers but large for non-stragglers. Overall, DynSGD adjusts the learning rate dynamically according to the delay of stragglers in completing gradient calculation, thereby reducing the communication cost and enhancing the accuracy of the training model.

Although SSP and DynSGD can effectively solve the problem of stragglers communication, distributed algorithms using asynchronous communication mechanisms may lead to a certain degree of degradation of the accuracy of the DNN training model. Even if the training model has a complex structure, and the edge environment is full of heterogeneity, the asynchronous communication algorithms severely affect the convergence of the training model [31] [32].

2.2 Coding approach

To ensure the accuracy and convergence of the training model while mitigating the impact of stragglers, gradient coding approaches are introduced into distributed edge computing. Raviv *et al.* [25] proposed Gradient Coding (GC) which was not the same as traditional direct data coding. GC encodes the gradients generated by using the optimization algorithm, so as to eliminate the constraints of training linear models. Specifically, GC stores redundant data between nodes, allowing for the tolerance of any stragglers in each iterative training. Besides, it avoids the waste of computing power and the degradation of model accuracy caused by the discarding of computational gradients from stragglers. The Gradient Coding with Multi-Message Communication (MMC) proposed by Ozfatura *et al.* [33] reduced the communication time of each iteration in DNN training. Ozfatura also proposed a Gradient Coding with Clustering (GCC) and then applied the gradient coding within each group. In the case that stragglers are uniformly distributed in each group, GCC can tolerate multiple stragglers simultaneously without increasing the calculation load. Gradient coding approaches, such as GC, MMC, and GGC are only applicable to a homogeneous environment.

However, the modern edge distributed training environment was heterogeneous [34]. The uniform grouping of stragglers cannot be ensured, which makes the traditional

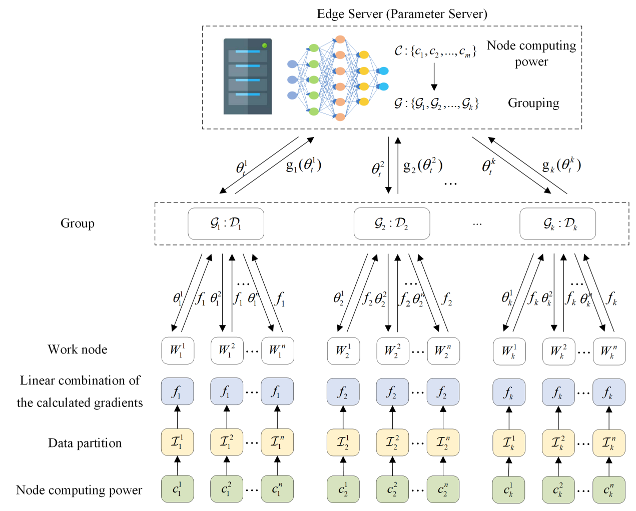

Edge Server (Parameter Server)
$\mathcal{C}:\{c_1,c_2,...,c_m\}$
Node computing power
$\mathcal{G}:\{\mathcal{G}_1,\mathcal{G}_2,...,\mathcal{G}_k\}$
Grouping
θ_t^1
$g_1(\theta_t^1)$
θ_t^2
$g_2(\theta_t^2)$
θ_t^k
$g_k(\theta_t^k)$
Group
$\mathcal{G}_1:\mathcal{D}_1$
$\mathcal{G}_2:\mathcal{D}_2$
$\mathcal{G}_k:\mathcal{D}_k$
Work node
W_1^1 W_1^2 W_1^n
W_2^1 W_2^2 W_2^n
W_k^1 W_k^2 W_k^n
Linear combination of the calculated gradients
f_1 f_2 f_k
Data partition
$\mathcal{I}_1^1$ $\mathcal{I}_1^2$ $\mathcal{I}_1^n$
$\mathcal{I}_2^1$ $\mathcal{I}_2^2$ $\mathcal{I}_2^n$
$\mathcal{I}_k^1$ $\mathcal{I}_k^2$ $\mathcal{I}_k^n$
Node computing power
c_1^1 c_1^2 c_1^n
c_2^1 c_2^2 c_2^n
c_k^1 c_k^2 c_k^n

Fig. 2. Distributed Computing in Heterogeneous Edge Networks

gradient coding schemes unsuitable for edge computing. In view of this, the Heterogeneity-aware Gradient Coding (HGC) [27] proposed to tolerate a predetermined number of stragglers, and made full use of the computing power of heterogeneous nodes for data allocation and the reduction of the average calculation time for iterative training. Compared with the traditional gradient coding scheme, the calculation time for training a model by using the HGC approach is reduced by three times.

2.3 Grouping approach

Grouping approaches are employed to mitigate the negative impact of stragglers on communication efficiency in heterogeneous edge networks. The essence of grouping approaches is to cluster nodes. Tang *et al.* [35] grouped all participating nodes into several clusters using k-Means clustering, based on their statistical characteristics to mitigate the problem of statistical heterogeneity and negative transfers of the generalization. Ghosh *et al.* [36] proposed an Iterative Federated Clustering Algorithm (IFCA) in which nodes were distributed and partitioned into clusters. IFCA alternately estimated the cluster identities of nodes and optimized model parameters for the node clusters via gradient descent. Sattler *et al.* [37] used the gradient consin similarity to segment nodes to avoid stragglers that withdraw from the federated learning system. Simply speaking, different clusters' gradient directions have significant divergences and they used the consin similarity to group nodes. Kopparapu *et al.* [38] proposed the Federated Cloning-and-Deletion (FedCD) to group nodes with similar data for addressing the non-IID problem in federated learning.

To maintain efficient gradient computation during each iterative training, the above approaches divide stragglers by the static grouping, without considering the dropout of stragglers during the actual training process. On this basis, Kopparapu *et al.* [39] improved FedCD and used a dynamic grouping approach called Fork-Merge-Consolidate (FedFMC) that first dynamically forked nodes into updating different global models, then merged and consolidated separate models into one. Buyukates *et al.* [28] proposed

Dynamic Clustering based on Gradient Coding (DCGC). On the basis of GGC, DCGC uses dynamic clustering to accelerate the gradient calculation, and dynamically adjusts the number of stragglers in each group, according to the dropout of training nodes in the previous iteration. DCGC realizes the uniform distribution of stragglers in the group, increasing tolerable stragglers' amount in the subsequent training process and reducing the communication waiting time.

However, DCGC does not consider the heterogeneity of nodes in the gradient calculation, which will waste the computational resources of nodes, and reduce the accuracy of the training model. Therefore, DGH-GC proposed in this paper can make full use of the computing power of heterogeneous nodes for data distribution, and encodes gradients that adapt to the computing power of stragglers to prevent stragglers from dropping out. In the meantime, DGH-GC can dynamically adjust the number of stragglers in each group based on the dropout frequency of stragglers, thereby tolerating more stragglers, which improves the efficiency of the edge distributed communication without reducing the model accuracy.

3 System Framework and Problem Formulation

Suppose that the distributed computing system in heterogeneous edge environment has m nodes $\{W_1, W_2, ..., W_m\}$, the computing power is expressed as $\mathcal{C}:\{c_1, c_2, ..., c_m\}$, as shown in Fig. 2. Considering the heterogeneity of nodes, parameter server (PS) divides these nodes into k groups, expressed as $\{\mathcal{G}_1,\mathcal{G}_2,...,\mathcal{G}_k\}$, with equivalent computing power in each group based on node computing power before training, and the data set $\mathcal{D}$ is also evenly divided into k non overlapping copies, expressed as $\{\mathcal{D}_1, \mathcal{D}_2..., \mathcal{D}_k\}$. Each group of nodes calculates one of the data $\mathcal{D}_i$, then the partial gradient $g_i(\theta_t)$ is calculated based on the data $\mathcal{D}_i$. The PS aggregates the partial gradient calculated by each node to restore the full gradient $g(\theta_t)$ as,

$$g(\theta_t) = \sum_{i=1}^{k} g_i(\theta_t), \tag{1}$$

where θ_t means the parameter θ of the model trained in t iteration round.

Each group has n nodes ($\lceil n{=}\frac{m}{k}\rceil$, n is an integer) and the group-based approach is used for m nodes. All groups can be expressed as $\{W_1^1, W_1^2, ..., W_1^n; W_2^1, W_2^2, ..., W_2^n; ...; W_k^1, W_k^2 ..., W_k^n\}$ and the set of all grouped node computing power is expressed as $\mathcal{C}:\{c_1^1, c_1^2, ..., c_1^n; c_2^1, c_2^2, ..., c_2^n; c_k^1, c_k^2, ..., c_k^n\}$.

The data is allocated according to node computing power within each group. Specifically, divide the data $\mathcal{D}_i$ of each group into d_i copies that do not overlap each other, represented as $\mathcal{D}_i:\{\mathcal{D}_i^1, \mathcal{D}_i^2, ..., \mathcal{D}_i^{d_i}\}$. d_i is calculated as,

$$d_i{=}c_i^{total}{=}\sum_{j=1}^{n} c_i^j, \tag{2}$$

where i denotes the group, j denotes the node in the group, c_i^{total} represents the total capacity of all nodes in i and c_i^j represents the computation capacity of the j in i. To tolerate s stragglers, each data in i needs to be redundantly stored r copies (r is the redundancy factor, $r{=}s+1$) and there are $d_i(s+1)$ copies of data in total. The number of data copies $amount_i^j$ allocated by j in i can be calculated as,

$$amount_i^j{=}d_i \cdot (s+1) \cdot \frac{c_i^j}{c_i^{total}}{=}(s+1) \cdot c_i^j. \tag{3}$$

$\mathcal{I}_i^j$ represents the data partition to be calculated as,

$$\begin{aligned}\mathcal{I}_i^j = \{&\mathcal{D}_i^{(index+1) \bmod d_i}, \mathcal{D}_i^{(index+2) \bmod d_i}, ...,\\ &\mathcal{D}_i^{(index+amount_i^j) \bmod d_i}\},\end{aligned} \tag{4}$$

where $index = \sum_{h=1}^{j-1} amount_i^h$ denotes the index of each group. The partial gradient calculated based on the data partition $\mathcal{D}_i^{d_i}$ is $g_i^{d_i}(\theta_t)$ and the partial gradient calculated from the node j in i is ,

$$\begin{aligned}\{&g_i^{(index+1) \bmod d_i}(\theta_t), g_i^{(index+2) \bmod d_i}(\theta_t), ...,\\ &g_i^{(index+amount_i^j) \bmod d_i}(\theta_t)\}.\end{aligned} \tag{5}$$

The intra-group gradient $g_i(\theta_t)$ of group i can be obtained by aggregating the partial gradients of each node within the group using a linear combination f_i as,

$$f_i = g_i(\theta_t){=}\sum_{d=1}^{d_i} g_i^d(\theta_t), \tag{6}$$

where f_i is the rule of gradient aggregation in group i.

After each group aggregates partial gradients based on f_i, the intra-group gradient $g_i(\theta_t)$ is passed to the PS.

The PS firstly aggregates the full gradient $g(\theta_t)$ of the global model, and then updates the model parameter by $\theta_{t+1} = \theta_t - \alpha g(\theta_t)$ and proceeds to the next iteration until the end of the training.

When using the system framework shown in Fig. 2 to train the DNN model, a synchronous communication mechanism is used for the distributed training. The time for each iteration depends on the group with the slowest training, which can be expressed as,

$$T^{(t)} = \max_{i\in\{1,2,...,k\}} \{T_i^{(t)}\}, \tag{7}$$

where $T^{(t)}$ denotes the overall iteration time after the number of training iteration t, k represents the number of groups and ${T_i}^{(t)}$ stands for the iteration time needed for the training of i after t training iteration. ${T_i}^{(t)}$ depends on the slowest working node within the group as,

$${T_i}^{(t)} = \max_{j\in\{1,2,...,n\}} \{Total\_T_i^{j\,(t)}\}, \tag{8}$$

where $Total\_T_i^{j\,(t)}$ represents t of j in i during the iteration training, and n represents the number of nodes in each group. $Total\_T_i^{j\,(t)}$ is made up of two parts: computation time $Comp\_T_i^{j\,(t)}$ and communication time $Comm\_T_i^{j\,(t)}$. That is,

$$Total\_T_i^{j\,(t)} = Comp\_T_i^{j\,(t)} + Comm\_T_i^{j\,(t)}. \tag{9}$$

- Computation time: the computation time of each node depends on the amount of data allocated and the computation capacity of the nodes. The computation time of each node is expressed as,

$$Comp\_T_i^{j\,(t)} = \frac{\left\| \mathcal{I}_i^j \right\|}{c_i^j}, \tag{10}$$

where $\left\| \mathcal{I}_i^j \right\|$ and c_i^j represent the data volume and the computation capacity of j in i, respectively.

- Communication time: The communication time is determined by the network condition and the amount of the transferred data. In the edge environment, the network connection performance of different edge devices varies as well as the large fluctuation of the network. Considering the complexity of the above reasons, we only assume that the communication time depends on the amount of the transferred data. $Comm\_T_i^{j\,(t)}$ is used to indicate the communication time of j of i in t.

To sum up, the total time for each iteration of the training can be expressed as,

$$T^{(t)} = \max_{i\in[1,k]} \{ \max_{j\in[1,n]} \{ \frac{\left\| \mathcal{I}_i^j \right\|}{c_i^j} + Comm\_T_i^{j\,(t)} \} \}. \tag{11}$$

To improve the communication efficiency of the DNN model training, the overall optimization goal is to minimize the total time of each iteration training as,

$$\arg\min T^{(t)}. \tag{12}$$

For ease of reading, the main notations used in this paper are summarized in the following Table 1.

TABLE 1
Symbols

Symbol	Definition
m	Total number of nodes
n	Number of nodes in each group
$\{W_1, W_2, ..., W_m\}$	node set
$\mathcal{C}$	Computing power
$\{c_1, c_2, ..., c_m\}$	node computing power set
k	Number of groups
$\{\mathcal{G}_1,\mathcal{G}_2,...,\mathcal{G}_k\}$	node group set
$\{\mathcal{D}_1, \mathcal{D}_2..., \mathcal{D}_k\}$	Data set divided into k copies
$g_i(\theta_t)$	Partial gradient based on group i
$g(\theta_t)$	Full gradient based on all groups
W_k^n	node n in group k
c_k^n	Computing power of node n in group k
d_i	Number of divided data in group i
r	Redundancy factor
s	Number of tolerable stragglers
$amount_i^j$	Number of data duplication
$\mathcal{I}_i^j$	Data partition by $amount_i^j$
f_i	Linear combination of partial gradients

Due to the reduction of the model accuracy caused by straggler dropout, tolerating a maximum number of stragglers to take full utilization of their computing resources becomes necessary. However, the more stragglers, the more significantly the computing time of the system will increase. Trading off the model accuracy and total system time influenced by the number of stragglers is worth considering. We propose DGH-GC to solve the above problem. It makes each node possible to have a similar completion time so that the consistent straggler dropout incurred by heterogeneity could be eliminated.

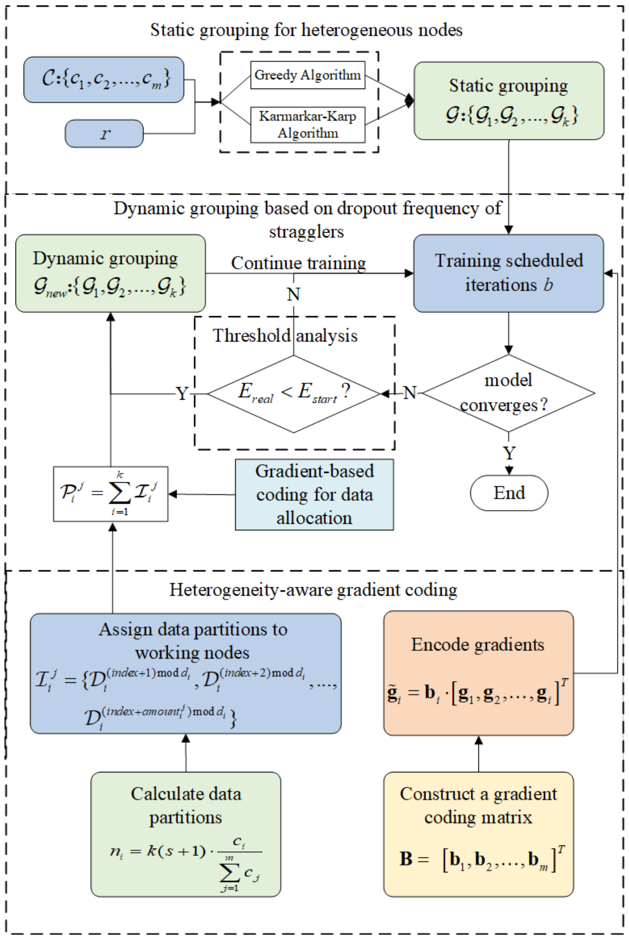

Static grouping for heterogeneous nodes
$\mathcal{C}:\{c_1,c_2,...,c_m\}$
r
Greedy Algorithm
Karmarkar-Karp Algorithm
Static grouping $\mathcal{G}:\{\mathcal{G}_1,\mathcal{G}_2,...,\mathcal{G}_k\}$
Dynamic grouping based on dropout frequency of stragglers
Dynamic grouping $\mathcal{G}_{new}:\{\mathcal{G}_1,\mathcal{G}_2,...,\mathcal{G}_k\}$
Continue training
Training scheduled iterations b
N
Threshold analysis
$E_{real} < E_{start}$?
Y
N
model converges?
Y
End
$\mathcal{P}_i^j = \sum_{i=1}^{k} \mathcal{I}_i^j$
Gradient-based coding for data allocation
Heterogeneity-aware gradient coding
Assign data partitions to working nodes
$\mathcal{I}_i^j = \{\mathcal{D}_i^{(index+1)\bmod d_i}, \mathcal{D}_i^{(index+2)\bmod d_i}, ..., \mathcal{D}_i^{(index+amount_i^j)\bmod d_i}\}$
Encode gradients
$\tilde{\mathbf{g}}_i = \mathbf{b}_i \cdot [\mathbf{g}_1, \mathbf{g}_2, ..., \mathbf{g}_i]^T$
Calculate data partitions
$n_i = k(s+1) \cdot \frac{c_i}{\sum_{j=1}^{m} c_j}$
Construct a gradient coding matrix
$\mathbf{B} = [\mathbf{b}_1, \mathbf{b}_2, ..., \mathbf{b}_m]^T$

Fig. 3. General idea of DGH-GC

4 DGH-GC

DGH-GC is proposed for the overall optimization objective. The implementation process of DGH-GC, as shown in Fig. 3, is divided into two steps. The first step refers to grouping heterogeneous nodes, including static grouping and dynamic grouping based on the dropout frequency of stragglers. That is, stragglers are evenly distributed in each group and more stragglers are tolerated to improve communication efficiency. For the second step, the heterogeneity-aware gradient coding is applied within each group to make full use of the computation power of heterogeneous nodes and encode gradients to prevent them from dropping out.

4.1 Static Grouping and Dynamic Grouping

In the static grouping, all nodes are divided into k groups. Each group tolerates $r - 1$ stragglers. k is limited by the following factors,

- To tolerate more stragglers than HGC, set $k > r - 1$.
- The number of data copies should satisfy $amount_i^j \leq d_i$ to obtain $r \cdot \frac{c_i^j}{c_i^{total}} \leq 1$. To satisfy the case that all nodes have the most computing power, $c_i^{total} \geq r \cdot c_{max}$ is required, where c_{max} represents the node

with the most computing power, and the number of k has to satisfy $k \leq \frac{c^{total}}{r \cdot c_{max}}$.

In summary, the number of groups k should satisfy the following,

$$k \in (r-1, \frac{c^{total}}{r \cdot c_{max}}]. \tag{13}$$

We design a static grouping approach based on the greedy algorithm and Karmarkar-Karp (KK) [29] to obtain the optimal grouping for stragglers. The greedy algorithm is less complex than the differential algorithm, but always makes the best choice when solving the problem. It is easy to fall into the local optimal and the final result is not global optimal. However, the KK considers the global optimum in each selection. Therefore, it is necessary to decide which grouping approach to use based on the set $\mathcal{C} : \{c_1, c_2, ..., c_m\}$ and k values of each node's computing power. After obtaining group's number k, the elements in set $\mathcal{C}$ are sorted first based on nodes' computing power and then the differences among the k elements are compared, starting from the first element. When the difference is small, that is, when the distribution of numbers is relatively uniform, the greedy algorithm can be used to get the optimal grouping and the KK can be used while the difference is large. The specific process of the static grouping is as follows,

1) Firstly, input the set $\mathcal{C} : \{c_1, c_2, ..., c_m\}$ of the computation power of m nodes and the redundancy factor r. Then, set $k = k_{max}$ using (13). Last, sort set $\mathcal{C}$ in decreasing order.

2) Judge differences in set $\mathcal{C}$ [29]. If differences are small, the static grouping based on the greedy algorithm is employed: Initialize k empty sets $\{\mathcal{G}_1, \mathcal{G}_2, ..., \mathcal{G}_k\}$. Consider one element in set $\mathcal{C}$ at a time, and place it in the subset with the smallest sum so far. In the case of subsets with equal sum, choose any subset until set $\mathcal{C}$ is empty. While differences are large, we use the KK to realize the static grouping: A list of k-tuples is created for each of the integers in set $\mathcal{C}$. The first integer of this tuple is the value in set $\mathcal{C}$ and the rest of the integers are set to 0. Combine the first two tuples in the list (the two tuples with the largest integers). Given the tuples $A = (a_1, a_2, ..., a_n)$ and $B = (b_1, b_2, ..., b_n)$, both sorted in decreasing order, A is combined with B to form a new tuple C as,

$$C = (a_1 + b_n, a_2 + b_{n-1}, ..., a_n + b_1). \tag{14}$$

After the combination of tuples A and B, the KK algorithm keeps track of the relative differences between subsets only, and normalizes set C by subtracting its minimum value. This merging process continues until only one tuple remains and the algorithm ends.

3) Determine whether the sum of each subset satisfies the condition $c_i^{total} \geq r \cdot c_{max}$. If so, output the group. Otherwise, make the groups' amount $k = k - 1$, and re-execute the static grouping approach.

After the static grouping, the dynamic grouping is employed to improve the rationality of groupings for stragglers. Dynamic grouping based on dropout frequency of stragglers includes two key steps, *i.e.*, threshold analysis and dynamic adjustment.

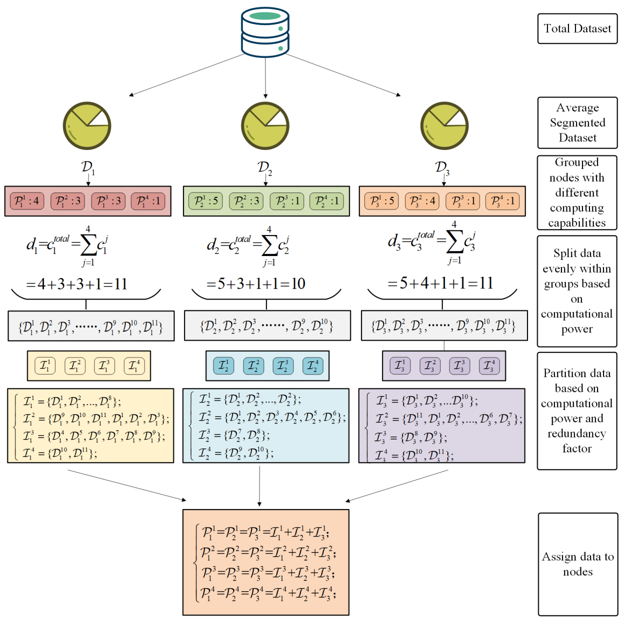

Total Dataset
Average Segmented Dataset
Grouped nodes with different computing capabilities
Split data evenly within groups based on computational power
Partition data based on computational power and redundancy factor
Assign data to nodes
$\mathcal{D}_1$
$\mathcal{D}_2$
$\mathcal{D}_3$
$\mathcal{P}_1^1:4$ $\mathcal{P}_1^2:3$ $\mathcal{P}_1^3:3$ $\mathcal{P}_1^4:1$
$\mathcal{P}_2^1:5$ $\mathcal{P}_2^2:3$ $\mathcal{P}_2^3:1$ $\mathcal{P}_2^4:1$
$\mathcal{P}_3^1:5$ $\mathcal{P}_3^2:4$ $\mathcal{P}_3^3:1$ $\mathcal{P}_3^4:1$
$d_1 = c_1^{total} = \sum_{j=1}^{4} c_1^j = 4+3+3+1 = 11$
$d_2 = c_2^{total} = \sum_{j=1}^{4} c_2^j = 5+3+1+1 = 10$
$d_3 = c_3^{total} = \sum_{j=1}^{4} c_3^j = 5+4+1+1 = 11$
$\{\mathcal{D}_1^1, \mathcal{D}_1^2, \mathcal{D}_1^3, \cdots\cdots, \mathcal{D}_1^9, \mathcal{D}_1^{10}, \mathcal{D}_1^{11}\}$
$\{\mathcal{D}_2^1, \mathcal{D}_2^2, \mathcal{D}_2^3, \cdots\cdots, \mathcal{D}_2^9, \mathcal{D}_2^{10}\}$
$\{\mathcal{D}_3^1, \mathcal{D}_3^2, \mathcal{D}_3^3, \cdots\cdots, \mathcal{D}_3^9, \mathcal{D}_3^{10}, \mathcal{D}_3^{11}\}$
$\mathcal{I}_1^1$ $\mathcal{I}_1^2$ $\mathcal{I}_1^3$ $\mathcal{I}_1^4$
$\mathcal{I}_2^1$ $\mathcal{I}_2^2$ $\mathcal{I}_2^3$ $\mathcal{I}_2^4$
$\mathcal{I}_3^1$ $\mathcal{I}_3^2$ $\mathcal{I}_3^3$ $\mathcal{I}_3^4$
$\mathcal{I}_1^1 = \{\mathcal{D}_1^1, \mathcal{D}_1^2, ..., \mathcal{D}_1^8\};$
$\mathcal{I}_1^2 = \{\mathcal{D}_1^9, \mathcal{D}_1^{10}, \mathcal{D}_1^{11}, \mathcal{D}_1^1, \mathcal{D}_1^2, \mathcal{D}_1^3\};$
$\mathcal{I}_1^3 = \{\mathcal{D}_1^4, \mathcal{D}_1^5, \mathcal{D}_1^6, \mathcal{D}_1^7, \mathcal{D}_1^8, \mathcal{D}_1^9\};$
$\mathcal{I}_1^4 = \{\mathcal{D}_1^{10}, \mathcal{D}_1^{11}\};$
$\mathcal{I}_2^1 = \{\mathcal{D}_2^1, \mathcal{D}_2^2, ..., \mathcal{D}_2^2\};$
$\mathcal{I}_2^2 = \{\mathcal{D}_2^1, \mathcal{D}_2^2, \mathcal{D}_2^3, \mathcal{D}_2^4, \mathcal{D}_2^5, \mathcal{D}_2^6\};$
$\mathcal{I}_2^3 = \{\mathcal{D}_2^7, \mathcal{D}_2^8\};$
$\mathcal{I}_2^4 = \{\mathcal{D}_2^9, \mathcal{D}_2^{10}\};$
$\mathcal{I}_3^1 = \{\mathcal{D}_3^1, \mathcal{D}_3^2, ... \mathcal{D}_3^{10}\};$
$\mathcal{I}_3^2 = \{\mathcal{D}_3^{11}, \mathcal{D}_3^1, \mathcal{D}_3^2, ..., \mathcal{D}_3^6, \mathcal{D}_3^7\};$
$\mathcal{I}_3^3 = \{\mathcal{D}_3^8, \mathcal{D}_3^9\};$
$\mathcal{I}_3^4 = \{\mathcal{D}_3^{10}, \mathcal{D}_3^{11}\};$
$\mathcal{P}_1^1 = \mathcal{P}_2^1 = \mathcal{P}_3^1 = \mathcal{I}_1^1 + \mathcal{I}_2^1 + \mathcal{I}_3^1;$
$\mathcal{P}_1^2 = \mathcal{P}_2^2 = \mathcal{P}_3^2 = \mathcal{I}_1^2 + \mathcal{I}_2^2 + \mathcal{I}_3^2;$
$\mathcal{P}_1^3 = \mathcal{P}_2^3 = \mathcal{P}_3^3 = \mathcal{I}_1^3 + \mathcal{I}_2^3 + \mathcal{I}_3^3;$
$\mathcal{P}_1^4 = \mathcal{P}_2^4 = \mathcal{P}_3^4 = \mathcal{I}_1^4 + \mathcal{I}_2^4 + \mathcal{I}_3^4;$

Fig. 4. Data allocation process based on heterogeneity-aware gradient coding

4.1.1 *Analysis of Threshold*

Firstly, the number of iterations b for static grouping training is predetermined. Then, it is determined whether to dynamically adjust stragglers groups depending on the actual expectation after b training times. The specific steps are as follows,

- Calculate the expectation E_{start} of the tolerable straggler number based on the assumption that k nodes has equal probability to become stragglers in each iteration.
- Perform b iterations, and calculate the actual expectation E_{real} of the number of tolerable stragglers by the frequency of becoming a straggler of each node. Determine whether the actual expectation E_{real} reaches the threshold, *i.e.*, the static expectation E_{start}. If not, that is $E_{real} < E_{start}$, the effect of static grouping is poor and has to be dynamically adjusted until E_{start}=E_{real}. Otherwise, no adjustment is needed.

4.1.2 *Dynamic Adjustment*

In general, dynamic adjustment includes three steps. Firstly, heterogeneity-aware gradient coding is applied within each group to assign data $\mathcal{I}_i^j$ to nodes. That is,

$$\begin{aligned} \mathcal{I}_i^j = \{&\mathcal{D}_i^{(index+1) \bmod d_i}, \mathcal{D}_i^{(index+2) \bmod d_i}, ..., \\ &\mathcal{D}_i^{(index+amount_i^j) \bmod d_i}\}, \end{aligned} \tag{15}$$

where $index$ denotes the index of nodes in the group, d_i means dividing Data $\mathcal{D}_i$ into d_i copies equally, and $amount_i^j$ indicates the number of data copies by j in i. Then, to make intergroup nodes replaceable, the nodes with comparable computing power between groups need to have data from each other. That is, allocate alternative data $\mathcal{P}_i^j = \sum_{i=1}^{k} \mathcal{I}_i^j$ to the nodes, where k represents the number of groups. Secondly, estimate k nodes with high probability to become stragglers based on previous iterations. Thirdly, set the group priority of stragglers according to their number and assign $n-r+1$ non-stragglers to each group based on the priority, which tolerates $r-1$ stragglers. If non-stragglers are not enough, offer the priority to the non-stragglers and then assign the alternative nodes of stragglers based on the data allocation $\mathcal{P}_i^j$ until all groups are allocated.

The data allocation process based on heterogeneity-aware gradient coding is shown in Fig. 4. For example, when the Set $\mathcal{I}_i^j$ is {5, 5, 4, 4, 3, 3, 3, 1, 1, 1, 1} and the redundancy factor r is 2, the optimal static grouping using the KK is $\mathcal{G}_1:\{4,3,3,1\}, \mathcal{G}_2:\{5,3,1,1\}, \mathcal{G}_3:\{5,4,1,1\}$. The data allocation process for each group of nodes is as follows. Firstly, the dataset $\mathcal{D}$ is divided equally into $k=3$ copies, *i.e.*, $\{\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3\}$. Within each group, the data set $\mathcal{D}_i$ is divided equally into $d_i = c_i^{total} = \sum_{j=1}^{n} c_i^j$, where $n = 12/3 = 4$ and $d_1 = 11, d_2 = 10, d_3 = 11$. Then the initial data is divided by (15). Finally, the divided data is assigned to each node.

4.2 Gradient Coding for Dynamic Grouping

Besides realizing the node grouping, DGH-GC also fully exploits the computation resources of heterogeneous nodes for data allocation and encodes gradients that adapt to the computation power of stragglers, reducing delays caused by straggler dropout. For data allocation, each data partition $\mathcal{I}_i^j$ has to be assigned with at least $s+1$ nodes to tolerate s stragglers by using (15). For the coding phrase, to prevent stragglers from dropping out, gradients coding matrix B is employed in each group as,

$$\mathbf{B} = [\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_m]^T, \tag{16}$$

where $b_j = 1$ if $\mathcal{D}_i^j \in \mathcal{I}_i^j$, else $b_j = 0$. By constructing coding matrix B, we encode gradients that adapt to the computation power of stragglers as,

$$\widetilde{\mathbf{g}}_i = \mathbf{b}_i \cdot [\mathbf{g}_1, \mathbf{g}_2, \ldots, \mathbf{g}_i]^T, \tag{17}$$

where g_i means the gradient that aggregates the partial gradient of each node in Group i, and $\widetilde{\mathbf{g}}_i$ represents encoded gradients by B for the global gradient aggregation of the server.

We describe the workflow of DGH-GC in Algorithm 1. The core idea of DGH-GC is to initialize the static grouping based on the computation capacity of nodes before the distributed training and adjust stragglers dynamically in each group according to the dropout frequency of stragglers during the training. Besides, gradient coding is applied in the grouping phrase to allocate reasonable amounts of data to stragglers, and in the coding phrase to encode and decode gradients transmitted between the server and nodes.

4.3 DGH-GC Complexity

The complexity of greedy algorithm and Karmarkar-Karp algorithm are both $O(2^m)$ [29], where m is the number of integers in the input set $\mathcal{C}$. Considering the dynamic grouping, the complexity is $O(s)$ with s stragglers. Given the gradient code, the complexity is $O\left(\frac{1}{\sqrt{kT}}\right)$ with the group number k and iteration time T [27]. Thus, the overall complexity of DGH-GC is $O\left(2^m + s + \frac{1}{\sqrt{kT}}\right)$, which is slightly higher than HGC.

Algorithm 1 Dynamic Grouping and Heterogeneity-aware Gradient Coding(DGH-GC)

```
Input: $k$: number of groups, $r$: redundancy number, $\mathcal{C}:\{c_1, c_2, ..., c_m\}$: computation power of nodes, $b$: training time for the static grouping, $\mathcal{I}_i^j$: data partition for the node j in the group i, $n$: number of nodes in each group, $m$: total number of nodes, $s$: total number of stragglers
Output: $\widetilde{\mathbf{g}}_i$: encoded gradient
1:  Initialize the static grouping:
2:  Set $k = k_{max}$ by using (13)
3:  Judge differences in Set $\mathcal{C}:\{c_1, c_2, ..., c_m\}$
4:  if differences are small then
5:      static grouping based on the greedy algorithm
6:  else
7:      static grouping based on the Karmarkar-Karp algorithm
8:  end if
9:  Conduct the dynamic grouping:
10: Perform $b$ distributed training
11: Calculate $E_{start}$ by the probability to become a straggler for each node
12: Calculate $E_{real}$ by the frequency to become a straggler for each node
13: if $E_{real} < E_{start}$ then
14:     repeat
15:         Allocate data $\mathcal{I}_i^j$ to all nodes by using (15)
16:         Assign $n - r+1$ non-stragglers to each group to tolerate $r-1$ stragglers
17:     until $E_{start}$=$E_{real}$
18: else
19:     break
20: end if
21: Encode gradients based on Heterogeneity-aware Gradient Coding by using (16-17)
22: return $\widetilde{\mathbf{g}}_i$
```

5 DGH-GC WITH GRADIENT COMPRESSION

Due to the reduction of the model accuracy caused by straggler dropout, tolerating a maximum number of stragglers to take full utilization of their computing resources becomes necessary. However, DGH-GC increases the computation time in the total system time when tolerating more stragglers by making data duplication, which fails to meet the optimal solution to the problem (12). Considering that the model accuracy is positively correlated with the computation time, achieving a high model accuracy with low computation time is difficult to handle.

Since formula (11) gives that the total system time includes not only the computation time, but also the communication time, optimizing the communication time to reduce the total system time is a feasible method. It can be seen from reference [40] that nodes will upload many useless gradients to the parameter server, which increases the amount of transferred data during the communication phase. Gradient compression is a common lossy compression technology to reduce the communication cost. Therefore, we propose a new gradient compression scheme based on DGH-GC (Dynamic Grouping and Heterogeneity-aware

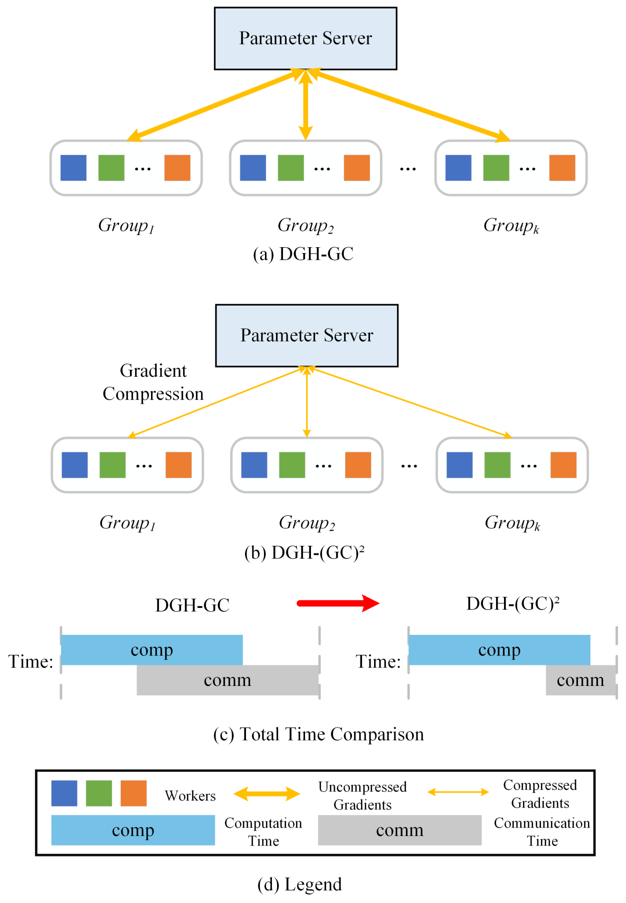


Fig. 5. Comparison Between DGH-GC and DGH-(GC)²

Gradient Coding with Gradient Compression, called DGH-(GC)²) to make up for the increased computing time caused by tolerating stragglers.

5.1 Algorithm Design

The core idea of DGH-(GC)² is to reduce the number of gradient bits and unimportant gradients by introducing the sparse ternary quantization gradient compression, thus compressing the amount of communication volume between the node and the parameter server after grouping. Many gradient compression algorithms adopt the one-way compression, including the upstream communication by compressing the transferred data from nodes to parameter servers or the downstream communication from parameter servers to nodes. Different from the one-way compression, DGH-(GC)² combines upstream compression and downstream compression to allow for efficient communication in heterogeneous edge networks. As shown in Fig. 5, DGH-(GC)² compresses gradients between different node groups and the parameter server relying on the DGH-GC scheme. Compared to DGH-GC, DGH-(GC)² decreases the communication time and maintains the same computation time in the total system time.

5.1.1 Upstream Compression

In each round of distributed DNN training, firstly, all nodes in different groups download the model parameter θ_{t-1}^i broadcast by the parameter server. Then, the DGH-GC algorithm is employed to allocate data for nodes with different computing power and encode gradients in line with stragglers to prevent them drop out. In the following, gradient sparsification and ternary quantization are performed to reduce the communication time.

Considering the fact that top-k sparsification shows the most promising performance in time-critical distributed learning environments, we adopt the method described in [40] to communication the fraction of the largest elements at full precision. Firstly, the factor m that sparses gradients needs to be determined as,

$$m = \max(np, 1), \tag{18}$$

where n represents the number of nodes in each group, and p means the sparsity. After aggregated gradients $\tilde{\mathrm{g}}_{\mathrm{i}}$ of Group i are encoded by DGH-GC, the first m important gradients are transmitted during each round of DNN training as,

$$s \leftarrow top_m(|\tilde{\mathrm{g}}_{\mathrm{i}}|), \tag{19}$$

where top_m is the sparsity function, and s is the sparse gradient.

Higher compression gains can be achieved when sparsification is combined with quantization of the non-zero elements. We quantize the remaining top-m elements of the sparsified updates to a ternary tensor. Several operations are taken to reduce the communication cost as,

$$mask = \varepsilon(|\tilde{\mathrm{g}}_i| - s), \tag{20}$$

$$\mathrm{g}_{\mathrm{i}}^{\mathrm{t}} = sign(mask \odot \tilde{\mathrm{g}}_{\mathrm{i}}), \tag{21}$$

where ε is the step function, and the sparse gradient s is used as the threshold of generating a $mask$ for quantization. $\odot$ is the Hadamard product, and $sign$ is the quantization function. Finally, quantized ternary gradient g_i^t after sparsification in Group i during the training round t is computed.

The above significantly reduces the size of updated gradients from nodes in all groups to the parameter server, thus reducing the cost of the upstream communication. However, if no additional measures are taken, the cost of downstream communication will not decrease. To further eliminate the remaining sources of redundancy in the communication, the next section will introduce the downstream compression at the server-side.

5.1.2 Downstream Compression

The server aggregates all compressed local gradients after sparse ternary quantization to generate the global gradient g^{t}. At the server side, the same ternarization is used to update the global model parameter θ_t^i in each node group as,

$$\theta_t^i = \theta_{t-1}^i + \eta * sign(mask \odot \mathrm{g}^{\mathrm{t}}), \tag{22}$$

where θ_{t-1}^i represents the model parameter trained by Group i in the $t-1$ round of iteration. η is the learning rate and g^{t} means the aggregation gradient after the t round of training by nodes in all groups.

Therefore, the server compresses the normalized global model again from the perspective of downstream communication and pushes the quantized global model to all nodes in different groups.

Different from standard edge computing algorithms, DGH-(GC)² compresses communications in the both upload and download stages, which brings the main advantage when deploying time-critical distributed machine learning tasks for resource-constrained nodes. The overall framework of the proposed DGH-(GC)² is summarized in Algorithm 2.

Algorithm 2 Dynamic Grouping and Heterogeneity-aware Gradient Coding with Gradient Compression

```
Input: k: number of groups, r: redundancy number, b: training time for the static grouping, n: number of nodes, p: sparsity, θ: global model parameter
Init: Broadcast θ to each node group
1: for round $t = 1, \ldots, T$ do
2:   for $\mathcal{G}_i \in \{\mathcal{G}_1, \mathcal{G}_2, ..., \mathcal{G}_k\}$ in parallel do
3:     node group $\mathcal{G}_i$ does:
4:         Download parameter $\theta_{t-1}^i$
5:         Compute local gradient $g_i(\theta_{t-1}^i)$
6:         $\tilde{g}_i \leftarrow$ DGH-GC$(g_i(\theta_{t-1}^i), r, b)$
7:         $m \leftarrow \max(np, 1)$
8:         $s \leftarrow top_m(|\tilde{g}_i|)$
9:         $mask \leftarrow \varepsilon(|\tilde{g}_i| - s)$
10:        $g_i^t = sign(mask \odot \tilde{g}_i)$
11:        Upload $g_i^t$ to Parameter Server
12:    end
13:  end for
14:  Parameter Server does:
15:      $g^t = \frac{1}{k} \sum_{i=1}^{k} g_i^t$
16:      $mask \leftarrow \varepsilon(|g^t| - s)$
17:      $\theta_t^i \leftarrow \theta_{t-1}^i + \eta * sign(mask \odot g^t)$
18:      Broadcast $\theta_t^i$ to node group $\mathcal{G}_i$
19:  end
20: end for
```

5.2 DGH-(GC)² Complexity

Relying on the DGH-GC scheme, DGH-(GC)² achieves a combination of gradient sparsification and ternarization to further compress gradients. Considering that the complexity of top-k sparsification and ternarization is $O(n)$ [40], where n represents the number of gradients transferred between nodes and the parameter server, therefore, the overall complexity of DGH-(GC)² is $O\left(2^m + s + \frac{1}{\sqrt{kT}} + n\right)$, which is more complicated than DGH-GC.

6 EXPERIMENTS

6.1 Experiment Setup

The experiments are carried out with two DELL PowerEdge R740 servers and 20 nodes with different CPU processing power. Each server is equipped with two 28-core Intel Xeon Platinum 8180M CPUs and one NVIDIA GeForce RTX 3090 GPU. The memory capacity of each server is 93GB. The memory capacity of each CPU node is 32GB. The software platform of our experiments is PyTorch, which is a deep learning experimental platform that provides a high degree of flexibility and efficiency.

TABLE 2
EXPERIMENTAL SETUP FOR nodes

Groups	Number of nodes with different CPU processing power (pcs)				
	600MHZ	1.2GHZ	1.8GHZ	2.4GHZ	3GHZ
Group-A	5	0	3	2	2
Group-B	5	3	3	4	4

Experiments focus on comparing the model training accuracy and the communication efficiency of DGH-GC with four gradient coding approaches: GC, HGC, GGC and DCGC. The experimental models are selected from the Convolutional Neural Network (CNN) model and the LeNet5 network model. MNIST and CIFAR-10 are used for experimental datasets.

The number of iterations I is set to 500, the learning rate α is 0.01, and the redundancy factor r is set to 2. Besides, the data transmission rate between nodes and servers is 3-5 MB/s. The settings are similar to [27]. In addition, the experimental settings of nodes are shown in Table 2. Group A has 12 nodes. Specifically, the number of nodes with five types of CPU processing power - 600MHZ, 1.2GHZ, 1.8GHZ, 2.4GHZ, 3GHZ are 5, 0, 3, 2, and 2, respectively. Group B has 20 nodes, and the number of nodes with five types of CPU processing capabilities are 5, 3, 3, 4 and 5, respectively. According to Equation (13), Group A is divided into 3 groups at the most, and Group B contains 5 groups maximally. Experiments are conducted based on the above preparation, and the evaluation metrics of the experiments include the communication efficiency and the model accuracy.

6.2 Experiments with DGH-GC

6.2.1 Training Time b

The param value b in DGH-GC indicates that one dynamic adjustment is executed after b rounds of iterations. The b value affects the number of dynamic groupings and the grouping effect of DGH-GC. Five different b values are selected for the comparative experiments on 12 nodes and 20 nodes, respectively. The experimental results are shown in Table 3 and 4, and each value is averaged after 10 independent experiments.

TABLE 3
DGH-GC AT DIFFERENT b VALUES FOR 12 nodes WITH 3 GROUPS

PARAM b	CNN(MNIST)		CNN(CIFAR-10)		LeNet5(CIFAR-10)	
	Time(s)	*Accuracy%*	*Time(s)*	*Accuracy%*	*Time(s)*	*Accuracy%*
100	1.02	95.3	0.96	62.6	1.1	64.4
50	**0.67**	95.4	**0.64**	62.6	**0.68**	64.8
30	0.79	95.5	0.76	62.5	0.82	65
20	0.88	95.4	0.84	62.4	0.93	64.5
10	0.95	95.2	0.92	62.8	1.02	64.7

TABLE 4
DGH-GC AT DIFFERENT b VALUES FOR 20 nodes WITH 5 GROUPS

PARAM b	CNN(MNIST)		CNN(CIFAR-10)		LeNet5(CIFAR-10)	
	Time(s)	*Accuracy%*	*Time(s)*	*Accuracy%*	*Time(s)*	*Accuracy%*
100	0.90	93.6	0.92	60.9	0.96	62.6
50	**0.56**	93.6	**0.54**	61.2	**0.59**	62.5
30	0.68	93.8	0.64	60.8	0.70	62.8
20	0.75	93.4	0.72	61.0	0.81	62.4
10	0.87	93.2	0.83	61.1	0.89	62.5

Table 3 shows the experimental results of the distributed training with different b values at 12 nodes. The experimental results of the training of the MNIST dataset on the CNN model are shown in Column 2 and 3 of Table 3. The accuracy corresponding to different b values are 95.3%, 95.4%, 95.5%, 95.4%, and 95.2%, respectively with a small gap. According to the experimental results, the difference of b values has almost no effect on the model training accuracy. However, in terms of communication efficiency, the shortest average iteration time is 0.67s when $b = 50$, while the longest is 1.02s when $b = 100$. It can be seen from the experiments that the average iteration time increases with the decrease of b values from the optimal value $b = 50$. The b value of 30, 20 and 10 corresponds to the average iteration time of 0.79s, 0.88s, and 0.95s, respectively.

The experimental results of the training of the CIFAR-10 dataset on the CNN model are shown in Column 4 and 5 of Table 3. Compared with the MNIST dataset, the CIFAR-10 dataset is more complex, and therefore has a lower accuracy rate when training the same model. However, due to the smaller number of training samples in the CIFAR-10 dataset, the time used to train the CNN model is shorter. The model accuracy is not greatly affected by the b value, which is maintained at 62.6%. Among all the results, the shortest average iteration time is 0.64s when $b = 50$, while the longest one is 0.96s when $b = 100$. Similarly, the average iteration time increases gradually when the b value decreases from 50 gradually.

The experimental results of the training of the CIFAR-10 dataset on the LeNet5 model are shown in the last two columns of Table 3. Basically, the b value does not affect the accuracy of the training model. The average iteration time is the shortest when $b = 50$, and DGH-GC has the best communication effect on the optimization of the distributed training.

Table 4 shows the results of the distributed training with different b values at 20 nodes. It can be observed from Table 4 that the value of b can hardly affect the model accuracy. When $b = 100$, the communication cost with DGH-GC is the least efficient. This is due to the fact that the number of samples is large enough, yet the number of groupings is small and close to static grouping, leading to the worst communication efficiency, while the best communication efficiency is achieved when $b = 50$. The average iteration time of the distributed training increases with the decrease of the value of b. This is because when the value of b decreases gradually, the number of dynamic groupings becomes large, which leads to certain time delay. Secondly, due to the small number of samples, the results of stragglers dynamic grouping are not ideal initially, which reduces the communication efficiency. The value of b is determined as 50 by experiments, and applied to the comparison experiments below.

6.2.2 Average Iteration Time

Fig. 6 shows the experimental results of the average iteration time based on five approaches, *i.e.*, GC, HGC, GCC, DCGC, and DGH-GC for training the MNIST dataset and CIFAR-10 dataset on the CNN model and the CIFAR-10 dataset on the LeNet5 model under 12 and 20 nodes, respectively.

From Fig. 6, it can be seen that HGC, GCC, DCGC and DGH-GC perform significantly better than the traditional GC algorithm in terms of communication efficiency, regardless of whether there are 12 or 20 nodes.

DCGC combines the idea of dynamic grouping based on GCC so as to tolerate more stragglers and make its communication efficiency higher than GCC. The experimental results show that the communication efficiency of DCGC outperforms that of GCC by about 12%.

The communication efficiency of GCC and DCGC is higher than that of HGC in the distributed training for 12 nodes, but is lower than that of HGC in the distributed training for 20 nodes. The reason is that the heterogeneity of 12 nodes is stronger, while the heterogeneity becomes smaller in the distributed training for 20 nodes, although the number of nodes becomes larger. Therefore, in the 12-node experiment, HGC considers the heterogeneity of nodes, achieving a higher edge distributed communication efficiency.

In all experiments, DGH-GC achieves the highest communication efficiency. The reason is that firstly, DGH-GC not only considers node heterogeneity, but also combines the idea of dynamic grouping to distribute stragglers in each group to the maximum extent. Secondly, DGH-GC does not perform the dynamic grouping in each iteration, but sets a param b to reduce the number of adjustments, thus reducing the time consumption, compared with the dynamic grouping in DCGC. As can be seen from Fig. 6, DGH-GC remains optimal in terms of communication efficiency even though the network model has changed and becomes more complex. The average speedup of DGH-GC is about 2.3 times that of the traditional training GC, about 1.53 times higher than HGC, about 1.58 times higher than GCC, and about 1.45 times higher than DCGC.

6.2.3 Number of Groups

Fig. 7(a) shows the experimental results of 12 nodes, divided into 2 and 3 groups, respectively. The average iteration time when k = 3 is significantly lower than that when k = 2. Meanwhile, Fig. 7(b) shows the experimental results of 20 nodes, divided into groups 2, 4, and 5, respectively. It can be seen that the average iteration time becomes shorter with the increase of the number of groups k, and the communication efficiency when k =5 is about 1.9 times faster than that when k =2. The number of groups k determines the maximum number of stragglers that can be tolerated in

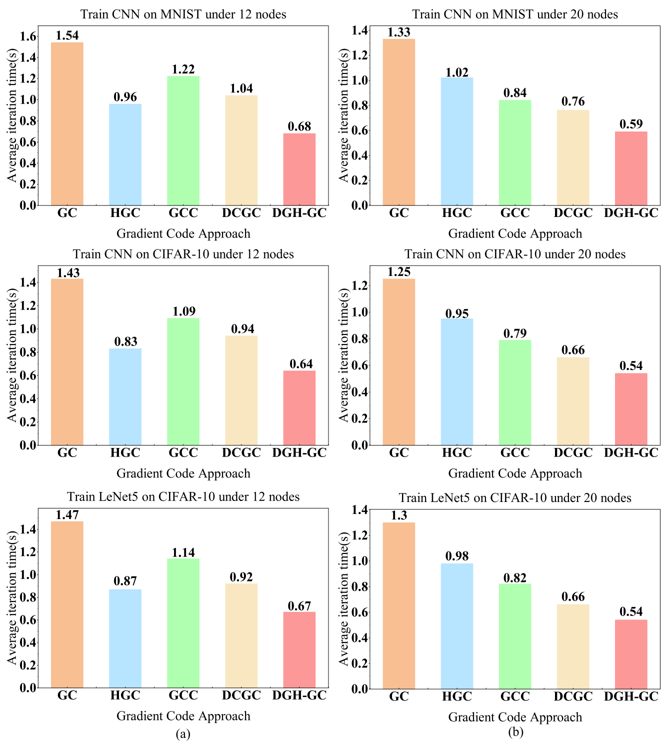


Fig. 6. Average iteration time of different gradient code approaches under 12 and 20 nodes

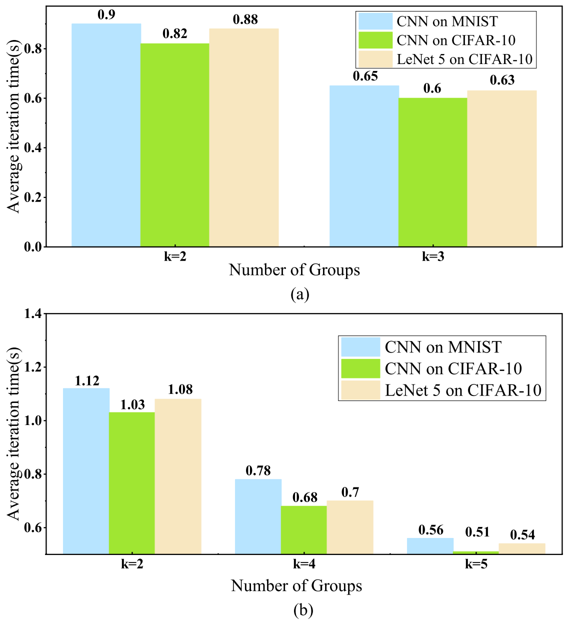


Fig. 7. Average iteration time of DGH-GC for 12 and 20 nodes in different groups

each iteration. The larger the k, the more obvious DGH-GC increases the number of stragglers tolerated in each iteration by dynamically adjusting the groups, resulting in faster iteration time and improvement of communication efficiency.

6.2.4 *Model Accuracy*

TABLE 5
TRAINING ACCURACY

APPROACH	ACCURACY(%)					
	CNN(MNIST)		CNN(CIFAR-10)		LeNet5(CIFAR-10)	
	m=12	m=20	m=12	m=20	m=12	m=20
GC	95.6	93.2	62.4	61.5	65.1	62.7
HGC	95.4	93.3	62.3	61.1	64.5	62.6
GCC	95.1	93.6	62.6	61.1	64.8	62.4
DCGC	95.7	93.4	62.9	61.4	65	62.4
DGH-GC	95.4	93.6	62.6	61.2	64.8	62.5
AVERAGE	95.4	93.4	62.5	61.2	64.8	62.5

Table 5 shows the experimental results of training accuracy under MNIST dataset and CIFAR-10 dataset on CNN model and under CIFAR-10 dataset on LeNet5 model based on five approaches, *i.e.*, GC, HGC, GCC, DCGC and DGH-GC at 12 nodes and 20 nodes, respectively.

As can be seen from Table 5, the accuracy of the five approaches, *i.e.*, GC, HGC, GCC, DCGC and DGH-GC under MNIST dataset on CNN model with 12 nodes and 20 nodes are 95.6%, 95.4%, 95.1%, 95.7%, 95.4% and 93.2%, 93.3%, 93.6%, 93.4%, 93.6% with the average accuracy of 95.4% and 93.4%, respectively. There is almost no difference between the accuracy and the average accuracy of the five approaches.

When replacing the dataset with CIFAR-10, the average accuracy of the five approaches is 62.5% (12 nodes) and 61.2% (20 nodes), trained on the CNN model. The results are much worse than that of MNIST dataset, which is because CIFAR-10 is a more complex color image dataset, leading to a lower accuracy when trained on the CNN model. However, the training accuracy of the five approaches still does not differ much.

When applying the network model to LeNet5 to train the CIFAR-10 dataset, the accuracy is improved by about 2% on average compared with the CNN model. It can be seen from the table that regardless of 12 or 20 nodes and whether the network model and dataset are changed, the accuracy of the five approaches remains slightly above or below the average value. The reason is that all these approaches utilize gradient encoding to store the data redundantly without changing the gradients obtained from each iteration of the aggregation. Therefore, DGH-GC does not suffer the decrease of the model training accuracy.

6.3 Comparison to DGH-GC

After exhibiting results of DGH-GC, we now present results of DGH-$(GC)^2$. We will prove that DGH-$(GC)^2$ further reduces the communication time for model training than DGH-GC through a combination of sparsification and quantization.

6.3.1 Sparsity p

Fig. 8 shows the experimental results of the accuracy based on DGH-(GC)² for training the MNIST dataset and CIFAR-10 dataset on the CNN model and the CIFAR-10 dataset on the LeNet5 model. The DGH-GC algorithm is used to group 12 nodes and 20 nodes respectively and determine the number of groups $k = 2$. Since we set the redundancy factor $r = 2$, data from $r - 1 = 1$ straggler is backed up for each group, and a total of 2 stragglers are tolerated to evaluate the effect of DGH-(GC)² on model accuracy at different sparsity p.

As can be seen in Fig. 8, the accuracy of the training model using the DGH-(GC)² algorithm is severely impaired as the sparsity p decreases, both with 12 nodes and 20 nodes. The reason is that the significant reduction of sparsity p leads to a sharp decrease in the number of gradients transmitted to the parameter server by the grouped nodes in each iteration round. The gradients valid for model updates are sieved out along with the invalid gradients.

Specifically, in Fig. 8(a), when 12 nodes are utilized to train the CNN model on the MNIST dataset, the accuracy of the training model decreases by 0.5%, 1.3%, 2.6%, 6.2% and 12.7% as the sparsity p varies by 0.05, 0.01, 0.005, 0.002 and 0.001, respectively, compared to the uncompressed model. When switching to 20 nodes with stronger heterogeneity to train the CNN model, with the reduction of sparsity p, the model accuracy decreases by 0.5%, 1.7%, 3.1%, 7.3% and 14.0% respectively. It is not difficult to conclude that the sparsity p plays a decisive role in the accuracy of the model, especially when the sparsity $p = 0.001$, the sparsity of gradients greatly reduces the accuracy of the model whether it is 12 nodes or 20 nodes. While p takes 0.05 and 0.01, gradient compression brings a slight global model loss.

Using a more complex dataset CIFAR-10 to train the CNN model, the relationship between the sparsity p of DGH-(GC)² and the model accuracy is shown in Fig. 8(b). The loss of model accuracy is more pronounced than training the MNIST dataset on the CNN model, especially in the case of training 20 nodes. This is because, with the increasing heterogeneity of nodes, differences between gradients computed by nodes in the same group are quite large, that is, values of gradients are unevenly distributed in each group, concentrated in the maximum and minimum. When using DGH-(GC)² to group 20 nodes, large gradients in the each group can determine the convergence speed and accuracy of the global model. Specifically, if large gradients are discarded while small gradients are retained, the model convergence slows down and the model accuracy is impaired seriously. As the enhancement of gradient sparsity, more large and valid gradients are abandoned during the distributed training, resulting in a large-scale accuracy loss.

In Fig. 8(c), the LeNet5 network we utilize is an optimization of the CNN network, which has more network layers and is more suitable for the classification of the CIFAR-10 dataset. Therefore, the loss of the model accuracy is significantly improved compared to that in Fig. 8(b). The optimized LeNet5 network can offset the interference of gradient sparsification to some extent because it has more layers to generate valid gradients and parts of them are saved. However, when taking the sparsity p less than 0.05, the global model accuracy trained with 12 nodes and 20 nodes is quite lower than that without compression. Specifically, the loss rate of the global model accuracy trained with 12 nodes and 20 nodes reaches 21.2% and 20.3% at $p = 0.001$.

From the above three sets of experiments, $p = 0.05$ is a reasonable parameter to compress gradients. Therefore, the following experiments are conducted at $p = 0.05$ to evaluate the optimization of the communication time when using DGH-(GC)².

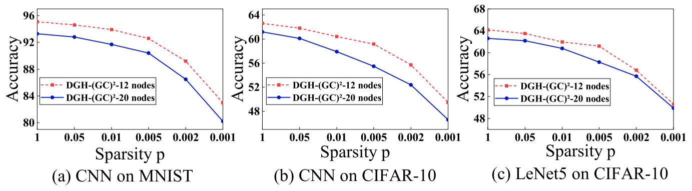


Fig. 8. Accuracy of DGH-(GC)² at different sparsity p under 12 and 20 nodes with 2 stragglers

6.3.2 Impact of Stragglers

We further discuss the performance comparison of DGH-GC and DGH-(GC)² over experiments with different numbers of stragglers. In Fig. 9, we group 12 nodes to tolerate 2 stragglers, and compare the model accuracy and average iteration time of two methods at a sparsity of $p = 0.05$. Employing the provided datasets and models for training, it can be observed that models trained with DGH-(GC)² converge at about 200 iterations, faster than with DGH-GC, which requires about 300 iterations to converge. However, the model accuracy trained by DGH-(GC)² is reduced. Meanwhile, DGH-(GC)² accelerates DGH-(GC) in terms of the average iteration time of the trained model by achieving approximately 25% time savings. This is because DGH-(GC)² compresses the amount of data transmitted between worker groups and the parameter server in edge computing through a combination of gradient sparsification and ternary quantization, thus achieving the proposed optimization goal by equation 12.

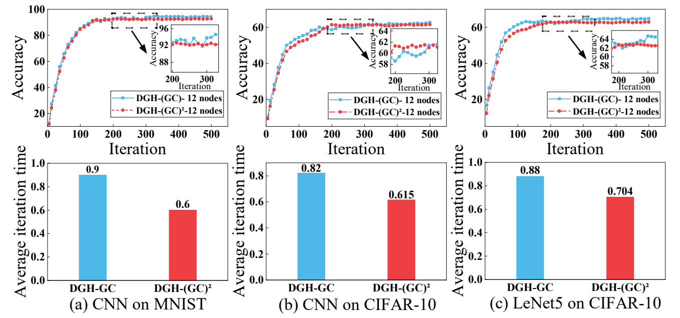


Fig. 9. Accuracy and average iteration time of DGH-(GC)² at sparsity $p = 0.05$ under 12 nodes with 2 stragglers

In Fig. 10, we set 3 stragglers to examine the impact of stragglers on the above two algorithms. The same experimental results as in Fig. 9 are still able to obtain, i.e., DGH-(GC)² speeds up the convergence of the model and decreases the time burden of the model training, but at the cost of compromised model accuracy. It is not difficult to observe

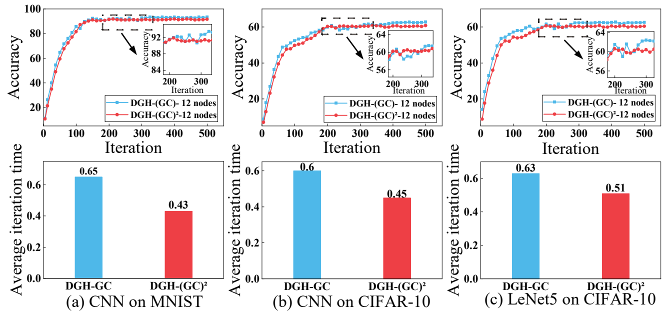


Fig. 10. Accuracy and average iteration time of DGH-(GC)² at sparsity $p = 0.05$ under 12 nodes with 3 stragglers

that increasing the number of stragglers reduces the average iteration time compared to Fig. 9. This is because when the redundancy factor r remains unchanged, the effect of backing up stragglers' data by dynamically adjusting groupings is more obvious. With the number of groups for stragglers increased, the parameter server obtains valid gradients for model updates earlier and speeds up the convergence time of the global model.

7 CONCLUSIONS

In the edge environment, the differences in computation power of terminal equipment lead to the severe straggler dropout in the distributed training, which reduces the model accuracy and communication efficiency. To achieve efficient computing and communication, in this paper, we firstly focus on dealing with the straggler dropout and then address the problem that tolerating stragglers increases the communication cost. First, we design a novel algorithm called DGH-GC to tolerate stragglers. Specifically, we employ static grouping and dynamic grouping to evenly distribute nodes with different computation capacity in each group. Grouping nodes decreases the frequency of straggler dropout and waiting time that the parameter server waits for stragglers to submit model updates. In addition, DGH-GC applies a heterogeneity-aware gradient coding to allocate reasonable data to stragglers based on their computation capacity and encode gradients to prevent stragglers from dropping out. Since DGH-GC increases the amount of data transferred between nodes and the parameter server by backing up the data of stragglers, we further propose a new algorithm, namely DGH-(GC)² that combines the gradient sparsification and ternary quantization to simultaneously compress upstream and downstream communication, thus greatly reduce the total time for model training. Finally, we evaluate our algorithms on multiple popular machine learning tasks under nodes with different scales and computation capacity. Experimental results show that DGH-GC outperforms all the state-of-the-art baselines and DGH-(GC)² further accelerates the convergence time of training models and reduces the communication time by about 26% more than DGH-GC.

ACKNOWLEDGMENTS

This work is supported by The National Natural Science Foundation of China (No. 61902110, 61832005), Postgraduate Research & Practice Innovation Program of Jiangsu Province (No. KYCX22_0610), The Fundamental Research Funds for The Central Universities (No. B210203024), Key Research and Development Project of Jiangsu Province (No. BE2020729), Science Technology Achievement Transformation of Jiangsu Province (No. BA2021002), Key Technology Project of China Huaneng Group (No.HNKJ19-H12, HNKJ20-H64).

REFERENCES

[1] F. Daghero, D. J. Pagliari, and M. Poncino, "Energy-efficient deep learning inference on edge devices," *Advances in Computers*, vol. 122, pp. 247-301, 2021.
[2] H. Salem, "Distributed computing system on a smartphones-based network," in *51st TOOLS 2019: Tatarstan*, Oct. 2019, pp. 313-325.
[3] Y. Liu, W. Yu, T. S. Dillon, W. Rahayu, and M. Li, "Empowering IoT predictive maintenance solutions with AI: A distributed system for manufacturing plant-wide monitoring", *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 3, pp. 683-697, Mar. 2022.
[4] Q. Zhou, S. Guo, H. Lu, L. Li, M. Guo, Y. Sun, and K. Wang, "Falcon: Addressing stragglers in heterogeneous parameter server via multiple parallelism," *IEEE Transactions on Computers*, vol. 70, no. 1, pp. 139-155, Jan. 2020.
[5] W. Li, D. Liu, K. Chen, K. Li, and H. Qi, "Hone: Mitigating stragglers in distributed stream processing with tuple scheduling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 8, pp. 2021-2034, Aug. 2021.
[6] Q. Zhou, S. Guo, Z. Qu, P. Li, L. Li, M. Guo, and K. Wang, "Petrel: Heterogeneity-aware distributed deep learning via hybrid synchronization," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 5, pp. 1030-1043, Dec. 2020.
[7] M. N. Krishnan, E. Hosseini, and A. Khisti, "Coded sequential matrix multiplication for straggler mitigation," *IEEE Journal on Selected Areas in Information Theory*, vol. 2, no. 3, pp. 830-844, Sep. 2021.
[8] J. Chen, and X. Ran, "Deep learning with edge computing: A review," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1655-1674, Aug. 2019.
[9] W. Liu, H. Liu, X. Liao, H. Jin, and Y. Zhang, "Straggler-aware parallel graph processing in hybrid memory systems," in *21st IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing, CCGrid 2021*, May. 2021, pp. 217-226.
[10] Q. Zhou, S. Guo, H. Lu, L. Li, M. Guo, Y. Sun, and K. Wang, "A comprehensive inspection of the straggler problem," *IEEE Computer*, vol. 54, no. 10, pp. 4-5, Oct. 2021.
[11] S. Okuno, M. Miwa, and N. Fukumoto, "Towards straggler-tolerant and accuracy-aware distributed DNN training in clouds," in *21st IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing, CCGrid 2021*, May. 2021, pp. 614-617.
[12] Y. Cai, and P. Yuan, "Time-varying mobile edge computing for capacity maximization," *IEEE Access*, vol. 8, pp. 142832-142842, Aug. 2020.
[13] A. Zhou, J. Yang, Y. Gao, T. Qiao, Y. Qi, X. Wang, Y. Chen, P. Dai, W. Zhao, and C. Hu, "Brief industry paper: Optimizing memory efficiency of graph neural networks on edge computing platforms," in *27th IEEE Real-Time and Embedded Technology and Applications Symposium*, Apr. 2021, pp. 445-448.
[14] J. Kai, H. Zhou, X. Chen, and H. Zhang, "Mobile edge computing for ultra-reliable and low-latency communications," *IEEE Communications Standards Magazine*, vol. 5, no. 2, pp. 68-75, Jun. 2021.
[15] T. Bahreini, H. Badri, and D. Grosu, "Mechanisms for resource allocation and pricing in mobile edge computing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 3, pp. 667-682, Mar. 2022.
[16] J. Zhang, and O. Simeone, "LAGC: Lazily aggregated gradient coding for straggler-tolerant and communication-efficient distributed learning," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 3, pp. 962-974, Mar. 2021.
[17] Y. Li, Q. K. Pan, K. Z. Gao, M. F. Tasgetiren, B. Zhang, and J. Q. Li, "A green scheduling algorithm for the distributed flowshop problem," *Applied Soft Computing*, vol. 109, no. 107526, Sep. 2021.

[18] B. Zhang, and Y. Liu, "Distributed resource allocation for green HetNets with renewable energy resources," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 35, no. 8, pp. 2159029:1-2159029:16, Aug. 2021.

[19] H. Lu, and K. Wang, "Distributed machine learning based mitigating straggler in big data environment," in *ICC 2021 - IEEE International Conference on Communications*, Jun. 2021.

[20] B. Tegin, E. E. Hernandez, S. Rini, and T. M. Duman, "Straggler mitigation through unequal error protection for distributed matrix multiplication," in *ICC 2021 - IEEE International Conference on Communications*, Jun. 2021.

[21] T. T. Vu, D. T. Ngo, H. Q. Ngo, M. N. Dao, N. H. Tran, and R. H. Middleton, "Straggler effect mitigation for federated learning in cell-free massive MIMO," in *ICC 2021 - IEEE International Conference on Communications*, Jun. 2021.

[22] G. Yuan, Z. Liao, X. Ma, Y. Cai, Z. Kong, X. Shen, J. Fu, Z. Li, C. Zhang, H. Peng, N. Liu, A. Ren, J. Wang, and Y. Wang, "Improving DNN fault tolerance using weight pruning and differential crossbar mapping for ReRAM-based edge AI," in *IEEE International Symposium on Quality Electronic Design*, Jun. 2021, pp. 135-141.

[23] G. Yuan, Z. Liao, X. Ma, Y. Cai, Z. Kong, X. Shen, J. Fu, Z. Li, C. Zhang, H. Peng, N. Liu, A. Ren, J. Wang, and Y. Wang, "Improving DNN fault tolerance using weight pruning and differential crossbar mapping for ReRAM-based edge AI," in *IEEE International Symposium on Quality Electronic Design*, Jun. 2021, pp. 135-141.

[24] X. Miao, X. Nie, Y. Shao, Z. Yang, J. Jiang, L. Ma, and B. Cui, "Heterogeneity-aware distributed machine learning training via partial reduce," in *SIGMOD 21th International Conference on Management of Data*, Jun. 2021, pp. 2262-2270.

[25] N. Raviv, I. Tamo, R. Tandon, and A. G. Dimakis, "Gradient coding from cyclic MDS codes and expander graphs," *IEEE Transactions on Information Theory*, vol. 66, no. 12, pp. 7475-7489, Dec. 2020.

[26] T. J. Nezhad, and M. A. Maddah-Ali, "Optimal Communication-Computation Trade-Off in Heterogeneous Gradient Coding," *IEEE Journal on Selected Areas in Information Theory*, vol. 2, no. 3, pp. 1002-1011, Sep. 2021.

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

[28] B. Buyukates, E. Ozfatura, S. Ulukus, and D. Gunduz, "Gradient coding with dynamic clustering for straggler mitigation," in *ICC 2021 - IEEE International Conference on Communications*, Jun. 2021.

[29] E. L. Schreiber, R. E. Korf, and M. D. Moffitt, "Optimal multi-way number partitioning," *Dissertations and Theses - Gradworks*, vol. 65, no. 4, pp. 24:1-24:61, Aug. 2018.

[30] J. Zhang, H. Zhang, W. Li, and K. Li, "Distributed edge-event triggered consensus control for multi-agent systems by edge-based asynchronous communications," *Applied Mathematics and Computation*, vol. 397, no. 125920, May. 2021.

[31] J. Liu, H. Xu, Y. Xu, Z. Ma, Z. Wang, C. Qian, and H. Huang, "Communication-efficient asynchronous federated learning in resource-constrained edge computing," *Computer Networks*, vol. 199, no. 108429, Nov. 2021.

[32] B. Kodavati, and M. Ramarakula, "Asynchronous gateway reallocation communication in heterogeneous 5G networks," *International Journal of Communication Systems*, vol. 34, no. 6, Jun. 2021.

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

[34] F. Chen, X. Chen, L. Xiang, and W. Ren, "Distributed economic dispatch via a predictive scheme: Heterogeneous delays and privacy preservation," *Automatica*, vol. 123, no. 109356, Jan. 2021.

[35] X. Tang, S. Guo, and J. Guo, "Personalized federated learning with clustered generalization," *CoRR*, vol. abs/2106.13044, 2021.

[36] A. Ghosh, J. Chung, D. Yin, and K. Ramchandran, "An efficient frameWork for clustered federated learning," in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020*, Dec. 2020.

[37] F. Sattler, K. R. Müller, and W. Samek, "Clustered federated learning: Model-agnostic distributed multi-task optimization under privacy constraints," *CoRR*, vol. abs/1910.01991, 2019.

[38] K. Kopparapu, E. Lin, and J. Zhao, "FedCD: Improving performance in non-iid federated learning," *CoRR*, vol. abs/2006.09637, 2020.

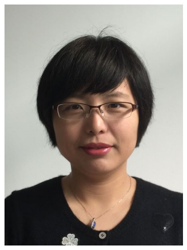
[39] K. Kopparapu, and E. Lin, "FedFMC: Sequential efficient federated learning on non-iid data," *CoRR*, vol. abs/2006.10937, 2020.

[40] F. Sattler, S. Wiedemann, K.-R. Müller, and W. Samek, "Sparse binary compression: Towards distributed deep learning with minimal communication," *CoRR*, vol. abs/1805.08768, 2018.

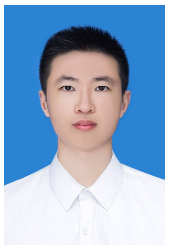
Yingchi Mao received her Ph. D. degree from the Department of Computer Science and Technology at the Nanjing University, Nanjing, China in 2007. She is currently a professor in the College of Computer and Information, Hohai University, Nanjing, China. She is a senior member of China Computer Federation (CCF) and Chinese Association of Automation (CAA). Her main research interests include edge intelligent computing, internet of things data analysis, and mobile sensing system.

Jun Wu received his B.S. degree from the College of Computer Science and Technology at the Hohai University, Nanjing, China in 2020. He is currently a Postgraduate in the College of Computer Science and Technology at the Hohai University, Nanjing, China. His research interests include distributed learning, edge computing, and federated learning.

Xiaoming He received his M.E. degree from Nanjing University of Posts and Telecommunications, Nanjing, China in 2019. He is currently a Ph.D. student in the College of Computer and Information, Hohai University, China. He is also a Visiting Research Fellow in the Pillar of Information Systems Technology and Design, Singapore University of Technology and Design. His research interests include deep reinforcement learning, edge computing, edge caching, and edge intelligence.



Ping Ping received her Ph.D. degree from the School of Computer Science and Engineering at the Nanjing University of Science and Technology, Nanjing, China in 2009. She is currently an assistant professor in the College of Computer and Information, Hohai University, Nanjing, China. Her research interests include network and information security, cloud computing and big data security, and image hiding encryption.



Jiajun Wang received his B.S. degree from the College of Computer Science and Technology at the Hohai University, Nanjing, China in 2021. He is currently a Postgraduate in the College of Computer Science and Technology at the Hohai University, Nanjing, China. His research interests include distributed learning, edge computing, and federated learning.

Jie Wu is the Director of the Center for Networked Computing and Laura H. Carnell professor at Temple University. He also serves as the Director of International Affairs at College of Science and Technology. He served as Chair of Department of Computer and Information Sciences from the summer of 2009 to the summer of 2016 and Associate Vice Provost for International Affairs from the fall of 2015 to the summer of 2017. Prior to joining Temple University, he was a program director at the National Science Foundation and was a distinguished professor at Florida Atlantic University. His current research interests include mobile computing and wireless networks, routing protocols, network trust and security, distributed algorithms, and cloud computing. Dr. Wu regularly publishes in scholarly journals, conference proceedings, and books. He serves on several editorial boards, including IEEE Transactions on Mobile Computing, IEEE Transactions on Service Computing, Journal of Parallel and Distributed Computing, and Journal of Computer Science and Technology. Dr. Wu is/was general chair/co-chair for IEEE IPDPS'08, IEEE DCOSS'09, IEEE ICDCS'13, ACM MobiHoc'14, ICPP'16, IEEE CNS'16, WiOpt'21, and ICDCN'22 as well as program chair/cochair for IEEE MASS'04, IEEE INFOCOM'11, CCF CNCC'13, and ICCCN'20. He was an IEEE Computer Society Distinguished Visitor, ACM Distinguished Speaker, and chair for the IEEE Technical Committee on Distributed Processing (TCDP). Dr. Wu is a Fellow of the AAAS and a Fellow of the IEEE. He is the recipient of the 2011 China Computer Federation (CCF) Overseas Outstanding Achievement Award.