2023 IEEE 29th International Conference on Parallel and Distributed Systems (ICPADS) | 979-8-3503-3071-7/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICPADS60453.2023.00143

FedCD: A Hybrid Centralized-Decentralized Architecture for Efficient Federated Learning

Pengcheng Qu[a], *Jianchun Liu[a], Zhiyuan Wang[a], Qianpiao Ma[b], Jinyang Huang[c]
[a]School of Computer Science and Technology, University of Science and Technology of China, Hefei, China
[b]Purple Mountain Laboratories, Nanjing, China
[c]School of Computer and Information, Hefei University of Technology, Hefei, China
Email: {qupengcheng, cswangzy}@mail.ustc.edu.cn, jcliu17@ustc.edu.cn, maqianpiao@pmlabs.com.cn, hjy@hfut.edu.cn

***Abstract*—With billions of IoT devices producing vast data globally, privacy and efficiency challenges arise in AI applications. Federated learning (FL) has been widely adopted to train deep neural networks (DNNs) without privacy leakage. Existing centralized and decentralized FL architectures have limitations, including memory burden, huge bandwidth pressure and non-IID data issues. This paper introduces a novel framework, named FedCD, merging the benefits of both centralized and decentralized FL architectures. FedCD strategically distributes the model based on layer sizes and consensus distances (measuring the deviation between the local models and the global average models), effectively relieving network bandwidth pressures and accelerating training speed even under the non-IID setting. This method significantly mitigates resource constraints and improves model accuracy, offering a promising solution to the challenges in distributed machine learning. Extensive experiment results show the high effectiveness of FedCD. The total completion time of FedCD is reduced by 16.3%-53% and the average accuracy improvement is 1.85% compared to the existing FL systems.**

***Index Terms*—federated learning, edge computing, layer distribution**

I. Introduction

Billions of Internet of Things (IoT) devices globally generate substantial data, including photos and voice samples, propelling the advancement of artificial intelligence (AI) [1]. Nonetheless, the process of cloud computing carries the inherent risk of privacy breaches since the data gathered by the cloud may contain sensitive and confidential user information. Also, transferring all data to a remote cloud server can increase latency and degrade user experience [2]. Therefore, edge computing (EC) [3], [4] has emerged as a solution to locally store data and shift high computing power applications from cloud servers to network edges [5]. Furthermore, to alleviate data privacy leakage concerns, federated learning (FL) [6] is employed for distributed machine learning at the network edge across distributed datasets.

The most prevalent and widely used architecture in the existing FL mechanisms is the centralized FL architecture [6], [7], which involves local training at the network edge and model aggregation at the parameter server (PS). Initially, each worker executes stochastic gradient descent (SGD) [6] on its local dataset to minimize the loss function and then dispatches the updated model to the PS for global aggregation. Subsequently, the PS circulates the averaged model back to the workers for the following round of local training. However, the PS can become a bottleneck due to potential traffic congestion caused by numerous workers simultaneously communicating, leading to system breakdown if the PS is compromised.

Decentralized federated learning (DFL) [8], [9] serves as an alternative FL architecture. Here, each worker exchanges models with its neighbors and aggregates them for the subsequent local training. As there is no central PS, DFL eliminates the likelihood of traffic congestion at the PS and the PS failure risks. Furthermore, communication between workers is faster than between workers and the PS, significantly reducing communication time. Despite these advantages, two challenges exist in DFL: 1) *Limited Memory Size*. Workers must receive and store neighbor models, which may strain memory resources because the memory size of a worker is always limited. 2) *Non-IID Local Data*. The training data of a client is always determined by the environment and users' preferences. The data in each worker is not independent and identically distributed (non-IID) in practice and cannot represent the overall data distribution. For example, in the garage, some cameras take more pictures of people, and some take more pictures of vehicles. This challenge also exists in centralized FL. But in DFL, each worker only exchanges models with a limited number of neighbors, and the training speed and the test accuracy are affected more by non-IID local data [10], [11].

To address the memory strain challenge due to large model sizes, model parallelism [12]- [13] is suggested. This approach divides the model into sub-models and distributes them across various devices, alleviating device resource consumption. For instance, RePurpose [12] adjusts neuron positions to decrease intermediate data transmission between workers. However, it still increases network bandwidth strain due to frequent data transmission. The pdADMM [13] divides the model by layers, allowing each layer to update the model independently without communicating with other layers. However, this method is only suitable for relatively small models. Current deep neural networks (DNNs) have large parameter sizes that continue to grow. Transmitting these large models to the PS or neighbors will inevitably consume substantial bandwidth resources, presenting a significant challenge for

2690-5965/23/$31.00 ©2023 IEEE
DOI 10.1109/ICPADS60453.2023.00143

both centralized and decentralized architectures.

In this paper, we propose a novel FL framework, named FedCD, combining features of both centralized and decentralized architectures. FedCD is a framework that belongs to model parallelism. In FedCD, some layers adopt centralized architecture and some layers adopt decentralized architecture. We design a score for each layer according to the layer's location and size, and we distribute the layers to the PS and the neighbors according to the scores at the beginning stage. Then we use the intermediate data such as consensus distances [14] to form a new layer distribution method at the following stage. FedCD relieves network bandwidth pressure by transmitting sub-models in both centralized and decentralized architectures. However, under the decentralized setup, each worker exchanges models with only a limited number of other workers, reducing model performance when local data is non-IID [11]. To address this, FedCD incorporates a centralized architecture to aggregate sub-models into a global model at the PS, yielding good performance even with non-IID local data. Furthermore, only sub-models are stored in the workers' memories which can relieve the burden of the memories. Consequently, our proposed FedCD can enhance model training even under non-IID settings and alleviate resource constraint pressures. The main contributions of our work can be summarized as follows:

- We propose a novel FL framework, called FedCD, which facilitates the distribution of sub-models to the PS and the workers' neighbors for efficient aggregation. We provide theoretical evidence affirming the convergence guarantee of model training with this framework.
- We design a novel algorithm that strategically determines the distribution of layers to the PS and the neighbors. This decision is initially based on the sizes and positions of the layers, and it is subsequently adjusted according to consensus distances, enabling accelerated convergence speed.
- Experiment results on classical models and real-world datasets show the effectiveness of the proposed method. FedCD can accelerate the training speed by 16.3% - 53% and reduce communication traffic compared to the existing FL systems.

II. Preliminaries and Problem Formulation

A. Federated Learning

Traditional centralized FL consists of N workers and a parameter server (PS). Each worker i has a loss function based on the local dataset D_i and the size of the dataset is $|D_i|$. The loss function can be defined as:

$$F_i(w_i^t) = \frac{1}{|D_i|} \sum_{\xi \in D_i} f_i(w_i^t, \xi) \tag{1}$$

where w_i^t is the model parameter of worker i at round t and ξ is a batch of the local dataset D_i. $f_i(w_i^t, \xi)$ is the local loss function over ξ. To minimize the loss function $F_i(w_i^t)$, worker i uses stochastic gradient descent (SGD) [6] to update the local model, which can be formulated as:

$$w_i^{t+\frac{1}{2}} = w_i^t - \eta \nabla F_i(w_i^t) \tag{2}$$

where η is the learning rate, $w_i^{t+\frac{1}{2}}$ is the local model of worker i which finishes the local training after round t, and $\nabla F_i(w_i^t)$ is the gradient of the loss function.

After local training, each worker pushes the local model to the PS for global aggregation. This process can be formulated as:

$$F(w^t) = \frac{1}{N} \sum_{i=1}^{N} F_i(w^t) \tag{3}$$

where $w^t = \frac{1}{N} \sum_{i=1}^{N} w_i^t$ and $F(w^t)$ is the global loss function. Then the PS sends the updated global model to the workers for the next training.

In decentralized FL, the workers are connected in a network topology. This topology can be modeled as an undirected graph $G = (V, E)$, where $V = \{1, 2, ..., N\}$ and $E \in V \times V$. We use a matrix $A = \{A_{i,j} \in \{0, 1\}, 1 < i, j < N\}$ to represent the graph, where $A_{i,j} = 1$ represents there is a link between worker i and worker j. Otherwise, $A_{i,j} = 0$. Worker i first updates the local model using SGD, and then sends the local model to neighbors. Worker i receives the models from their neighbors and aggregates them as:

$$w_i^{t+1} = w_i^t + \sum_{j \in N_i^t} u_{i,j}^t (w_j^{t+\frac{1}{2}} - w_i^{t+\frac{1}{2}}) \tag{4}$$

where N_i^t is the neighbor set of worker i at round t. $u_{i,j}$ is the mixing weight for aggregating the model of worker j. After aggregating the received models, the worker i adopts the aggregated model for the next local training.

B. Overview of FedCD

In this section, we will introduce our proposed framework FedCD, which includes an edge server and some clients. In FedCD, the PS periodically receives the status information (*e.g.*, the consensus distance and the accuracy improvement). After that, the PS generates the layer distribution policy and sends it to the clients at regular intervals. Once the clients receive the policy, the clients will send some layers to the PS and send the rest layers to the neighbors according to the policy. Then the clients aggregate the received models. The clients combine the models received from the PS and the models averaged by themselves. The clients will use the combined model for the next local training. This entire process operates for numerous communication rounds until the model achieves convergence. The process of FedCD is illustrated in Fig. 1.

C. Convergence Analysis

In this section, we propose the convergence analysis of FedCD and make four widely used assumptions as follows:

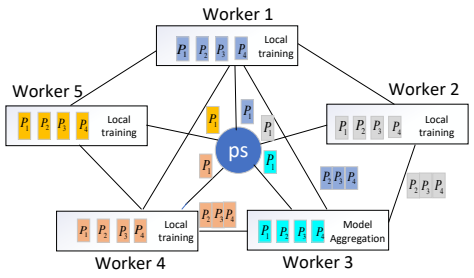


Fig. 1: The training process of FedCD (Worker 3 performs model aggregation).

- **Assumption1: Lipschitzian gradient.** The loss function F_i is with L Lipschitzian gradients, *i.e.*,

$$||\nabla F_i(w_1) - \nabla F_i(w_2)||^2 \leq L^2 \|w_1 - w_2\|^2, \forall w_1, w_2, i \tag{5}$$

- **Assumption2: Network connectivity.** The network topology G is a connected topology.
- **Assumption3: Bounded gradient variance.** The variance of stochastic gradients is bounded, *i.e.*,

$$E_{\xi \in D_i}|| \bigtriangledown F_i(w) - \bigtriangledown f_i(w;\xi)||^2 \leq \sigma^2, \forall i, w \tag{6}$$

$$E_{i \in V}|| \bigtriangledown F_i(w) - \bigtriangledown F(w)||^2 \leq \varsigma^2, \forall w \tag{7}$$

- **Assumption4: Bounded model variance.** The variance between the local model and global model is bounded by ϵ^2 , *i.e.*,

$$E_{i \in V}||w^t - w_i^t||^2 \leq \epsilon^2, \forall t, i. \tag{8}$$

where w^t is the average of the combined model of all the workers, and w_i^t is the combined model of worker i.

To express the relationship between the average of the combined models and the global model when one worker finishes local training, we adopt an upper bound α^2, *i.e.*,

$$||w^{t+1} - w^{t+\frac{1}{2}}||^2 \leq \alpha^2 \tag{9}$$

where $w^{t+\frac{1}{2}}$ represents the global model when one worker finishes the local training using SGD after round t, and w^{t+1} represents the average of the combined models in round $t+1$.

We conduct an experiment to test the change in the value of $||w^{t+1} - w^{t+\frac{1}{2}}||$. The results show that $||w^{t+1} - w^{t+\frac{1}{2}}||$ oscillates around 0.8 in IID settings and around 1 in non-IID settings with the increase of communication rounds. So we can use a small bound to limit $||w^{t+1} - w^{t+\frac{1}{2}}||^2$.

We also define the upper bound β^2 as:

$$||\nabla F(w^{t+\frac{1}{2}})||^2 - ||\nabla F(w^t)||^2 \leq \beta^2 \tag{10}$$

We replace the parameter in [15] (*e.g.*, $\hat{M}_k$ by ϵ^2). The detailed proof is as follows: (w^0 is the initial model, w^* is the optimal model which minimizes F)

$$\begin{aligned}
&E[F(w^{t+\frac{1}{2}}) - F(w^*))] \\
&\leq E[F(w^t) - F(w^*)] - \frac{\eta M}{2N} E||\nabla F(w^t)||^2 + \lambda
\end{aligned} \tag{11}$$

where $\lambda \leq \frac{\eta M L^2 \epsilon^2}{2N} + \frac{\eta^2 L(\sigma^2 M + 6\varsigma^2 M^2)}{2N^2} + \frac{6\eta^2 L^3 M^2 \epsilon^2}{N^2}$, M is the size of the data used in each communication round by one worker. We further have:

$$\begin{aligned}
&E[F(w^{t+1}) - F(w^{t+\frac{1}{2}})] \\
&\leq E < \nabla F(w^{t+\frac{1}{2}}), w^{t+1} - w^{t+\frac{1}{2}} > + \frac{L}{2}||w^{t+1} - w^{t+\frac{1}{2}}||^2 \\
&= \frac{1}{2}||\nabla F(w^{t+\frac{1}{2}}) + w^{t+1} - w^{t+\frac{1}{2}}||^2 - \frac{1}{2}||\nabla F(w^{t+\frac{1}{2}})||^2 \\
&- \frac{1}{2}||w^{t+1} - w^{t+\frac{1}{2}}||^2 + \frac{L}{2}||w^{t+1} - w^{t+\frac{1}{2}}||^2 \\
&\leq \frac{1}{2}\left\{||\nabla F(w^{t+\frac{1}{2}})||^2 + ||w^{t+1} - w^{t+\frac{1}{2}}||^2\right\} \\
&+ \frac{L}{2}||w^{t+1} - w^{t+\frac{1}{2}}||^2 \\
&\leq \frac{1}{2}E||\nabla F(w^t)||^2 + \frac{1}{2}\beta^2 + \frac{L+1}{2}\alpha^2
\end{aligned} \tag{12}$$

By adding Eq. (11) and Eq. (12), we obtain the convergence bound between two consecutive training rounds:

$$\begin{aligned}
&E[F(w^{t+1}) - F(w^*)] \\
&\leq E[F(w^t) - F(w^*)] - \frac{\eta M - N}{2N} E||\nabla F(w^t)||^2 \\
&+ \lambda + \frac{1}{2}\beta^2 + \frac{L+1}{2}\alpha^2 \\
&E[F(w^{t+1}) - F(w^t)] \\
&\leq -\frac{\eta M - N}{2N} E||\nabla F(w^t)||^2 + \lambda + \frac{1}{2}\beta^2 + \frac{L+1}{2}\alpha^2
\end{aligned} \tag{13}$$

We sum the results in Eq. (13) from t=0 to t=T-1 and obtain:

$$\begin{aligned}
&\sum_{t=0}^{T-1} E[F(w^{t+1}) - F(w^t)] = E[F(w^T) - F(w^0)] \\
&\leq -\frac{\eta M - N}{2N} \sum_{t=0}^{T-1} E||\nabla F(w^t)||^2 + T(\lambda + \frac{1}{2}\beta^2 + \frac{L+1}{2}\alpha^2) \\
&\frac{1}{T}\sum_{t=0}^{T-1} E||\nabla F(w^t)||^2 \\
&\leq \frac{2N(F(w^0) - F(w^*))}{T(\eta M - N)} + \frac{2N}{(\eta M - N)}(\lambda + \frac{1}{2}\beta^2 + \frac{L+1}{2}\alpha^2)
\end{aligned} \tag{14}$$

D. Problem Formulation

To train models among distributed workers by FL, it is inevitable to consume resources (*e.g.*, CPU cycles and network traffic). Formally, we define the computing resource consumption of worker k in one round as c_k. Thus, the computing resource consumption of T rounds is Tc_k. We accumulate N workers' computing consumption and the total consumption should not exceed its budget B_c. Each worker's transmission workload of centralized architecture is defined as W_a, where $W_a = \sum_{i=1}^{L}(1 - x_i)M(i)$. We define the total size of the transmitted data of one peer in a decentralized architecture as W_b, where $W_b = \sum_{i=1}^{L} 2x_i M(i)$. $M(i)$ is the data volume of the i-th layer model. x_i is a binary variable and it indicates whether layer i adopts centralized architecture or not (0 for adopting centralized architecture and 1 for adopting decentralized architecture). We define the total transmission budget of a round as B_b. Let B_1 and B_2 represent the inbound bandwidth between workers and the PS, and the outbound bandwidth between workers and the PS, respectively. We denote the bandwidth of worker j and worker

m as B_{jm}. r_{jm} is a binary variable and it indicates whether there is a link between worker j and worker m. We define the capacity budget for PS node as C_a and the capacity budget for worker j as C_j. We formulate the problem as follows:

$$\min \ \lambda H + (1-\lambda) f(w^t)$$

$$\text{s.t.} \begin{cases} \sum_{k=1}^{N} T c_k \leq B_c \\ \sum_{k=1}^{N} 2W_a + \sum_{j=1}^{N} \sum_{m=1}^{N} W_b r_{jm} \leq B_b \\ N W_a \leq C_a \\ W_b \sum_{m=1}^{N} r_{jm} \leq C_j \\ x_i = \{0,1\} \\ r_{jm} = \{0,1\} \end{cases}$$

Let $H = \max\{W_a/B_1 + W_a/B_2, \max\{W_b/B_{jm}\}\}$.

Algorithm 1 LDLS ($c_1 < c_2$)

1: Initialize $\mu_1, \mu_2, L_1, L_2, sum_1 = 0, sum_2 = 0$;
2: Initialize the set of unassigned layers $Q_1 = \{1, 2, ..., L\}$;
3: Sort all layers in non-decreasing order by $\mu_1(l)$;
4: Select the first element c_1;
5: Sort all layers in non-increasing order by $\mu_2(l)$;
6: Select the first element c_2;
7: $Q_1 \leftarrow Q_1 - \{c_1\}$; $Q_1 \leftarrow Q_1 - \{c_2\}$;
8: $L_1.insert(c_1)$; $L_2.insert(c_2)$;
9: $sum_1 += \mu_1(c_1)$; $sum_2 += \mu_2(c_2)$;
10: **while** $Q_1 \neq \phi$ **do**
11: **if** $sum_1 < sum_2$ **then**
12: search the element l_1 with minimum score in Q_1;
13: $L_1.insert(l_1)$;
14: $Q_1 \leftarrow Q_1 - \{l_1\}$;
15: $sum_1 += \mu_1(l_1)$;
16: **else**
17: search the element l_2 with maximum score in Q_1;
18: $L_2.insert(l_2)$;
19: $Q_1 \leftarrow Q_1 - \{l_2\}$;
20: $sum_2 += \mu_2(l_2)$;
21: **end if**
22: **end while**

Our objective is to minimize the maximum communication time and maximize the training speed of FedCD. If λ is set as a large number, we pursue the minimum communication time of a round. If λ is set as a small number, we pursue the quick training speed of a round. The first inequality indicates that the computing workload of T rounds for N workers is less than a budget. The second inequality indicates the total transmission cost per round is less than a budget. The third inequality indicates the PS node's capacity is larger than the total transmission volume between the PS node and workers. The fourth inequality indicates each worker's capacity is larger than the total sizes of the collected models of each worker.

III. Algorithm Design

A. Layer Distribution based on the Layer's Location and Size

We design two algorithms to decide which layer is sent to the PS and which layer is sent to the neighbors. Let's first introduce the layer distribution method which is based on the layers' locations and sizes (LDLS) in FedCD, and it is described in Algorithm 1. In LDLS, worker i first initializes two vectors μ_1 and μ_2, which represent the maximum time when each layer is sent to the PS and neighbors, respectively. FedCD uses sum_1 and sum_2 to express the total time when the layers are sent to the PS or the neighbors (Line 1). FedCD also initializes the unassigned layers set Q_1 which contains all the layers (Line 2). Because the bandwidth in the network fluctuates, the architecture uses the average value to represent the bandwidth of the network. Let's assume the inbound bandwidth between the workers and the PS is B_1, and the outbound bandwidth between the workers and the PS is B_2. Worker i computes the time when it transmits the layer l's parameter from the worker to the PS and downloads it from the PS to the worker, *i.e.*, $\mu_1(l) = M(l)/B_1 + M(l)/B_2$. $M(l)$ is the size of layer l. Let's define the minimum bandwidth in the network as B_{min}, the maximum time when each worker transmits the layer l's parameter using the decentralized method is $\mu_2(l) = M(l)/B_{min}$. The layers sent to the PS are in the set L_1, and the layers sent to the neighbors are in the set L_2.

We choose the layer with the minimum element in μ_1 and the layer is named c_1 (Line 4), and choose the layer with the maximum element in μ_2 and the layer is named c_2 (Line 6). Then, c_1 and c_2 are removed from Q_1 (line 7). We insert c_1 and c_2 to L_1 and L_2 respectively (Line 8). Then we add $\mu_1(c_1)$ to sum_1 (Line 9) and add $\mu_2(c_2)$ to sum_2 (Line 9). d_1 is used to express the layer's distance from c_1. If the number of the layer is lower than c_1, d_1 will be lower than 0, and if the number of the layer is higher than c_1, d_1 will be higher than 0. d_2 is used to express the layer's distance from c_2. Then we compute the score of each layer: We denote $m = (\sigma_1 d_1 + \sigma_2 d_2) M(l)$:

$$score(l) = e^m \tag{15}$$

Algorithm 1 introduces the LDLS method when $c_1 < c_2$. If $c_1 < c_2$, the centralized method will choose the layer with the minimum score, and the decentralized method will choose the layer with the maximum score. (That is because the scores of the layers next to c_1 are small and the scores of the layers next to c_2 are very large). If $sum_1 < sum_2$, we first choose the element in Q_1 with the minimum score and the layer will be named l_1. Then we move the layer l_1 out of Q_1 and insert l_1 to L_1. And sum_1 will be added by $\mu_1(l_1)$ (Line 11-15). If $sum_1 > sum_2$, we will choose the element in Q_1 with the largest score and the layer will be named l_2. Then l_2 will be sent to the neighbors for aggregation. We move the layer l_2 out of Q_1 and insert l_2 to L_2 . And sum_2 will be added by $\mu_2(l_2)$ (Line 17-20). If $c_1 > c_2$, the centralized method will choose the layer with the maximum score, and the decentralized

method will choose the layer with the minimum score. The algorithm will end when all the layers are distributed.

B. Layer Distribution based on Consensus Distance

LDLS method cannot perform well when the number of workers in the network is large and the number of epochs in a communication round is small. So the layer distribution based on consensus distances [14] (LDC) is proposed. The consensus distance $D^t(l)$ represents the deviation between the local model and the average global model of layer l. If $D^t(l)$ is small, it represents that the local model is similar to the average global model and the workers don't need to send the layer l to the PS. On the contrary, if $D^t(l)$ is large, it means that the local model differs greatly from the average global model, so the workers need to send layer l to the PS. We define $D^t(l) = \frac{1}{N}\sum_{i=1}^{N} D_i^t(l)$. $D_i^t(l)$ is the consensus distance of worker i. If the layer uses centralized method, $D_i^{t+1}(l)$ is defined as $D_i^{t+1}(l) = ||\bar{w}^{t+1}(l) - w_i^{t+\frac{1}{2}}(l)||$,where $w_i^{t+\frac{1}{2}}$ is the model of worker i which finishes performing local training after round t and $\bar{w}^{t+1}(l) = \frac{1}{N}\sum_{i=1}^{N} w_i^{t+\frac{1}{2}}(l)$. If the layer uses decentralized method, $D_i^{t+1}(l)$ is defined as $D_i^{t+1}(l) = ||\bar{w}^{t+1}(l) - w_i^{t+1}(l)||$, where $w_i^{t+1}(l)$ is the model of worker i which finishes aggregation using the models received from neighbors after round t.

However, in DFL, the average model $\bar{w}^{t+1}(l)$ is not available in practice. So when the layer adopts decentralized method to update the model, we will use the following method to calculate the consensus distance:

$$
\begin{aligned}
&D_i^{t+1}(l)\\
&= ||\bar{w}^{t+1}(l) - w_i^{t+1}(l)||\\
&= \big|\big|\tfrac{1}{N}\textstyle\sum_{j=1}^{N} w_j^{t+\frac{1}{2}}(l)\\
&- (w_i^{t+\frac{1}{2}}(l) + \textstyle\sum_{j=1}^{N} u_{i,j}^t A_{i,j}(w_j^{t+\frac{1}{2}}(l) - w_i^{t+\frac{1}{2}}(l)))\big|\big|\\
&= \big|\big|\textstyle\sum_{j=1}^{N} \frac{w_j^{t+\frac{1}{2}}(l) - w_i^{t+\frac{1}{2}}(l)}{N} - u_{i,j}^t A_{i,j}(w_j^{t+\frac{1}{2}}(l) - w_i^{t+\frac{1}{2}}(l))\big|\big|
\end{aligned}
$$

We set $u_{i,j}^t = \frac{1}{N}$ for simplicity, and then $D_i^{t+1}(l)$ is the possible maximum value, thus it follows:

$$
\begin{aligned}
&D_i^{t+1}(l)\\
&= \big|\big|\textstyle\sum_{j=1}^{N} \frac{(1-A_{i,j})(w_j^{t+\frac{1}{2}}(l) - w_i^{t+\frac{1}{2}}(l))}{N}\big|\big|\\
&\le \tfrac{1}{N}\textstyle\sum_{j=1}^{N}(1-A_{i,j})D_{i,j}^{t+1}
\end{aligned}
$$

where $D_{i,j}^{t+1}(l) = ||w_i^{t+\frac{1}{2}}(l) - w_j^{t+\frac{1}{2}}(l)||$,

$$
\begin{aligned}
&D_{i,j(max)}^{t+1}(l)\\
&= ||w_i^{t+\frac{1}{2}}(l) - w_k^{t+\frac{1}{2}}(l) + w_k^{t+\frac{1}{2}}(l) - w_j^{t+\frac{1}{2}}(l)||\\
&\le ||w_i^{t+\frac{1}{2}}(l) - w_k^{t+\frac{1}{2}}(l)|| + ||w_k^{t+\frac{1}{2}}(l) - w_j^{t+\frac{1}{2}}(l)||\\
&= D_{i,k}^{t+1}(l) + D_{k,j}^{t+1}(l)\\
&D_{i,j(min)}^{t+1}(l)\\
&= ||w_i^{t+\frac{1}{2}}(l) - w_k^{t+\frac{1}{2}}(l) - (w_j^{t+\frac{1}{2}}(l) - w_k^{t+\frac{1}{2}}(l))||\\
&\ge ||||w_i^{t+\frac{1}{2}}(l) - w_k^{t+\frac{1}{2}}(l)|| - ||w_k^{t+\frac{1}{2}}(l) - w_j^{t+\frac{1}{2}}(l)||||\\
&= ||D_{i,k}^{t+1}(l) - D_{k,j}^{t+1}(l)||
\end{aligned}
$$

Thus, we can estimate $D_{i,j(max)}^t(l)$ as $\hat{D}_{i,j(max)}^t(l)$:

$$\hat{D}_{i,j(max)}^t(l) = \min_{k\in[N]-\{i,j\}}(D_{i,k}^t(l) + D_{k,j}^t(l)) \tag{16}$$

we also can estimate $D_{i,j(min)}^t(l)$ as $\hat{D}_{i,j(min)}^t(l)$:

$$\hat{D}_{i,j(min)}^t(l) = \max_{k\in[N]-\{i,j\}}(||D_{i,k}^t(l) - D_{k,j}^t(l)||) \tag{17}$$

If $D_{i,j(max)}^t(l)$ is used to calculate $D^t(l)$, $D^t(l)$ is recorded as $D_{max}^t(l)$. $D_{min}^t(l)$ is the same.

t_{ps} and t_{nb} represent the total time when the layers are sent to the PS and the neighbors. If $\frac{t_{ps}}{t_{nb}}$ is low, we need to arrange more layers to adopt centralized method. So p_l^t which represents the probability of transmitting layer l to the PS is inversely proportional to $\frac{t_{ps}}{t_{nb}}$.

Based on the above analysis, we determine the variable p_l^t by using the following equation (We perform LDC when the round t=nT, T is the interval of performing two adjacent LDC):

$$p_l^t = \frac{[\lambda\sum_{h=(n-1)T}^{nT} D^{h+1}(l) + (1-\lambda)(e^{|d-d_m|})]e^{\frac{1}{\triangle\xi}}}{e^{\sqrt{\frac{t_{ps}}{t_{nb}}}}} \tag{18}$$

If the layer uses decentralized FL, we will compute the $p_{l(max)}^t$ using $D_{max}^h(l)$ and $p_{l(min)}^t$ using $D_{min}^h(l)$ to decide whether the layer will use centralized FL. $\triangle\xi$ represents the average accuracy improvement of the models. The minimum $\triangle\xi$ is set as 20%. If the accuracy improvement is lower, we will increase the p_l^t to make more layers use centralized mechanism. So we add $e^{\frac{1}{\triangle\xi}}$ to the equation. We compute the average number of the layers using the decentralized method and denote it as d_m, and use d to represent the number of layer l. We use p_l^t to determine the probability of the layer l to use the centralized method. The layer with higher p_l^t will be preferentially chosen to use the centralized method. We use the exponential moving average to smooth the probability:

$$P_l = \varphi p_l^t + (1-\varphi)P_l \tag{19}$$

where φ ($0 \le \varphi \le 1$) is a hyperparameter that reflects the weight of the previous probability and the newly-computed probability. We use K to denote the number scale of P_l. For example, if $P_l = 5\times10^{-4}$, then $K = -4$. We use $V(l)$ to represent the communication rounds when layer l uses centralized FL. We use V_{max} to represent the total communication rounds. We use $H(l)$ to denote the communication rounds when layer l uses the decentralized method. Because we want the layers which perform less centralized FL to have a high probability of performing centralized FL, we add a penalty item for the probability P_l. If the layer l is in L_1, we use Eq. (20) to calculate the decision variable R_l. If the layer l is in L_2, we use Eq. (21) to calculate the decision variable R_l.

$$R_l = P_l + 10^K\sqrt{ln(t+1)}(1 + \frac{V_{max}}{V(l)+1}) \tag{20}$$

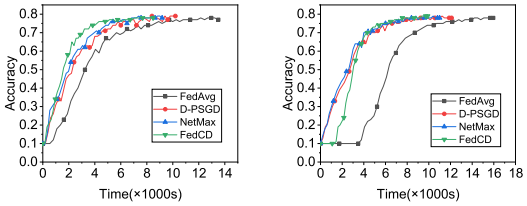
$$R_l = P_l + 10^K\frac{\sqrt{ln(t+1)}}{1 + V_{max} - H(l)} \tag{21}$$

Algorithm 2 LDC

Input: the probability of the layers R_l; T_1, T_2; S_1, S_2
Output: L_1 and L_2.

1: **while** round t=nT **do**
2: **if** $\sum_{l \in L_1} R_l > T_1$ **then**
3: $S_1 = |L_1|$;
4: **else**
5: $S_1 = max\{S_1 - 1, 1\}$;
6: **end if**
7: choose S_1 layers in L_1 with the largest R_l and these S_1 layers perform centralized FL (clear L_1 and insert these these S_1 layers to L_1).
8: **if** $\sum_{l \in L_2} R_{l(max)} > T_2$ **then**
9: $S_2 = max\{|L_2|/2, 1\}$;
10: **else**
11: $S_2 = 0$;
12: **end if**
13: choose S_2 layers in L_2 and these S_2 layers perform centralized FL (insert these these S_2 layers to L_1);
14: clear L_2 and add the unselected layers to L_2;
15: **end while**

We design a method to decide the layer distribution based on consensus distance (LDC) in Algorithm 2. When performing LDC for the first time, we first choose the minimum layer, and choose another L-S_1-S_2-1 layers which are next to the minimum layer with the smallest sizes to use the decentralized method. The average number of these layers is denoted as d_m. The rest layers adopt centralized method. When it is not the first time to perform LDC, we first sum up R_l of the layers in L_1. If the sum is larger than the threshold T_1, S_1 is set as $|L_1|$ (Line 2-3). S_1 is the number of the layers which continue to perform centralized FL. If the sum is lower than the threshold T_1. S_1 will minus 1 (Line 5). Then we choose S_1 layers in L_1 with the largest R_l and these S_1 layers perform centralized FL (Line 7). Because the estimated values of $D^t(l)$ of the layers in L_2 differ significantly from the actual value, we will not compare them with the $D^t(l)$ of the layers in L_1 together. If we use $p^t_{l(max)}$ to compute R_l, we denote R_l as $R_{l(max)}$, and if we use $p^t_{l(min)}$ to compute R_l, we denote R_l as $R_{l(min)}$. We sum up $R_{l(max)}$ of the layers in L_2 to compare it with the threshold T_2. If the sum is larger than T_2, S_2 is set to be $max\{|L_2|/2, 1\}$, otherwise $S_2 = 0$ (Line 8-11). And then we choose S_2 layers in L_2 to use the centralized method. If S_2 isn't 0, we sort the layers by $R_{l(max)}$ using descending order. If the difference between the value of the S_2-th and the (S_2+1)-th $R_{l(max)}$ is less than a threshold, we choose S_2-1 layers with the highest $R_{l(max)}$ and we choose one layer of the S_2-th and the (S_2+1)-th layers with higher $R_{l(min)}$. Otherwise, we choose S_2 layers with higher $R_{l(max)}$, and these layers perform the centralized FL (Line 13). After that, we clear L_2 and add the unselected layers to L_2 (Line 14). In the validation experiment, the LDC method can accelerate the training speed, but the final accuracy may be reduced. So after performing LDC for some rounds, the workers will use the LDLS method to achieve higher accuracy.



(a) node=5 (b) node=10

Fig. 2: AlexNet over CIFAR-10 under IID setting.

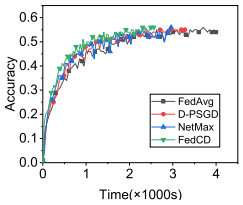


Fig. 3: VGG-9 over CIFAR-100.

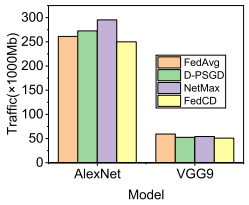


Fig. 4: Total traffic in the network

IV. Experiments

A. Experiment Platform

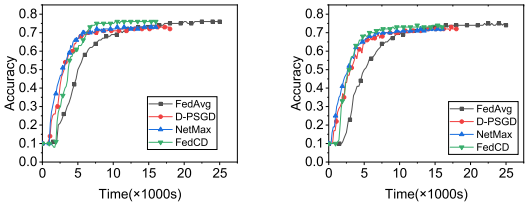
We perform experiments on an AMAX deep learning workstation equipped with an Intel(R) Core(TM) i9-10900X CPU, 4 NVIDIA GeForce RTX 2080Ti GPUs and 128 GB RAM. On the workstation, we implement 5-10 processes to simulate 5-10 workers and implement 1 process to simulate the parameter server. The execution of each worker's model training is based on the PyTorch framework [16]. The socket library of Python [16] is used to build up communication between workers and the parameter server.

B. Models and Datasets

The experiments are conducted on three well-known DNNs: VGG-9, VGG-16 [17] and AlexNet [18], which represent the middle-size model (VGG-9) and the large-size models (VGG-16 and AlexNet). The sizes of DNNs are 13.33MB,128.25MB and 88.78 MB respectively. AlexNet is trained over CIFAR-10, which includes 50,000 images for training and 10,000 for testing. The images in CIFAR-10 are 32×32×3 dimensional and are labeled in 10 classes. VGG9 and VGG16 are trained over CIFAR-100 dataset which consists of 100 classes. We employ the SGD [6] optimizer for AlexNet, VGG-16 and VGG-9 and the learning rates are initialized as 0.01. The models are trained with a batch size of 64.

C. Baselines and Metrics

Baselines: We choose three classical and efficient algorithms as baselines for performance comparison, which are summarized as follows:



(a) non-IID 50% (b) non-IID 70%

Fig. 5: AlexNet over CIFAR-10 under different non-IID levels.

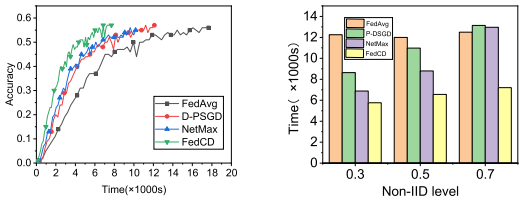


Fig. 6: VGG-16 non-IID 50%

Fig. 7: Time cost under different non-IID levels

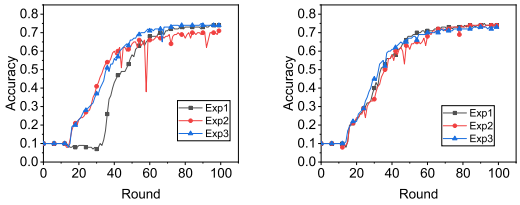
- **FedAvg** [6] is a famous algorithm in federated learning in which the workers send the entire model to the parameter server and download the models after aggregation at PS.
- **D-PSGD** [9] is a famous algorithm in DFL. Each worker sends the trained model to the neighbors and each worker aggregates the models locally.
- **NetMax** [19] is a communication-aware DFL technique over the heterogeneous network. It enables each worker to asynchronously pull models from one peer for aggregation. The peers with higher bandwidth are selected with higher probabilities.

Metrics: We adopt the following metrics to evaluate the performance of our proposed FedCD and baselines.

- **Test accuracy** is the amount of the right data predicted by the model divided by the amount of all the data. It is used to test whether the method can converge or not.
- **Completion time** is the time when each worker finishes local training and model aggregation. It is used to evaluate the training speed.
- **Network traffic** is the total size of the models transmitted through the network, which is adopted to quantify the communication cost.

D. Overall Performance

Training Performance: We use LDLS to test the performance of FedCD under IID settings. Fig. 2 shows the training performance of AlexNet over CIFAR-10 with 5-10 workers under IID settings. Fig. 3 shows the performance of VGG-9 over CIFAR-100. As we can see in Fig. 2-3, our proposed FedCD converges faster than FedAvg, D-PSGD, and NetMax. This is because FedCD sends the model to the PS and the neighbors simultaneously which can save time.



(a) The difference between using LDC and LDLS. (b) Performance on different section selection.

Fig. 8: Experiments using LDC

Fig. 4 shows the total traffic in the network when AlexNet reaches an accuracy of 70% and VGG9 reaches an accuracy of 50%. We can find out that FedCD consumes the least traffic. FedCD needs 22 rounds for AlexNet and 30 rounds for VGG9 to reach the target accuracy which uses the least rounds compared to the three baselines. In Fig. 2 we can find out that when the scale of workers becomes larger, our proposed FedCD still maintains its advantages over the baselines.

Performance on non-IID Data: We use LDLS to test the performance of FedCD under non-IID settings. LDLS cannot perform well when the number of nodes is large and the epochs of a communication round are small under non-IID settings. So in our experiments, we use 8 workers and the number of epochs in a communication round is set as 20. We distribute the dataset into different Non-IID levels χ (*e.g.*, 30%, 50%, and 70%) as suggested in [20]. As we can see in Fig. 5-6, we train AlexNet over CIFAR-10 and train VGG-16 over CIFAR-100 under non-IID settings. FedCD outperforms baselines at all non-IID levels. For example, FedCD achieves an accuracy of 76.67%, when we train AlexNet over CIFAR-10 with χ=50%, which is higher than that of FedAvg (76.42%), D-PSGD (73.37%) and NetMax (73.79%). This is because FedCD combines the advantage of centralized and decentralized methods which is more robust in non-IID settings. Secondly, as χ increases, the time requirement of each system increases when achieving the same accuracy. But Fig. 7 shows that FedCD is more robust without a significant increase in the completion time when we train AlexNet over CIFAR-10 and set the accuracy of 72%, compared with the baselines. For FedCD, the time consumption is 5,860.8s (χ=30%), 6,837.6s (χ=50%) and 7,326s (χ=70%), while they are 7,000.4s, 9,116.8s, 13,349.6s for NetMax. This is because the per-epoch completion time of FedCD is much shorter than other systems, and FedCD iterates more epochs and achieves a better performance under a given time budget.

Performance of LDC: We conduct some experiments to test the effect of the LDC algorithm using AlexNet over CIFAR-10 with 10 workers. In Fig. 8(a), the first experiment uses the LDLS algorithm to train the model, and the second experiment combines the LDC and LDLS algorithms. For the first 15 rounds, we use the LDLS algorithm and for the rest rounds, we use the LDC method and don't change the layer distribution after 15 rounds. The third experiment

uses LDLS for the first 15 rounds. For the rounds 15-30, we use LDC, and for the rounds after 30, we use the LDLS method again. We can find out in Fig. 8(a) that in the first experiment, the improvement of accuracy is very slow at the beginning, experiments 2 and 3 converge faster than experiment 1. However, the final accuracy of experiment 2 is lower than that of experiments 1 and 3. Experiment 3 performs well on both training speed and final test accuracy.

We continue to conduct three experiments to test the performance of LDC. In Fig. 8(b), all the three experiments use LDLS at the beginning. In the intermediate stage, all the three experiments use LDC. Experiment 1 changes the layer distribution at rounds 15 and 30. Experiment 2 changes the layer distribution at rounds 15, 30, and 45. Experiment 3 changes the layer distribution at rounds 10, 20, and 30. We continue to use the LDLS method after round 45 in experiment 1, round 60 in experiment 2 and round 40 in experiment 3. We can observe in Fig. 8(b) that experiment 3 converges faster than its two counterparts at the beginning, however, its training speed falls behind after 50 rounds and the final test accuracy is the lowest in the three experiments. Experiment 1 converges faster than its two counterparts after round 50 and it reaches the highest test accuracy in the end. Thus the number of training sections of LDC (when to change the layer distribution) is essential to achieve quick training speed and high test accuracy.

V. Conclusion

In this paper, we propose a FL framework named FedCD, which is a combination of centralized and decentralized architectures. The method addresses the challenges of the memory burden, huge bandwidth pressure, and non-IID local data. We have further proposed two algorithms to decide which layer is sent to the PS and which layer is sent to the neighbors. The experiment results show that FedCD significantly outperforms baselines.

Acknowledgement

This article is supported in part by the National Science Foundation of China (NSFC) under Grants U1709217, 61936015, 62132019 and 62302145; in part by Anhui Province Science Foundation for Youths (Grant No. 2308085QF230);in part by Jiangsu Province Science Foundation for Youths under Grant BK20230275; in part by Xiaomi Young Talents Program.

References

[1] G. Zhu, D. Liu, Y. Du, C. You, J. Zhang, and K. Huang, "Toward an intelligent edge: Wireless communication meets machine learning," IEEE communications magazine, vol. 58, no. 1, pp. 19–25, 2020.

[2] A. Aral, M. Erol-Kantarci, and I. Brandic, "Staleness control for edge data analytics," Proceedings of the ACM on Measurement and Analysis of Computing Systems, vol. 4, no. 2, pp. 1–24, 2020.

[3] M. Satyanarayanan, "The emergence of edge computing," Computer, vol. 50, no. 1, pp. 30–39, 2017.

[4] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," IEEE internet of things journal, vol. 3, no. 5, pp. 637–646, 2016.

[5] X. Wei, Q. Li, Y. Liu, H. Yu, T. Chen, and Q. Yang, "Multi-agent visualization for explaining federated learning." in IJCAI, 2019, pp. 6572–6574

[6] 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.

[7] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, "Scaling distributed machine learning with the parameter server," in 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14), 2014, pp. 583–598.

[8] S. Savazzi, M. Nicoli, and V. Rampa, "Federated learning with cooperating devices: A consensus approach for massive iot networks," IEEE Internet of Things Journal, vol. 7, no. 5, pp. 4641–4654, 2020.

[9] 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," arXiv preprint arXiv:1705.09056, 2017

[10] M. Waqas, Y. Niu, M. Ahmed, Y. Li, D. Jin, and Z. Han, " Mobility aware fog computing in dynamic environments: Understandings and implementation," IEEE Access, vol. 7, pp.38 867–38 879, 2018

[11] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, "Federated learning with non-iid data," arXiv preprint arXiv:1806.00582, 2018.

[12] Afshin Abdi, Saeed Rashidi, Faramarz Fekri, Tushar Krishna "Restructuring, Pruning, and Adjustment of Deep Models for Parallel Distributed Inference" arXiv preprint arXiv:2008.08289, 2020.

[13] Wang J , Chai Z , Cheng Y , et al. "Toward Model Parallelism for Deep Neural Network based on Gradient-free ADMM Framework."20th IEEE International Conference on Data Mining, ICDM ,2020, pp. 591–600

[14] Liao Y, Xu Y, Xu H, et al. Adaptive Configuration for Heterogeneous Participants in Decentralized Federated Learning[J]. arXiv preprint arXiv:2212.02136, 2022

[15] X. Lian, W. Zhang, C. Zhang, and J. Liu, "Asynchronous decentralized parallel stochastic gradient descent," in International Conference on Machine Learning. PMLR, 2018, pp. 3043–3052

[16] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga et al., "Pytorch: An imperative style, high-performance deep learning library," Advances in neural information processing systems, vol. 32, 2019.

[17] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," arXiv preprint arXiv:1409.1556, 2014

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," Advances in neural information processing systems, vol. 25, pp. 1097–1105, 2012 .

[19] Zhou P, Lin Q, Loghin D, et al. Communication-efficient decentralized machine learning over heterogeneous networks[C]//2021 IEEE 37th International Conference on Data Engineering (ICDE). IEEE, 2021: 384-395.

[20] H. Wang, Z. Kaplan, D. Niu, and B. Li, "Optimizing federated learning on non-iid data with reinforcement learning," in IEEE INFOCOM 2020-IEEE Conference on Computer Communications. IEEE, 2020, pp. 1698–1707.