MPLS: Stacking Diverse Layers Into One Model for Decentralized Federated Learning

Yang Xu[1,2], Zhiwei Yao[1,2], Hongli Xu[1,2](✉), Yunming Liao[1,2], and Zuan Xie[1,2]

[1] School of Computer Science and Technology, University of Science and Technology of China, Hefei, China
{xuyangcs,xuhongli}@ustc.edu.cn,
{zhiweiyao,ymliao98,xz1314}@mail.ustc.edu.cn

[2] Suzhou Institute for Advanced Research, University of Science and Technology of China, Suzhou, China

Abstract. Traditional Federated Learning (FL) enables collaborative training of deep neural networks (DNNs) across massive edge devices while preserving data privacy. However, its reliance on a centralized parameter server (PS) introduces communication bottlenecks and security risks. To address these issues, Decentralized Federated Learning (DFL) has emerged, which adopts peer-to-peer (P2P) communication to eliminate the PS. Despite its advantages, DFL faces critical challenges: limited bandwidth resources, dynamic network conditions, and heterogeneous data properties. To conquer these challenges, we design and implement a *communication-efficient DFL framework with* ***p****eer and* ***l****ayer* ***s****election (MPLS)*, which has the following advantages. 1) Different from exchanging an entire model between two workers in previous works, each worker just collects multiple sub-models (*i.e.*, some critical layers) from the chosen peers and stacks them into one model for aggregation. 2) MPLS adopts asynchronous training among workers without any coordinator and enables each worker to develop the peer and layer selection strategy adaptively via the proposed list scheduling algorithm. We implement MPLS on a physical platform, and extensive experiments on real-world DNNs and datasets demonstrate that MPLS achieves 2.1–4.2× speedup compared to the baselines.

Keywords: Decentralized Federated Learning · Edge Network · Device Heterogeneity

1 Introduction

Recently, Federated Learning (FL) has emerged as a promising paradigm for distributed model training among workers (*e.g.*, end devices or edge nodes). Traditional FL relies on a central parameter server (PS) to periodically aggregate and distribute models from/to the local workers. However, this centralized approach can lead to network congestion, making the PS a bottleneck, and pose security risks if the PS is compromised [1]. To conquer this, Decentralized Federated Learning (DFL) [1] has been proposed, allowing workers to collaboratively train models by exchanging model parameters directly. By eliminating the PS, DFL avoids single point of failure and reduces network congestion, thereby accelerating the training process.

Table 1. Comparison of different DFL systems with different aggregation schemes.

Aggregation Schemes	DFL Systems	Heterogeneity Awareness	Resource Cost		Convergence Performance
			Time	Traffic	
Synchronous	[9]	No	Long	Massive	Good
	[10]	No	Long	Average	Good
	[11,12]	Yes	Long	Average	Good
Asynchronous	[2]	No	Average	Massive	Good
	[3,4]	Yes	Average	Massive	Good
	[8]	No	Average	Low	Poor
	[7]	Yes	Low	Low	Poor
	MPLS	Yes	Low	Low	Good

DFL systems can operate in synchronous or asynchronous manners. Existing studies mainly focus on improving the training efficiency of the synchronous DFL. Synchronous DFL requires all workers to synchronize their updates, which creates a synchronization barrier. This barrier forces faster workers to wait for slower ones, leading to decreased training efficiency and poor fault tolerance, as a single failed node can disrupt the entire system. To conquer these issues, asynchronous DFL [2–4] has been introduced, allowing workers to train DNNs at their own pace without waiting for others. Since the synchronization barrier among workers can be alleviated, asynchronous DFL is more effective than the synchronous one for handling system heterogeneity [5].

Existing asynchronous DFL systems can be categorized into two types. The first type involves workers collecting *entire* models from multiple or all peers for aggregation [2–4]. While this type can achieve effective convergence, it faces two major challenges: 1) *Huge Bandwidth Pressure*. Transmitting large DNN models consumes substantial bandwidth resources, leading to network congestion. 2) *Long Training Time*. In edge network, workers typically communicate with peers over wireless links with limited bandwidth [6]. Moreover, the communication delay in each epoch always depends on the slowest link, as model aggregation only proceeds once all models from selected peers are received. This results in prolonged training time and slow convergence rate [7]. The second type allows each worker to pull an *entire* model from only one peer, either randomly [8] or based on the highest-speed link [7]. While this reduces communication overhead, it introduces two key limitations: 1) *Inefficient Bandwidth Utilization.* Since each worker aggregates the model only from one peer, the bandwidth resource of other peers remains underutilized. 2) *Poor Convergence Performance.* Since the local data held by different workers are not independent and identically distributed (Non-IID), the data distribution at an individual worker may not represent the overall population distribution. Such data heterogeneity will inevitably deteriorate training convergence [2]. In addition, many asynchronous DFL systems [2,7] rely on a coordinator to determine system parameters, which can become a bottleneck and reduce fault tolerance. A detailed comparison is shown in Table 1.

To this end, we propose MPLS, a communication-efficient decentralized federated learning system. Unlike existing approaches [7,8] that exchange entire models between two workers, MPLS enables each worker to pull multiple sub-models (*i.e.*, some critical layers) from selected peers in parallel for aggregation. In this way, the bandwidth resource of the entire system can be effectively utilized, and the bandwidth pressure on each worker is significantly alleviated. Moreover, MPLS supports fully asynchronous and independent training and aggregation without any coordinator, ensuring high scalability and fault tolerance. However, implementing MPLS is non-trivial due to system and data heterogeneity and network dynamics. Intuitively, peers with higher bandwidth should be selected with larger probability, but selecting too many layers from them can cause unbalanced communication delay among peers and slow down the training speed. Besides, frequent layer collection from fixed peers may hinder model generalization and convergence. Moreover, the development of the aggregation strategy should also vary over training epochs to adapt to network dynamics for improving training performance. The main contributions of this paper are as follows:

- We propose MPLS, a decentralized federated learning system that enables workers to collect multiple sub-models from selected peers for aggregation.
- To accelerate training in heterogeneous and dynamic edge network, MPLS employs a dynamic peer and layer selection strategy using a novel list scheduling algorithm.
- Extensive experiments on a physical platform demonstrate that MPLS can provide 2.1–4.2$\times$ speedup over the state-of-the-art DFL systems.

2 Background and Motivation

2.1 Decentralized Federated Learning

DNN Model. A specific DNN model is composed of a sequence of multiple layers with different types (*e.g.*, convolutional layers and fully-connected layers). Let L be the number of layers of a DNN model $w = \{w(1), \cdots, w(L)\}$ and M_l be the size of $w(l), \forall l \in \{1, 2, ..., L\}$. Thus, the size of whole model can be denoted as $\sum_{l=1}^{L} M_l$.

Network Model. DFL uses a set of N workers to collaboratively train models with its local data by peer-to-peer communication, *i.e.*, each worker exchanges model parameters with its connected peers in a network topology. We denote the network topology as a symmetric adjacency matrix $A = \{A_{i,j} \in \{0, 1\}, 1 \leq i, j \leq N\}$, where $A_{i,j} = 1$ denotes a connection link between worker i and j.

Training Procedure. DFL generally consists of two iterative steps. *1) Local training.* Each worker i maintains a local model w_i and a dataset D_i of size $|D_i|$, with the local loss function defined as $f_i(w_i) = \frac{1}{|D_i|} \sum_{\xi \in D_i} F_i(w_i; \xi)$, where ξ is a batch of data samples in D_i and $F_i(w_i; \xi)$ is the loss over ξ. To minimize $f_i(w_i)$, worker i updates its model using stochastic gradient descent (SGD), *i.e.*, $\overline{w}_i = w_i - \eta \nabla f_i(w_i)$, where $\overline{w}_i$ is the local model of worker i after local training, η is the learning rate and $\nabla f_i(w_i)$ is the gradient of the loss with respect to the current model. *2) Model aggregation.* Worker i collects models from connected (or selected) peers and aggregates them to generate a new model for the next epoch. The model aggregation step may vary significantly across different systems. For example, in some systems, each worker i communicates with all its peers and updates its local model asynchronously [4], *i.e.*, $w_i = (\sum_{j=1}^{N} A_{i,j} w_j + \overline{w}_i)/(1 + \sum_{j=1}^{N} A_{i,j})$. In contrast, in [7], each worker i just collects a model from one connected worker j, and sets $w_i = \frac{1}{2}(\overline{w}_i + w_j)$. These two steps are performed iteratively until the predefined termination conditions (*e.g.*, time constraints or target accuracy) are satisfied. Notably, in an asynchronous DFL system [7], local training and model aggregation can overlap, as workers begins to pull models from peers during local training.

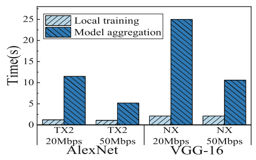


Fig. 1. Per-epoch time consumption.

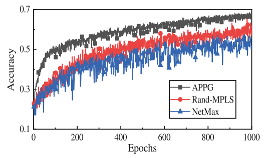


Fig. 2. Training process of DFL systems.

2.2 Why is Communication a Bottleneck?

In DFL, communication costs during model aggregation are a critical bottleneck, especially in edge networks. The rapid growth in DNN size has significantly increased the network burden and delays. While computing performance of edge devices has improved 10–100× compared to CPUs due to hardware accelerators (*e.g.*, NVIDIA TX2/NX), network bandwidth improvements (typically 5-25Mbps for WANs) have lagged. To further analyze the overhead of local training (computing) and model aggregation (communication), we measure delays on NVIDIA TX2 and NX devices (see Table 2) under various network conditions while training AlexNet and VGG-16 over STL-10. Figure 1 shows that communication time dominates training time, often exceeding local training delay by up to 10×. Although the local training time can be increased by performing more iterations of SGD in each epoch, it will deteriorate the training performance and trap the model of each worker in local optima. Therefore, a proper aggregation strategy will result in a significant reduction of training time.

2.3 Observation and Motivation

For asynchronous DFL, different model aggregation strategies greatly impact communication costs and training efficiency. We evaluate three systems: 1) *APPG* [4] lets each worker collect models from all connected peers for aggregation, and achieves the best performance (see Fig. 2). 2) *NetMax* [7] enables each worker to pull models from the peers with the highest bandwidth. 3) *Rand-MPLS* lets each worker pull each layer of a DNN from a randomly selected peer. Compared to APPG, each worker in NetMax and Rand-MPLS only collects one entire model (from only one or many peers) for aggregation. By pulling multiple sub-models to rebuild a complete DNN for aggregation, each worker in Rand-MPLS can learn model information from different peers, and the generalization of the combined model can be improved. NetMax, however, exhibits unstable training performance with significant fluctuations in its training curve. These results verify the potential of MPLS for performance improvement. Unlike Rand-MPLS's random strategy, we will further incorporate dynamic and heterogeneous network conditions and data heterogeneity to develop an efficient aggregation strategy.

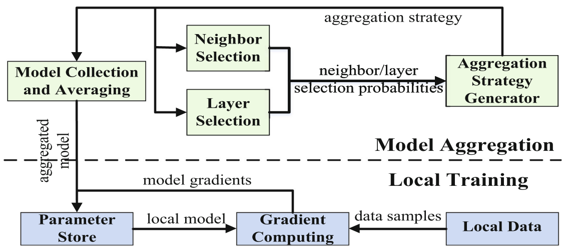


Fig. 3. Key modules and workflow on each worker in MPLS.

3 MPLS Design

3.1 System Overview

Figure 3 presents the key modules of MPLS on each worker with respect to two training processes, *i.e.*, local training and model aggregation. Specifically, three modules (*i.e.*, *gradient computing*, *local data* and *parameter store*) cooperate to perform the local training. Besides, the model aggregation process, which is the focus of MPLS design, consists of four modules, *e.g.*, *aggregation strategy generator*, *peer selection*, *etc.*. The detailed workflow among these modules is as follows: ① In the local training step, the *gradient computing* module pulls the stored local model and a batch of data samples from *parameter store* and *local data*, respectively, and returns the computed model gradients to *parameter store* for model updating. ② In the model aggregation step, the *model collection and averaging* module collects layers from the chosen peers based on the developed aggregation strategy, rebuilds a combined DNN, and averages it with the local model. The aggregated model will be returned to *parameter store*. ③ To accommodate the network dynamics in edge network during model aggregation and improve the model convergence performance, the *peer selection* determines the selection probability for each peer, and the *layer selection* derives the layer selection probabilities guided by historical aggregation strategies. ④ By jointly considering the peer and layer selection probability distribution, the *aggregation strategy generator* is able to determine which layers of a DNN should be collected from each peer for model aggregation.

3.2 Design of Key Components

We now focus on the model aggregation step of an arbitrary worker i and describe its individual modules in Fig. 3. Let K be the total number of local iterations of worker i.

The total number of training epochs T in MPLS is the sum of local iterations among all workers, as all workers perform training asynchronously.

Model Collection and Averaging. This module is responsible for collecting layers from the peers for model aggregation. Take Fig. 4 as an example, where there are 5 workers and a DNN consists of four layers (*i.e.*, $\{P_1, P_2, P_3, P_4\}$). Figure 4 plots the aggregation steps of two consecutive local iterations of worker #3. Firstly, worker #3 pulls $\{P_1, P_4\}$, $\{P_2\}$ and $\{P_3\}$ from the parameter stores of workers #1, #2 and #4 respectively at local iteration k, and averages them with the local model after local training $\overline{w}_3^k$. The aggregated model w_3^{k+1} is used for local training of iteration $k+1$. Secondly, to adapt to network dynamics and enhance performance, different aggregation strategies may be adopted across iterations. For example, if the link between workers #3 and #4 is broken in iteration $k+1$, worker #3 will only pull sub-models from workers #1 and #2. Besides, worker #3 pulls $\{P_2\}$ and $\{P_1, P_3, P_4\}$ from workers #1 and #2 respectively, which is different from that of local iteration k. This is because aggregating the same layer from different peers across epochs is conducive to the model generalization.

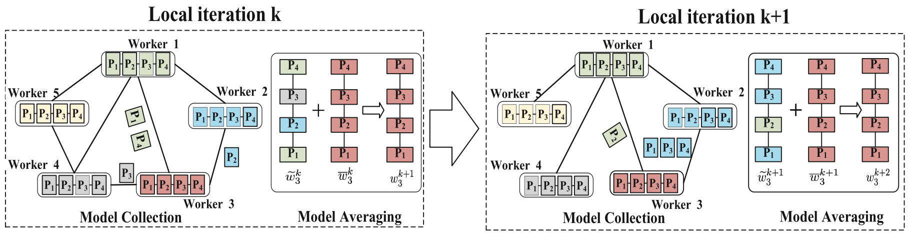


Fig. 4. The procedure of model collection and averaging of worker #3 in MPLS.

Generally, assume that worker i has S peers and performs model collection and averaging at local iteration k, and we adopt $y_s^k(l)$ to represent the layer selection strategy of peer s, *e.g.*, $y_s^k(l) = 1$ means that worker i pulls layer l from peer s at iteration k. The approach of model averaging for worker i as follows

$$w_i^{k+1}(l) = \frac{\sum_{s=1}^{S} y_s^k(l) w_s^k(l) + w_i^k(l)}{\sum_{s=1}^{S} y_s^k(l) + 1} \tag{1}$$

The new model w_i^{k+1} for further local training is a combination of each updated layer, *i.e.*, $w_i^{k+1} = \{w_i^{k+1}(1), ..., w_i^{k+1}(L)\}$. This aggregated model will be sent to parameter store for local training at iteration $k+1$. Though each worker only collects sub-models from the chosen peers for training, MPLS can still guarantee training convergence. With connected network topology, our theoretical analysis shows that the convergence bound is mainly related to the number of workers N and the total training epochs T, *i.e.*, $\frac{1}{T}\sum_{t=0}^{T-1} \mathbb{E}\|\nabla f(w^t)\|^2 = \mathcal{O}(\frac{N}{T})$, where $w^t = \frac{1}{N}\sum_{i=1}^{N} w_i^t$. We present the detailed proof in https://github.com/Jollydos/MPLS_Euro_Par.

Peer Selection. Since training data on each worker always depends on its local environment and usage pattern, the distribution of the local datasets often varies across workers [13], negatively impacting model convergence. To address this, we focus on the model aggregation process, enabling each worker to incorporate model information trained on datasets with diverse data distributions. Specifically, assume the local data distribution of worker i is $P_i = \{p_i^1, p_i^2, ..., p_i^C\}$, where $C = |P_i|$ is the total number of classes and p_i^c represents the proportion of the local data samples in D_i with label $c \in \{1, 2, ..., C\}$. Thus, the distribution divergence over classes between the local datasets of device i and its peer s is denoted as

$$DD_s = \sum_{c=1}^{C} ||p_i^c - p_s^c||. \tag{2}$$

Intuitively, by aggregating the local model from the peer s with higher DD_s more frequently, the local model of worker i can learn more model information within different data distributions, and then be a good representative of the global model [13].

However, since the bandwidth of workers in edge network may vary with time, such factor should also be considered during peer selection. Specifically, we denote the link speed between worker i and its peer s at iteration k as B_s^k. Jointly considering the heterogeneous network conditions among peers and Non-IID data, we design a two-step peer selection rule: Firstly, we standardize the network status and data distribution divergence of all peers. For each peer s at local iteration k, we have

$$\overline{B}_s^k = \frac{B_s^k}{\sum_{s'=1}^{S} B_{s'}^k}, \qquad \overline{DD}_s = \frac{DD_s}{\sum_{s'=1}^{S} DD_{s'}} \tag{3}$$

Secondly, we introduce two variables τ_1 and τ_2 to control the weight of each factor, where $\tau_1 + \tau_2 = 1$. As the result, we update the communication probability p_s^k for worker i to aggregate layers from peer s at local iteration k as

$$p_s^k = \frac{\tau_1 \overline{B}_s^k + \tau_2 \overline{DD}_s}{\sum_{s'=1}^{S} (\tau_1 \overline{B}_{s'}^k + \tau_2 \overline{DD}_{s'})} \tag{4}$$

A larger τ_1 denotes that device prefers to aggregate layers from the peer with better network conditions, while a larger τ_2 favors peers with greater distribution divergence. The impact of τ_1 and τ_2 on the training performance will be explored in the experiment.

Layer Selection. Each worker i iteratively performs the training process to make its local model w_i^t gradually approach to the global optima w^*. We use $g_i^t(l)$ to represent the distance between $w_i^t(l)$ and $w^*(l)$ for layer l, *i.e.*, $g_i^t(l) = \|w_i^t(l) - w^*(l)\|^2$. Intuitively, each worker i can aggregate each layer l from the peer s with a minimum $g_s^t(l)$ to accelerate its training process. However, since w^* is unavailable until model converges, we cannot obtain $g_s^t(l)$ to guide layer selection. Instead, we estimate the training efficiency of each peer s based on its gradient variation and define $g_s^{t,t'}(l)$ as its training performance from epochs t' to t on layer l, *i.e.*,

$$g_s^{t',t}(l) = \|w_s^t(l) - w_s^{t'}(l)\|^2 \tag{5}$$

The peer with stronger training capacity can perform local training and model aggregation more frequently and achieve a higher convergence rate, resulting in a greater $g_s^{t,t'}(l)$. Based on this, we further denote $q_s^k(l)$ as the probability that worker i pulls layer l from peer s at iteration k (epoch t). Assume worker i performs two consecutive model aggregations at epoch t' and t, respectively. For each layer l, we have $\sum_{s=1}^{S} q_s^k(l) = 1$, where $q_s^k(l)$ is derived by

$$q_s^k(l) = \frac{g_s^{t',t}(l)}{\sum_{s'=1}^{S} g_{s'}^{t',t}(l)} \tag{6}$$

The better the training performance of peer s on layer l from epoch t' to t, the higher the probability worker i will pull this layer from it. Notably, worker i only receives $g_s^{t',t}(l)$ values from each peer s for layer selection, incurring minimal communication cost.

Aggregation Strategy Generator. In MPLS, the peer selection and layer selection in MPLS are not independent, and the aggregation strategy generator will jointly optimize them to develop an aggregation strategy for accelerating the training procedure in each iteration k. For a given strategy, the communication cost λ_s^k for collecting several layers from the peer s can be expressed as

$$\lambda_s^k = \sum_{l=1}^{L} \frac{y_s^k(l) M_l}{B_s^k} \tag{7}$$

As a result, for worker i, the total delay λ^k for model aggregation in the local iteration k is determined by the last received layers, *i.e.*, $\lambda^k = \max_s \lambda_s^k$. We target to minimize the total communication delay (*i.e.*, $\min \lambda^k$) at any local iteration k while ensuring the convergence performance. Specifically, we let each worker i pull layer l from a peer s with a probability $p_s^k q_s^k(l)$ (*i.e.*, the product of peer and layer selection probabilities) greater than a lower bound $\theta(l)$ at local iteration k, *i.e.*,$p_s^k q_s^k(l) > \theta(l)$. Intuitively, to ensure that each layer can be aggregated by worker i, we set $\theta(l) = \frac{1}{S}\sum_{s=1}^{S} p_s^k q_s^k(l)$.

By solving this problem, the generator will output the aggregation strategy to the model collection and averaging module. When only considering layer selection, this problem simplifies to the generalized assignment problem [14], *i.e.*, assigning each layer l to a selected peer s for minimizing the delay (an NP-hard problem). Due to its complexity, we propose a list scheduling [15] based algorithm in the next section.

4 Aggregation Strategy Development

We formally present the proposed list scheduling based algorithm for model aggregation strategy development in MPLS, which consists of four phases.

Initialization. In each local iteration k, worker i initializes two $S \times L$ matrixes μ_1 and μ_2. Using the link speed measured by network monitor, the worker can compute the time $\mu_1(s,l)$ for pulling each layer l from each peer s, *i.e.*, $\mu_1(s,l) = M_l/B_s^k$. Besides, $\mu_2(s,l)$ in matrix μ_2 is the probability for aggregating layer l from peer s, *i.e.*, $p_s^k q_s^k(l)$.

Problem Transformation. The development of aggregation strategy can be modeled by an assignment function $\pi : \{1, ..., L\} \rightarrow \{1, ..., S\}$. $\pi(l) = s$ means the layer l will be pulled from peer s. The communication delay of the assignment π is denoted as

$$\lambda(\pi) = \max_{s \in \{1,...,S\}} \sum_{l|\pi(l)=s} \mu_1(s, l) \tag{8}$$

To meet the constraint $p_s^k q_s^k(l) > \theta(l)$, we modify the matrix μ_1 using μ_2. Specifically, for each layer l and peer s, if $\mu_2(s, l) < \theta(l)$, we adjust μ_1 by setting $\mu_1(s, l) = \infty$, which indicates that layer l will not be assigned to peer s.

List Scheduling. To generate an assignment π by list scheduling, we first define the start time ϕ and the finish time ψ for each layer l. Specifically, if worker i pulls layer l from peer s at time $\tilde{t}$, we have $\phi(l) \le \tilde{t} \le \psi(l)$, where $\psi(l) = \phi(l) + \mu_1(s, l)$. Then, the total communication delay of model collection is equal to the maximum finish time among all layers, *i.e.*, $\lambda(\pi) = \max_l \psi(l)$. The matrix μ_1 associates S possible transmission time with each layer l, and we define the shortest one as $\varphi(l) = \min_s \mu_1(s, l)$. Furthermore, we obtain the efficiency of pulling layer l from peer s by the shortest transmission time $\varphi(l)$, denoted as $E(s, l)$, *i.e.*,

$$E(s, l) = \frac{\varphi(l)}{\mu_1(s, l)} \tag{9}$$

where $E(s, l) \in [0, 1]$. The larger $E(s, l)$ represents a shorter transmission delay. For each peer s, we sort all layers in a list by descending order of $E(s, l)$. We denote the total efficiency of each peer s on the entire DNN as $rank(s) = \sum_{l=1}^{L} E(s, l)$. Then, we sort all peers in descending order. Each worker likely pulls layers from the peer s with a higher $rank(s)$ for reducing the time consumption.

Algorithm 1. Aggregation Strategy Development in MPLS

```
1: Initialize $\mu_1$, $\mu_2$ and $sum(s) = 0, \forall s$;
2: Initialize the set of unassigned layers $Q_1 = \{1, 2, ..., L\}$;
3: Initialize the set of available peers $Q_2 = \{1, 2, ..., S\}$;
4: Sort all layers in descending order by $E(s, l), \forall l$ for each peer $s$;
5: Sort all neighbors in descending order by $rank(s), \forall s$;
6: while $Q_1 \neq \emptyset$ do
7:     Select a neighbor $k \in Q_2$ such that $sum(s) < sum(s')$ or $rank(s) \ge rank(s')$ when $sum(s) = sum(s'), \forall s'$;
8:     Select a layer $l \in Q_1$ with highest $E(s, l)$ from $s$'s list;
9:     if $E(s, l) \le 1/\sqrt{S}$ then
10:        Update $Q_2 \leftarrow Q_2 - \{s\}$;
11:    else
12:        Update $Q_1 \leftarrow Q_1 - \{l\}$, and set $\pi(l) = s$;
13:        Update $sum(s) \leftarrow sum(s) + \mu_1(s, l)$;
14:        Update $rank(s) \leftarrow rank(s) - E(s, l)$;
15: Compute $MAX = \max_s sum(s)$;
16: for $s \in \{1, 2, ..., S\}$ do
17:     while peer $s$'s list is not empty do
18:         Select a layer $l$ with highest $E(s, l)$ from $s$'s list;
19:         if $\pi(l) \neq s$ and $sum(s) + \mu_1(s, l) \le MAX$ then
20:             Aggregate layer $l$ from peer $s$;
21:             $sum(s) \leftarrow sum(s) + \mu_1(s, l)$;
22:         Remove layer $l$ from $s$'s list;
23: Return The final aggregation strategy, i.e., assignment $\pi$;
```

Assignment Generation. The lines 1–5 show the operations in the former three phases. Based on this, each worker is able to develop the preliminary model aggregation strategy by using the proposed list scheduling approach (lines 6–14). Specifically, we adopt $sum(s)$=$\sum_{l|\pi(l)=s} \mu_1(s,l)$ to record the current transmission delay of each peer s during the process of layer assignment, and create two sets (Q_1 and Q_2) to record the unassigned layers and available peers, respectively. Based on these parameters, worker i firstly chooses the peer s with the minimum $sum(s)$. Worker i then searches for the layer l that has not been assigned (*i.e.*, $l \in Q_1$) and is with the highest $E(s,l)$ in the list of the selected peer s (lines 7–8), and determines whether to pull this layer from it (lines 9–14). If the efficiency $E(s,l)$ of this selected layer is smaller than a predefined lower bound (*i.e.*, $1/\sqrt{S}$ [16]), worker i will remove the peer s from the set Q_2 (lines 9–10) to prevent to pull any other layers from this peer (*i.e.*, leading to long transmission time). Otherwise, worker i will pull layer l from peer k and the parameters, *e.g.*, $sum(s)$, will be updated accordingly (lines 12–14).

Once the preliminary assignment π is obtained (lines 6–14), we compute the maximum delay (*i.e.*, MAX) incurred by any single peer under this assignment (line 15). Then, we search for the layers that should be further aggregated from each peer (lines 16–22). Specifically, for each peer s, we sequentially judge whether each layer l can be aggregated from it or not. If layer l is not assigned to peer s in current assignment π and aggregating this layer will not increase the maximum delay (line 19), worker i will pull this layer l from peer s and update $sum(s)$ (lines 20–21).

Table 2. Different computing modes of TX2/NX devices.

	Mode	Denver2	ARMA57	ARMv8.2	GPU	Numbers	Location
TX2	1	2.0 GHz×2	2.0 GHz×4	–	1.3 GHz	1–5	A
	2	1.4 GHz×2	1.4 GHz×4	–	1.12 GHz	6–10	B
	3	0.0 GHz	1.2 GHz×4	–	0.85 GHz	11–15	C
NX	1	–	–	1.4 GHz×6	1.1 GHz	16–20	A
	2	–	–	1.2 GHz×4	0.8 GHz	21–25	B
	3	–	–	1.5 GHz×2	0.8 GHz	26–30	C

We analyze the computational complexity of Algorithm 1. When we adopt the quick sort, the complexity of lines 4–5 is $O(SL\log L)$. Since each iteration of the loop (lines 6–14) will remove one element from either Q_1 or Q_2, the worst time complexity of this procedure is $O((S+L)\log S)$, where $O(\log S)$ is the time for selecting the peer s by binary search [17]. Similarly, we remove one layer from each peer's list in each iteration in lines 16–22, the time complexity is $O(SL)$. Thus, the total complexity is $O(SL\log L)$. Considering the system performance, this algorithm only incurs negligible cost on each worker in MPLS while improving the training performance.

5 Performance Evaluation

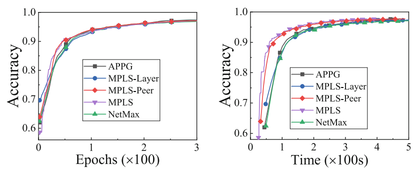
5.1 Experimental Setup

Experimental Environment. We evaluate the efficiency of MPLS on a physical platform using PyTorch. The platform includes 15 NVIDIA Jetson TX2 and 15 NVIDIA Jetson Xavier NX devices, labeled from 1 to 30, configured to represent a heterogeneous edge network: 1) *Computing heterogeneity.* Both TX2 and NX devices are set to 3 different computing modes, with 5 devices per mode (see Table 2). Specifically, the computing capacity of TX2/NX decreases in order from mode 1 to 3. 2) *Communication heterogeneity.* All the devices are interconnected via a 5 GHz WiFi router, distributed across three rooms (10 devices each). The WiFi router is located in room A.

Models and Datasets. We evaluate three well-known DNNs, *i.e.*, LeNet-5, NIN, ResNet-18, over three real datasets, *i.e.*, MNIST, CIFAR-10, CIFAR-100. These cover a wide range of small to large models and datasets for edge network. Models are trained with a batch size of 64 and a learning rate of 0.01 [7]. Datasets are randomly and uniformly distributed across devices.

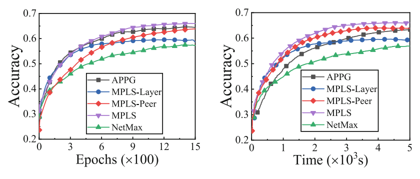
Baselines and Metrics. We adopt four baselines, *i.e.*, *NetMax* [7], *APPG* [4], *MPLS-Peer* and *MPLS-Layer*. Here, *MPLS-Peer* and *MPLS-Layer* are variants of MPLS with only peer or layer selection capabilities. The performance is evaluated using the metrics: 1) *Classification accuracy* is used to validate whether an FL algorithm can efficiently guarantee convergence. 2) *Completion time* is used to assess training speed. We adopt the average results of 3 independent experiments to avoid accidents.

Network Topology. For a fair comparison, we evaluate three common network topologies, *i.e.*, ring, randomly-generated graph and fully-connected graph. Specifically, 1) In the ring topology, workers are connected sequentially in the order $1 \rightarrow 2 \rightarrow \cdots \rightarrow 30 \rightarrow 1$. 2) In a randomly-generated graph, each pair of workers is connected with a probability of 50%. 3) In the fully-connected graph, each worker is directly connected with all other workers. By default, we adopt the randomly-generated graph.



(a) Accuracy vs Epochs. (b) Accuracy vs Time.

Fig. 5. Training performance for LeNet-5 over MNIST.



(a) Accuracy vs Epochs. (b) Accuracy vs Time.

Fig. 6. Training performance for NIN over CIFAR-10.

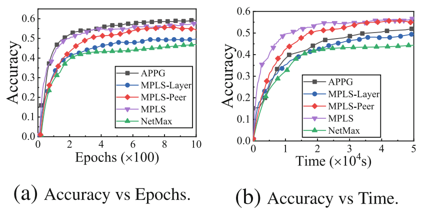


Fig. 7. Training performance for ResNet-18 over CIFAR-100.

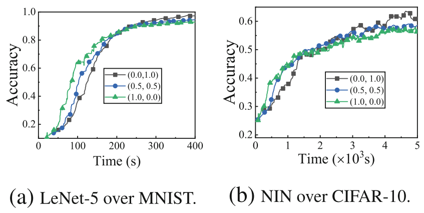


Fig. 8. Accuracy vs. Time for LeNet-5 and NIN with different values of τ_1 and τ_2.

5.2 Overall Performance

Convergence Performance. Figures 5, 6 and 7 show the training performance of LeNet-5 over MNIST, NIN over CIFAR-10, and ResNet-18 over CIFAR-100, respectively. For the smallest model (*i.e.*, LeNet-5), Fig. 5a indicates that all systems performed similarly. However, for deeper models, performance gaps between different systems occur in Figs. 6a and 7a. We note that MPLS achieves a higher convergence accuracy, compared with MPLS-Peer, MPLS-Layer and NetMax. That is because MPLS ensures each worker learns more information from its chosen peers than the other three systems in each local iteration without incurring extra time. Specifically, the final accuracy of APPG, NetMax, MPLS-Peer, MPLS and MPLS-Layer is 64.42%, 58.64%, 63.87%, 66.01%, and 57.22% for NIN over CIFAR-10 after 1,500 epochs, respectively. For ResNet-18, the accuracy is 59.12%, 49.71%, 54.27%, 57.43%, and 46.91%, correspondingly. Similarly, for the larger ResNet-18, APPG obtains better convergence performance than MPLS given the same number of epochs due to its strategy of collecting all peers' models for aggregation, at the expense of more resources (*e.g.*, time and traffic) consumption.

Resource Consumption. Figs. 5, 6 and 7 show the comparison of accuracy versus training time. We observe that MPLS-Peer and MPLS converge faster and achieve a higher accuracy given the same time budget. For NIN, MPLS achieves 2.1×, 3.4× and 4.2× speedup compared with APPG, NetMax and MPLS-Layer. Since all systems operate under the same computing environment, MPLS improves training efficiency by significantly reducing communication delays. With the efficient model aggregation strategy, MPLS and MPLS-Peer can maximize bandwidth utilization across all workers. As a result, our systems outperform NetMax, which mainly utilizes high-speed links.

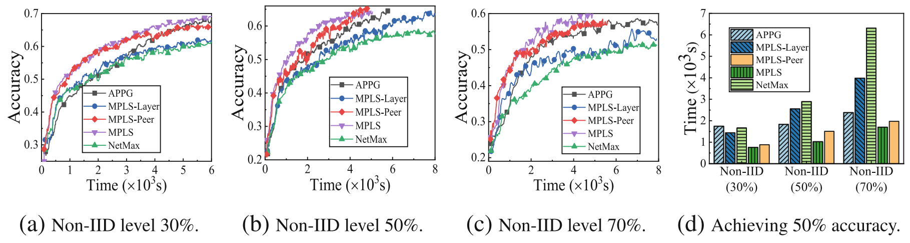


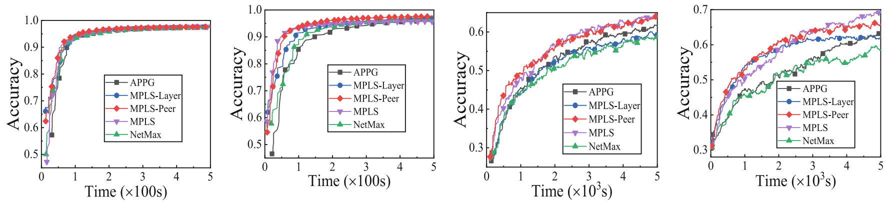
Fig. 9. Accuracy vs. Time for NIN over CIFAR-10 under different data distribution.

5.3 Hyper-Parameter Evaluation

To evaluate MPLS' sensitivity to the hyperparameters, we adopt different values of (τ_1, τ_2), including (1.0, 0.0), (0.5, 0.5) and (0.0, 1.0). The results for LeNet-5 over MNIST and NIN over CIFAR-10 with Non-IID level 70% are shown in Fig. 8. As the value of τ_1 increases (τ_2 decreases), we observe that models converge faster initially but with a little sacrifice of the final accuracy. A larger τ_1 means that each worker will collect layers from the peers with high link speed and thus guarantees a faster convergence at the early stage. However, due to Non-IID data, the worker may miss diverse class information, leading to slightly lower accuracy. On the contrary, when a larger τ_2 is adopted, each worker more frequently aggregates layers from peers with different data distributions. However, it inevitably increases the time required for model aggregation, resulting in a slower training speed. Achieving an optimal balance between time consumption and convergence performance by tuning τ_1 and τ_2 remains to be studied.

5.4 Effect of Data Distribution

We study the effect of Non-IID data on training efficiency by varying Non-IID levels χ (*e.g.*, 30%, 50%, and 70%) as in [13]. Specifically, for each worker, χ indicates the proportion of local data samples sharing the same label while the remaining $1-\chi$ belong to other labels. The results for NIN over CIFAR-10 are shown in Fig. 9. We make two observations as follows. Firstly, we observe that MPLS outperform baselines across all Non-IID levels. For instance, at $\chi = 30\%$, MPLS achieves 68.89% accuracy, surpassing APPG (68.37%), NetMax (61.68%), MPLS-Layer (61.21%) and MPLS-Peer (66.42%). This highlights the effectiveness of combining peer and layer selection in dealing with Non-IID issues. Secondly, as χ increases, all systems require more time to achieve the same accuracy. However, Fig. 9d shows that our systems achieve greater robustness with minimal time increase compared with NetMax, MPLS-Peer and MPLS-Layer. For example, MPLS completes training in 758.4 s ($\chi = 30\%$), 1025.9 s ($\chi = 50\%$) and 1700.2 s ($\chi = 70\%$), while NetMax takes 1437.6 s, 2551.2 s, and 3982.8 s, respectively. This efficiency stems from MPLS's shorter per-epoch completion time, enabling more iterations and better performance with the same time budget.



(a) LeNet-5 (ring graph). (b) LeNet-5 (fully-connected graph). (c) NIN (ring graph). (d) NIN (fully-connected graph).

Fig. 10. Accuracy vs. Time for LeNet-5 over MNIST and NIN over CIFAR-10 with different network topologies.

5.5 Effect of Network Topology

We further investigate the effect of network topology on the training efficiency. The results are shown in Fig. 10. Firstly, for the ring graph, in which each node only has two peers, our systems achieve slightly better performance compared with baselines. Since each worker can only access limited information from its two peers, the performance divergence among different systems will not be significantly widened. Secondly, for the fully-connected graph, training accuracy improves for most systems, as each worker can aggregate models from any other workers. For example, MPLS's accuracy for NIN over CIFAR-10 increases from 63.91% to 65.94% at 5,000 s. In addition, we observe that communication costs rise in systems such as APPG and Rand-MPLS due to increased peer interactions. However, systems like MPLS, Adv-MPLS, and NetMax, which optimize for link speed when performing model aggregation, converge faster on the fully-connected graph compared to the ring graph. As a result, the performance gap between systems widens significantly in the fully-connected graph. For example, for LeNet-5 over MNIST with the fully-connected graph, MPLS achieve 90% accuracy in 57.96 s, achieving a speedup of 2.5×, 1.6× and 2.1× over APPG (147.48 s), NetMax (91.6 s), and Rand-MPLS (121.36 s), respectively.

6 Conclusion

In this paper, we have designed MPLS, a communication-efficient and fully decentralized federated learning system to accelerate the DNN training over the heterogeneous and dynamic edge network. We further introduce adaptive algorithms so that each worker can develop the model collection and aggregation strategy independently and adaptively to minimize the completion time while ensuring convergence performance. Experimental results demonstrate that MPLS significantly outperforms baselines.

Acknowledgments. The corresponding author is Hongli Xu. This article is supported by the National Science Foundation of China (NSFC) under Grants 62472401, 62132019, and 624B2136.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Zhang, H., Bosch, J., Olsson, H.H.: Federated learning systems: architecture alternatives. In: 2020 27th Asia-Pacific Software Engineering Conference (APSEC), pp. 385–394. IEEE (2020)
2. Luo, Q., He, J., Zhuo, Y., Qian, X.: Prague: high-performance heterogeneity-aware asynchronous decentralized training. In: Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 401–416 (2020)
3. Zhao, L., Song, W.-Z., Ye, X., Yujie, G.: Asynchronous broadcast-based decentralized learning in sensor networks. Int. J. Parallel Emergent Distrib. Syst. **33**(6), 589–607 (2018)

4. Zhang, J., You, K.: Asynchronous decentralized optimization in directed networks. arXiv preprint arXiv:1901.08215 (2019)
5. Zhang, H., Hsieh, C.J., Akella, V.: Hogwild++: a new mechanism for decentralized asynchronous stochastic gradient descent. In: 2016 IEEE 16th International Conference on Data Mining (ICDM), pp. 629–638. IEEE (2016)
6. Hu, C., Bao, W., Wang, D., Liu, F.: Dynamic adaptive dnn surgery for inference acceleration on the edge. In: IEEE INFOCOM 2019-IEEE Conference on Computer Communications, pp. 1423–1431. IEEE (2019)
7. Zhou, P., et al.: Communication-efficient decentralized machine learning over heterogeneous networks. In: 2021 IEEE 37th International Conference on Data Engineering (ICDE), pp. 384–395. IEEE (2021)
8. Lian, X., Zhang, W., Zhang, C., Liu, J.: Asynchronous decentralized parallel stochastic gradient descent. In: International Conference on Machine Learning, pp. 3043–3052. PMLR (2018)
9. Lian, X., Zhang, C., Zhang, H., Hsieh, C.J., Zhang, W., Liu, J.: Can decentralized algorithms outperform centralized algorithms? A case study for decentralized parallel stochastic gradient descent. arXiv preprint arXiv:1705.09056 (2017)
10. Koloskova, A., Stich, S., Jaggi, M.: Decentralized stochastic optimization and gossip algorithms with compressed communication. In: International Conference on Machine Learning, pp. 3478–3487. PMLR (2019)
11. Jiang, J., Liang, H., Chenghao, H., Liu, J., Wang, Z.: Bacombo–bandwidth-aware decentralized federated learning. Electronics **9**(3), 440 (2020)
12. Hu, C., Jiang, J., Wang, Z.: Decentralized federated learning: a segmented gossip approach. arXiv preprint arXiv:1908.07782 (2019)
13. Wang, H., Kaplan, Z., Niu, D., Li, B.: Optimizing federated learning on non-iid data with reinforcement learning. In: IEEE INFOCOM 2020-IEEE Conference on Computer Communications, pp. 1698–1707. IEEE (2020)
14. Cattrysse, D.G., Van Wassenhove, L.N.: A survey of algorithms for the generalized assignment problem. Eur. J. Oper. Res. **60**(3), 260–272 (1992)
15. Arabnejad, H., Barbosa, J.G.: List scheduling algorithm for heterogeneous systems by an optimistic cost table. IEEE Trans. Parallel Distrib. Syst. **25**(3), 682–694 (2013)
16. Davis, E., Jaffe, J.M.: Algorithms for scheduling tasks on unrelated processors. J. ACM (JACM) **28**(4), 721–736 (1981)
17. Bentley, J.L.: Multidimensional binary search trees used for associative searching. Commun. ACM **18**(9), 509–517 (1975)