Intelligent and Converged Networks **Volume x, Number x, xxxxxxx 20xx**
ISSN xxxx-xxxx **DOI: 10.26599/ICN.20xx.90200xx**

Topology Design and Graph Embedding for Decentralized Federated Learning

Yubin Duan, Xiuqi Li, and Jie Wu *

Abstract: Federated learning has been widely employed in many applications to protect the data privacy of participating clients. Although the data set is decentralized among training devices in federated learning, the model parameters are usually stored in a centralized manner. Centralized federated learning is easy to implement; however, a centralized scheme causes a communication bottleneck at the central server, which may significantly slow down the training process. To improve training efficiency, we investigate the decentralized federated learning scheme. The decentralized scheme has become feasible with the rapid development of device-to-device communication techniques under 5G. Nevertheless, the convergence rate of learning models in the decentralized scheme depends on the network topology design. We propose optimizing the topology design to improve training efficiency for decentralized federated learning, which is a non-trivial problem, especially when considering data heterogeneity. In this paper, we first demonstrate the advantage of hypercube topology and present a hypercube graph construction method to reduce data heterogeneity by carefully selecting neighbors of each training device — a process that resembles classic graph embedding. In addition, we propose a heuristic method for generating torus graphs. Moreover, we have explored the communication patterns in hypercube topology and propose a sequential synchronization scheme to reduce communication cost during training. A batch synchronization scheme is presented to fine-tune the communication pattern for hypercube topology. Experiments on real-world data sets show that our proposed graph construction methods can accelerate the training process, and our sequential synchronization scheme can significantly reduce the overall communication traffic during training.

Key words: Data heterogeneity, decentralized federated learning, graph embedding, network topology.

1 Introduction

Federated learning is a promising approach for performing distributed machine learning while protecting the data privacy of each participating client. Machine learning, especially deep learning, has been widely deployed in many application scenarios, such as natural language processing and computer vision. In traditional machine learning schemes, the training data is usually shared among all training devices. However, centralized data storage has caused privacy issues. For example, patient information stored in medical institutions should not be shared with a third party. To protect data privacy, feder-

• Yubin Duan, Xiuqi Li, and Jie Wu are with the Department of Computer and Information Sciences, Temple University, Philadelphia, PA 19122, USA. E-mail: {yubin.duan, xli, jiewu}@temple.edu.
∗ Corresponding author: Jie Wu, jiewu@temple.edu.
• This research was supported in part by NSF grants SaTC 2310298, CNS 2214940, CPS 2128378, CNS 2107014, CNS 2150152, CNS 1824440, CNS 1828363, and CNS 1757533.
Manuscript received: year-month-day; accepted: year-month-day

ated learning is proposed [1]. In federated learning, each training device has its own local data set that would not be exchanged with other devices.

Although the training data set is decentralized among devices, many federated learning schemes use a centralized server to maintain the parameters of machine learning models like Fig. 1(a). In particular, each training device in federated learning has its local model parameters. In every training iteration, participating training devices would update their local models based on their local data sets. Then, the local updates are aggregated by a central server and the global model stored in the central server would be updated accordingly. Centralized federated learning is easy to implement and the performance of the global model is relatively easy to evaluate. However, the centralized scheme causes a communication bottleneck at the central server. Especially when the network bandwidth is low, the network traffic may cause congestion at the server side and significantly slow down the training process [2]. To mitigate the communication bottleneck, we explore decentralized federated learning in this paper.

In decentralized federated learning shown in Fig. 1(b), training devices directly communicate with each other to synchronize local model updates. With the development of wireless communication techniques, device-to-device (D2D) communication has become feasible in real-world applications using 5G [3]. Utilizing the D2D communication channels, training devices can directly exchange model updates with each other without going through a centralized server, which can amortize the communication cost among all training devices and avoid the communication bottleneck. Nevertheless, decentralized federated learning has its unique challenges, namely, each training device only synchronizes with its neighbor nodes in each training iteration, which may affect the convergence property of learning algorithms. [2], [4], and [5] have analyzed the performance of optimization algorithms for decentralized training. [4] and [5] have shown that the decentralized scheme can achieve the same convergence rate while avoiding the communication traffic jam. [5] also shows that the degree of the network plays an important role in the convergence rate. It is worthwhile to investigate the topology design problem for decentralized federated learning.

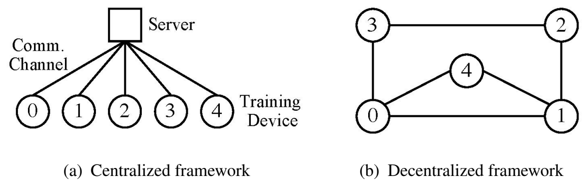


Fig. 1 Different federated learning frameworks.

The convergence rate of decentralized optimization methods depends on network topology. [6] and [7] have analyzed the convergence rate of decentralized optimization methods for deep learning and have shown that the network topology impacts the convergence rate. Their analyses mainly focus on homogeneous training data sets, i.e. the data samples among training devices are independent and identically distributed (IID). However, training data sets in federated learning are usually heterogeneous. In this paper, we investigate the topology design for federated learning and take the data heterogeneity into consideration. In particular, we first explore the hypercube topology, which has $\log n$ diameter for n devices and achieves an efficient information flow rate. In addition, we investigate the topology design problem for federated learning with heterogeneous data. We use data similarity [8] to measure the data heterogeneity. Given the data similarity among training devices, we propose the maximization of the sum of data similarities over the edges in the constructed graph. Intuitively, we attempt to reduce data heterogeneity in the network and improve training efficiency.

It is not trivial to construct the optimal topology for decentralized federated learning with heterogeneous training data. Firstly, it is challenging to compare the performance of different topologies and identify the optimal topology. For example, [7] shows that it is difficult to find a tight bound for the convergence rate of federated learning with IID training data. If a certain topology achieves a fast convergence rate on a loss bound of the convergence rate, there is no guarantee that the topology can significantly improve the training efficiency in practice. In addition, even if the topology is selected, it is challenging to construct the connectivity graph with the given topology such that the data heterogeneity is minimized, which resembles classic graph embedding

[9]. For example, building a ring topology graph where the sum of data similarities over edges in the graph is a traveling salesman problem, which is NP-hard.

In this paper, we first demonstrate the advantage of the hypercube topology. To improve the training efficiency of decentralized federated learning, we present an approximate graph construction method to build a hypercube graph and we attempt to maximize the sum of data similarities over edges in the constructed graph. In addition, we also show a heuristic algorithm to construct a torus graph following a greedy approach. Moreover, we also investigate the communication pattern in hypercube graphs and propose a sequential synchronization scheme to reduce the communication cost during training. A batch synchronization scheme for the hypercube graph is presented where the communication pattern among training devices can be fine-tuned. We have conducted experiments to evaluate our proposed methods using the CIFAR-10 and CIFAR-100 data sets [10]. Our evaluation results show that our proposed graph construction methods can efficiently reduce the data heterogeneity and improve the convergence speed of learning models. Moreover, the evaluation results show that our proposed sequential communication scheme for hypercube graphs can significantly reduce the communication traffic during training while maintaining the convergence performance of learning models.

Our contributions are summarized as follows:

- We investigate the network topology design problem to improve the training efficiency of decentralized federated learning with heterogeneous training data sets.
- We demonstrate the advantage of the hypercube topology for decentralized federated learning and present a hypercube graph embedding method to reduce the data heterogeneity for federated learning with Non-IID data.
- We present a heuristic graph embedding method to construct torus graphs with Non-IID data and maximize the sum of data similarities among the neighbors.
- We propose a sequential synchronization scheme for training over the hypercube topology to reduce the communication cost during training. A batch synchronization scheme is proposed to fine-tune the communication pattern during training.
- We test our proposed methods using real-world data sets. Evaluation results show that the hypercube and torus graph constructed by our algorithms can significantly improve the training efficiency.

The remainder of the paper is structured as follows. We reviewed related work in Section 2. The preliminaries of federated learning and the network model of the decentralized federated learning scheme are introduced in Section 3. Section 4 presents our proposed topology design methods, including hypercube and torus graph construction algorithms. Section 5 focuses on using graph embedding to tackle the Non-IID data by maximizing the sum of data similarities among neighbors. Section 6 proposes a sequential communication scheme to reduce the communication cost during the training process. Our evaluation setups and results are shown in Section 7. Finally, Section 8 concludes the paper.

2 Related Work

Federated learning (FL) is a machine learning technique where training data is stored in local client devices without that data being exchanged with one another [1, 11]. Training without centralized data is an efficient way to protect data privacy. While the training data is decentralized in FL, the parameters of machine learning models can be stored in either a centralized or decentralized way. Depending on where the model parameter is kept, FL schemes can be categorized as centralized or decentralized.

For centralized FL, the parameter server framework [12–14] is the most widely deployed training scheme [2, 15]. In this framework, there is a centralized parameter server to maintain model parameters. All training devices need to synchronize model parameters with the parameter server, which causes a communication bottleneck on the server side. To reduce the communication cost, existing methods can be categorized in two major approaches: reducing the communication frequency [16, 17], and compressing the communication volume

[18–20]. In particular, we can reduce the communication frequency by optimizing the communication scheme and aggregating multiple iterations of local updates in each communication round. Although this approach can efficiently reduce the overall communication cost and speed up the training process of FL, Wang *et al* [21] and Stich [22] show that error terms also accumulate when aggregating local updates.

Compressing the model updates in each communication round is another approach for reducing the communication cost. Common compression techniques include sparsification [19, 23, 24], quantization [18, 25, 26], and low-rank methods [27–29]. Specifically, sparsification reduces the parameter tensor size by selecting a subset of tensor elements. Ozfatura *et al* [19] present a time-correlated sparsification to reduce the communication cost for FL with parameter server implementation. Quantization decreases the parameter tensor size by encoding the tensor in less number of bits. Reisizadeh *et al* [18] present FedPAQ that reduces the communication cost for FL by periodic averaging and quantization. In low-rank methods, model updates would be decomposed into several low-rank matrices, which is a lossy compression method and may break the convergence of the machine learning models during training. Error-feedback strategies [28, 30, 31] are proposed to mitigate the error introduced by compression and maintain the convergence of the learning models. Moreover, adaptive parameter freezing [20] is a promising approach to compress the communication volume by avoiding synchronizing stable model parameters during the training process.

Decentralized FL can resolve the communication bottleneck in centralized schemes by amortizing the communication cost over participating training devices [32]. Decentralized optimization methods have been well-studied [2, 5, 33–36]. Koloskova *et al* [35] investigate the decentralized stochastic optimization algorithms and take the communication compression into consideration. The efficiency of decentralized FL also depends on the network topology design [7]. Neglia *et al* [7] investigate the impact of network topologies on decentralized FL with IID data. Unlike the existing work, we investigate the topology design for learning from Non-IID data and take data similarities into consideration when constructing communication graphs. Moreover, we propose sequential and batch communication schemes to fine-tune the communication pattern for decentralized FL over the hypercube topology.

3 Model

3.1 Centralized Federated Learning

Federated learning is a distributed learning framework where each training device or client has its own data set and will not share its local data set with other clients. We use V to denote the set of training devices that participate in the training process. The number of participating devices is denoted as $|V| = n$. Each training device v has its local data set, which is denoted as D_v. Training a machine learning or deep learning model with the federated learning framework can be formulated as the optimization of the global objective function:

$$\min_x F(x) = \sum\nolimits_{v=1}^{n} w_v f_v(x),$$

where $x \in \mathbb{R}^d$ is the parameter vector of the learning model, $F : \mathbb{R}^d \mapsto \mathbb{R}$ is the global objective function, $f_v : \mathbb{R}^d \mapsto \mathbb{R}$ is the local objective function of each training device v, and w_v is the weight of the device v. The local objective function is usually a loss function, such as the cross-entropy loss, to measure the performance of the learning model on its local data set. In common settings, w_v is usually set to $1/n$ showing that every device has the same weight, or $|D_v|/\sum_{v=1}^{n}|D_v|$ showing that the weight of every device is based on the size of its data set.

Stochastic gradient descent (SGD) is a commonly applied algorithm to optimize the global objective function. Logically, SGD starts from a random solution and iteratively moves toward to the optimal point. In every iteration, each participating device v retrieves a data sample from its local data set and computes the gradient $\nabla f_v(x)$ of its local objective function using the data sample. Then, participating devices would synchronize their gradient information and update the global model. This step can be implemented in either a centralized or a decentralized way.

In the centralized federated learning[1, 37, 38], there is a central server that coordinates participating training devices and maintains the global model parameters. As shown in Fig. 1(a), every participating training device needs to communicate with the central server in order to upload local model updates and download the latest global model parameters. In a fully synchronized setting, training devices need to communicate with the server in every iteration of SGD. In each communication round, training devices need to pull the latest global model parameters from the server and push their local updates to the server, which would easily cause congestion at the network interface of the server. The congestion at the central server would significantly extend the training time. [18] and [39] show that the communication frequency can be reduced by allowing some stale model updates, and the global model still can converge. The overall communication volume can be reduced by decreasing the communication frequency. However, the congestion at the central server still exists and affects training efficiency.

3.2 Decentralized Federated Learning

With the development of wireless communication technology, device-to-device (D2D) communication among mobile devices become more and more reliable. For example, Ozyurt *et al* [40] present a Li-Fi based D2D communication system for industrial IoT devices. By utilizing D2D communication, federated learning can be implemented in a decentralized manner. Specifically, training devices can directly communicate with peers and exchange model updates. Decentralized federated learning does not rely on central servers and avoids congestion at servers, which can improve communication efficiency and accelerate the training process.

In decentralized federated learning, each device still needs to sample local data and compute local model updates. Differently from centralized federated learning, each device needs to maintain a set of local model parameters. In each training iteration, each device needs to gather neighbors' model updates, aggregate them with local updates, and modify local model parameters with the aggregated updates. Formally, let $x_{v,t}$ denote the vector of model parameters at the t-th training iteration of device v. Then, the model updates in the decentralized optimization can be formulated as

$$x_{v,t+1} = x_{v,t} - \alpha \sum\nolimits_{j=1}^{n} m_{vj} \nabla f_j(x_{j,t}),$$

where $\alpha \in [0,1]$ is the hyper-parameter representing the learning rate and $m_{vj} \in [0,1]$ represents the weights of neighbor updates. The weight $m_{vj} = 0$ if devices v and j are not connected. Otherwise $0 < m_{vj} \leq 1$. In addition, we assume $m_{vj} = m_{jv}$, which means the mutual influence between devices v and j are equal. A common setting is letting $m_{vj} = 1/N(v)$, where $N(v)$ represents the number of neighbors of device v. This setting means that every neighbor makes the same contribution of the model updates.

We use a graph $G = (V, E)$ to model the network topology of training devices, as illustrated in Fig. 1(b). The vertex set consists of training devices. There is an edge $(i, j) \in E$ if devices i and j are connected. Notably, we assume the D2D communication channels are full-duplex, and edges in E are undirected. To analyze the network topology, we use a connectivity matrix M to model the graph G. For n participating devices, M is a $n \times n$ matrix. The matrix element $M(i,j) = m_{ij}$ shows the weight of connections between devices i and j. Notably, we assume a device is connected to itself by default, and $M(i,i) = m_{ii} > 0$. For the undirected graph G, the connectivity matrix M is symmetric. Moreover, we assume M is a doubly stochastic matrix, i.e., each of the rows and columns in M sums to 1 or formally $\sum_i M(i,j) = \sum_j M(i,j) = 1$. The spectral gap $\delta(M)$ of the matrix M can measure the information flow efficiency in graph G. The formal definition of the spectral gap is shown as follows.

Definition 1 For a symmetric double stochastic matrix M with eigenvalues $|\lambda_1(M)| \leq \cdots \leq |\lambda_{n-1}(M)| < |\lambda_n(M)| = 1$, its spectral gap $\delta(M)$ is the difference between the moduli of the two largest eigenvalues of M. Formally, $\delta(M) \triangleq 1 - |\lambda_{n-1}(M)|$.

3.3 Data Heterogeneity

In addition to the network topology, we also consider the impact of data heterogeneity on federated learning. In particular, training data on participating devices in federated learning usually are not independent and identically distributed (Non-IID). For example, sensor data gathered from IoT devices located in different areas is Non-IID. We use the similarity among local data sets of training devices to measure the data heterogeneity. Formally, let

Table 1 Notations and Explanations

Notation	Explanation
V	The set of training devices
v	An individual training device
n	The number of participating devices in V
$G = (V, E)$	The topology of decentralized federated learning
D_v	The training data set of device v
F	The global objective function
f_v	The local objective function of device v
$x_{v,t}$	Device v's model parameter at the t-th iteration
M,$M(i,j)$	The connectivity matrix and its element
S,$S(i,j)$	The data similarity matrix and its element
$\lambda_i(M)$	The i-th smallest eigenvalue of M
$\delta(M)$	The spectral gap of M
H_d	The connectivity graph of the d-D hypercube

$S \triangleq [S(i,j)]_{1\le i,j\le n}$ denote the data similarity matrix, where $S(i,j)$ is the similarity between local data sets of device i and j. The similarity $S(i,j)$ is defined as the probability that a data sample from D_i is similar to at least one data sample from D_j. The standard that measures whether two data samples are similar varies with application scenarios. For image classification applications, two data samples are similar if they have the same ground-truth label. Moreover, there are different formulations to evaluate the data heterogeneity. Bars *et al* [41] present a quantity named neighborhood heterogeneity. For a node, its neighborhood heterogeneity is based on aggregating the differences with its neighbors with 2-norm. We follow a similar approach while using 1-norm based on graph embedding.

4 Topology Design

The convergence rate of distributed federated learning heavily depends on the network topology. Theoretical analyses [35] have shown that the convergence rate of distributed training is closely related to the spectral gap of the connectivity matrix M. Formally, the model parameter $x_{v,t}$ at the t-th training iteration of device v converges linearly when the connectivity matrix M is symmetric doubly stochastic, as stated in Theorem 1.

Theorem 1 The model parameter $x_{v,t}$ converges linearly to $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_{v,0}$ with the rate

$$\sum_{i=1}^{n} \|x_{v,t}-\bar{x}\|^2 \le (1-\gamma\delta(M))^{2t} \sum_{i=1}^{n} \|x_{v,0}-\bar{x}\|^2,$$

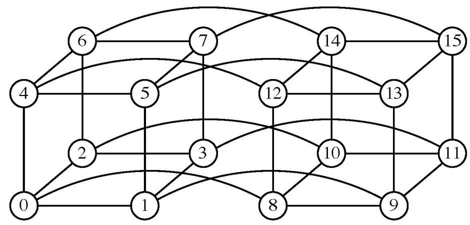


Fig. 2 4-dimensional hypercube graph.

where $\gamma \in (0,1]$ and M is a symmetric doubly stochastic connectivity matrix.

Notice that $\gamma\delta(M) \in (0,1]$ and $(1-\gamma\delta(M)) \in [0,1)$, we have $\lim_{t\to+\infty}(1-\gamma\delta(M))^{2t} = 0$. This shows that the model parameter $x_{v,t}$ will converge eventually. Moreover, from the convergence rate shown in Theorem 1, we notice that the spectral gap $\delta(M)$ of the connectivity matrix M plays an important role. Especially when the number of devices n is large, the difference in the spectral gap δ of different network topologies becomes significant. It is shown in [35] that the spectral gap δ of a ring and 2-dimensional (2-D) torus is $O(1/n^2)$ and $O(1/n)$, respectively. According to Theorem 1, a greater δ leads to a higher convergence rate. Therefore, compared to the ring topology, the 2-D torus graph has a faster convergence speed. This difference shows it is worth optimizing the network topology design for improving the training efficiency of distributed federated learning. To optimize the network topology, a natural question to ask is: what causes the significant difference in the spectral gaps of different graphs?

By comparing the difference between ring and torus topology, we observe that the diameter of the ring graph is greater than the diameter of the torus graph, given the same number of vertices in the graph. Intuitively, the larger diameter of the ring graph may cause the smaller spectral gap and the slower convergence speed. Inspired by the observation, we investigate the hypercube topology whose diameter increases in $O(log\ n)$ with the number of vertices n in the graph. There are $n = 2^d$ vertices in a d-dimensional (d-D) hypercube. The vertex set of the hypercube graph is defined on $\{0,1\}^d$. In the d-dimensional hypercube, each vertex has exactly d neighbors. Two vertices are connected if their labels (in binary code) differ in exactly one dimension. For example, the structure of a 4-dimensional hypercube is

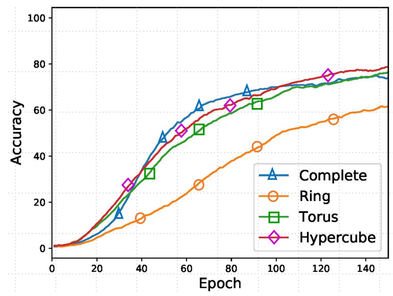


Fig. 3 Training ResNet-50 on CIFAR-100 with 64 workers.

shown in Fig. 2. Let H_d denote the connectivity matrix of the d-dimensional hypercube. H_d can be recursively defined by the following equation:

$$H_d = \frac{1}{d+1}\begin{bmatrix} dH_{d-1} & I_{2^{d-1}} \\ I_{2^{d-1}} & dH_{d-1} \end{bmatrix},$$

where $I_{2^{d-1}}$ represents the $2^{d-1} \times 2^{d-1}$ identity matrix, and the base case $H_0 = [1]$. Given the connectivity matrix, we can calculate the spectral gap of the hypercube topology, as shown in Theorem 2.

Theorem 2 The spectral gap $\delta(H_d)$ is $2/d$ and $\delta(H_d)^{-1} = O(d) = O(\log n)$.

Proof. We can verify that the eigenvalues of H_d are $\frac{1}{d}(n - 2|I|)$ where $I \subseteq \{1, \ldots, d\}$. The moduli of the two largest eigenvalues of H_d are n/d and $(n-2)/d$. According to the definition, the spectral gap $\delta(H_d)$ of H_d is $2/d$, and $\delta(H_d)^{-1} = O(d) = O(\log n)$. ■

Based on the spectral gap, we can analyze the convergence speed of the decentralized federated learning over the hypercube graph. Compared to ring and torus topologies, the spectral gap of the hypercube is much larger, especially when the number of participating devices is large. According to Theorem 1, the convergence speed of distributed federated learning over the hypercube graph is faster.

We have evaluated the convergence rate of SGD over different network topologies. Fig. 3 shows the preliminary experiment results when there are 64 participating training devices. This figure shows the top-1 accuracy of ResNet-50 model when training on the CIFAR-100 data set. Data samples in the CIFAR-100 are randomly shuffled and allocated to participating devices. From the figure, we can observe that the model convergence speed heavily depends on the topology of the communication graph. For example, to achieve 60% accuracy, hypercube takes 73 epochs, while torus and ring need 83 and 144 epochs, respectively. Compared to the hypercube topology, torus and ring are 13.7% and 97.3% slower. The preliminary result shows that optimizing the topology of the communication graph can significantly improve the training efficiency for decentralized federated learning.

From the theoretical analyses and the preliminary experiment results, we notice that the network topology would impact the convergence rate of decentralized federated learning. In addition to the network topology, data heterogeneity also affects the convergence rate. Too many updates from extremely heterogeneous data may diverge the learning model. In this paper, we propose to jointly consider those two factors. When scheduling the communication among participating training devices, we attempt to find a graph G such that the spectral gap of the connectivity matrix is maximized and the heterogeneity of neighbor nodes in the graph is minimized. This is not a trivial problem and there may be a trade-off between the information flow efficiency and the data heterogeneity given a set of decentralized training devices. It is challenging to minimize the data heterogeneity of neighbor nodes while maintaining a desired network topology.

There is some recent effort on diameter minimization based on a fixed node degree for a given number of nodes, but their results generate a random graph with probabilistic guarantee [42]. We investigate more deterministic approaches to optimize graph embedding with multiple graph topologies, and we use the spectral gap as a mathematical tool to analyze the approximation property of our proposed methods.

5 Graph Embedding for Non-IID Data

In this section, we focus on optimizing the communication graph for heterogeneous data. For the similarity, intuitively, when data distributions of two workers are similar, we should connect them together so that the disturbance from other non-similar workers can be avoided. The authors in [43] use this intuition to design the communication graph. The experimental findings in [43] confirm this intuition. We use data similarity to measure the data heterogeneity of communication graphs. In particular, each edge in the communication graph G shows

Algorithm 1 Max-Similarity Hypercube Construction

Input: The complete graph $G = (V, E)$ with $|V| = n = 2^d$ and similarity matrix S

Output: The d-D hypercube G_d with max-similarity

Define $G_0 = (V_0, E_0)$, where $V_0 = V$ and $E_0 = \{\}$ //initialization for G_0

for dimension $i = 0$ to $d - 1$ **do**

  //determine G_{i+1} from G_i

  for each virtual node pair vn and vn' in G_i **do**

    call **Virtual_Node_Similarity(**vn**,** vn'**)**

  Apply Blossom's algorithm to V_i based on virtual node similarity in G_i

  Each matching pair in V_i forms a virtual node in V_{i+1}

  A set of one-to-one mapping connections along dimension i in each matching pair in V_i forms E_{i+1}, together with existing links in E_i

Virtual_Node_Similarity(vn**,** vn'**)**

//determine virtual node similarity in G_i

for each node pair $(u, v), u \in vn, v \in vn'$ **do**

  sum up $S(u, v)$ //both $u, v \in V$

the data similarity between two vertices induced on the edge. Our objective is to select a set of edges such that the summation of similarity among neighbors is maximized and the desired network topology is maintained. This process resembles classic graph embedding, where a target graph, i.e., hypercube or ring, is embedded in a given graph, i.e., a completely connected graph in this case. This optimization problem is challenging even for generating the max-similarity for a graph with a simple ring topology. Finding such a graph with ring topology is equivalent to a traveling salesman problem, which is NP-hard. We propose two heuristic graph construction methods for hypercube and torus topologies from a given complete graph, respectively.

5.1 Hypercube Graph Construction

Given a complete graph $G = (V, E)$ with the similarity matrix S, we denote the virtual network of level i as $G_i = (V_i, E_i)$, where $i = 0, 1, ..., d - 1$. Initially, $G_0 = (V_0, E_0)$, where V_0 is V and E_0 is empty. Our algorithm iteratively constructs G_{i+1} from G_i, $i = 0, 1, ..., d - 1$. The hypercube construction algorithm is shown in Algorithm 1. It is a dimension-based perfect matching using Blossom's algorithm [44], which constructs a maximum matching on a graph in polynomial time. Blossom's algorithm starts with an empty matching. Then, it repeatedly increases the size of the matching by one by finding and utilizing an augmented path in the graph at each iteration. When no more augmented paths exist, the result is a maximum matching.

Our hypercube construction process first applies Blossom's algorithm to find matching pairs of physical nodes. Each matching pair forms a 1-D cube, which is a virtual node of level 1. Then, Blossom's algorithm is repeatedly applied to virtual nodes of level i to form virtual nodes of level $i + 1$. G_{i+1} is constructed as follows: each matching pair in G_i is a virtual node in V_{i+1}. A one-to-one node-level connection along dimension i in each matching pair plus all existing links in G_i form E_{i+1}. The construction stops when there is only one virtual node, which is the d-D hypercube, G_d. The construction of a 3-D hypercube in three iterations is illustrated in Figure 4.

Figure 5 shows two virtual nodes (i.e. two 3-D hypercubes). The virtual node on the left is matched to the virtual node on the right with the maximum similarity. To find the maximum similarity matching between two virtual nodes of level k (i.e. two k-D hypercubes), there are $k!2^k$ choices (i.e. the number of automorphisms). At level 0, the virtual node is the real node and therefore the matching is at the maximum. In subsequent levels, our construction algorithm approximates the maximum using the total pairwise similarity between two virtual nodes, which is the sum of pairwise similarity between two physical nodes with one from each virtual node of level k. This approximation has a complexity of $(2^k)^2$. Once matching pairs are constructed in the i-th iteration, our algorithm randomly selects one-to-one node-pair (i.e. nodes in the original G) connections along the i-th dimension without considering different rotations.

Our proposed method for hypercube graph construction has an approximation ratio of $1/d$, i.e., the sum of data similarities over edges in the graph is at least $1/d$ of the optimal solution. The complexity of our proposed method is shown in Theorem 3. The approximation property of our proposed method is shown in Theorem 4.

Theorem 3 The hypercube graph construction method shown in Algorithm 1 has a complexity of

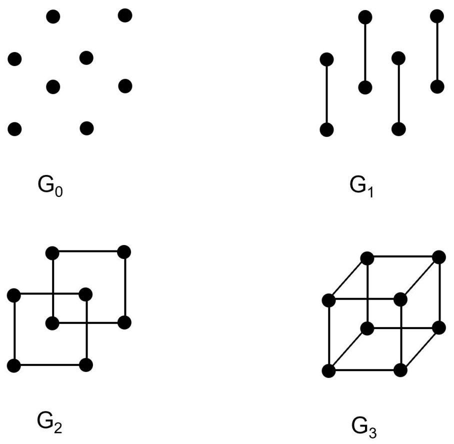


Fig. 4 A 3-D max-similarity hypercube construction process.

$O(n^4)$, where n is the number of nodes.

Proof. For each dimension i, when Blossom's Algorithm is applied to G_i, the time complexity is $O(|E_i||V_i|^2) = O(|V_i|^4)$ based on [44]. $O(|V_i|^4) = O((2^{d-i})^4) = O((n2^{-i})^4)$. The time complexity of calculating the similarities of all virtual node pairs in G_i is $O(|V_i|^2(2^i)^2) = O((n2^{-i})^2 2^{2i}) = O(n^2)$. The time complexity of our hypercube construction algorithm is $\sum_{i=0}^{d-1}(O((n2^{-i})^4) + O(n^2)) = O(\sum_{i=0}^{d-1}(n2^{-i})^4) + O(\sum_{i=0}^{d-1} n^2) = O(n^4) + O(n^2 log\, n) = O(n^4)$. ■

Theorem 4 The hypercube graph construction method shown in Algorithm 1 is $1/d$-approximate.

Proof. Our hypercube graph construction method would iteratively maximize the similarities over edges in every dimension. In the first iteration, our method would pick $1/d$ of total edges for the hypercube graph. Considering that we apply the maximum weight perfect matching in this iteration, any other matching plans that select a $1/d$ portion of total edges would have a smaller sum of data similarities. Therefore, sum of data similarities over edges in the graph generated by Algorithm 1 is at least $1/d$ of the optimal solution. ■

5.2 Torus Graph Construction

In addition to the hypercube topology, we have investigated the torus graph construction for heterogeneous data such that the sum of data similarities over edges in the graph is maximized.

Given the complete graph $G = (V, E)$ of n nodes with the similarity matrix S, our torus construction algorithm creates a 2-D torus in two major steps: ring construction and ring matching. We define $m = \sqrt{n}$. The ring construction step creates m rings, $R_1, R_2, ..., R_m$, in sequence. Each ring contains m nodes. The ring matching step connects the m rings, $R_1, R_2, ..., R_m$, to form a ring of rings, which is a 2-D torus. This process is described in Algorithm 2.

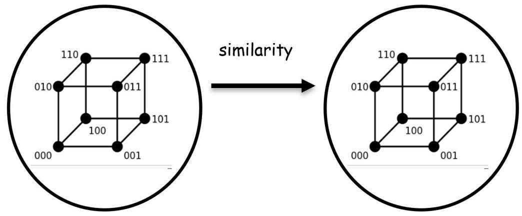


Fig. 5 Max-similarity hypercube matching in G_3.

To construct a new ring R_i ($i = 1, 2, ..., m$) of size m, our algorithm randomly selects an unmatched node u in G as the head in the ring. It then finds an unmatched node v with the maximum similarity to u. This new node v is set to be the new head. Repeat these two steps until the new ring size is m. Then, connect the head and tail in the new ring to form a circle. This new ring construction process is repeated m times to create m rings.

The ring matching process begins with randomly choosing an unmatched ring R as the head of the rings of rings, i.e. the 2-D torus. Then our algorithm finds an unmatched ring R' with the maximum similarity to R. Next, ring R' is set to be the new head. To match each ring R_i ($i = 1, 2, ..., m - 1$), repeat the last two steps. Then connect the head and tail of the rings of rings to form a 2-D torus.

The 2-D torus construction algorithm is also heuristic. Clearly, the ring of rings created is a 2-D torus. The maximum ring similarity between two rings is the summation of one-to-one node pair similarities. There are totally m^2 possible matchings with various rotations, including m rotations of a given ring and another m rotations after flipping the ring. Figure 6 shows an example for $m = 6$. The ring on the left is matched to the ring on the right with the maximum similarity rotation (flip, then rotate) among all possible rotations. The complexity of our proposed method is shown in Theorem 5.

Theorem 5 The torus graph construction method shown in Algorithm 2 has a complexity of $O(n^2)$, where

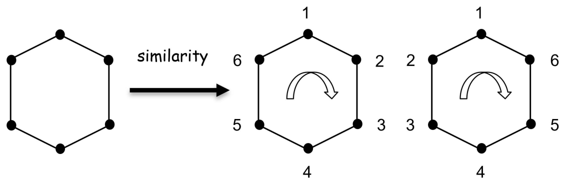


Fig. 6 Ring matching through rotation and flipping (the rightmost ring).

n is the number of nodes.

Proof. In Construct_Rings, the outer for-loop in Line 5 repeats $\sqrt{n}$ times. The inner for-loop in Line 7 iterates $\sqrt{n}-1$ times. In Line 8, at most $n-1$ nodes are checked. The total run time of ring construction is $O(\sqrt{n}\sqrt{n}\, n) = O(n^2)$.

In Match_Rings, the for-loop in Line 14 repeats $\sqrt{n}-1$ times. In Line 15, at most $\sqrt{n}-1$ rings are checked. For each pair of rings of size $\sqrt{n}$, the time complexity of computing their similarity is $O(\sqrt{n}\sqrt{n}) = O(n)$. The total run time of ring matching is $O(\sqrt{n}\sqrt{n}\, n) = O(n^2)$. The time complexity of our torus construction algorithm is $O(n^2) + O(n^2) = O(n^2)$. ■

Algorithm 2 Max-Similarity Torus Construction

Input: The complete graph G with the similarity matrix S
Output: 2-D torus with the maximum total similarity G'

1: $m \leftarrow \sqrt{n}$
2: call **Construct_Rings**
3: call **Match_Rings**
4:
5: **Construct_Rings**
6: **for** $i = 1$ to m //construct R_i **do**
7: randomly select an unmatched node u in G
8: **for** $j = 2$ to m **do**
9: find an unmatched node v in G that has the maximum similarity to u
10: set v to u //v becomes the head of R_i
11: connect the head and tail of R_i.
12:
13: **Match_Rings**
14: //connect $R_1, R_2, ..., R_m$ to form a ring of rings
15: randomly select an unmatched ring R
16: **for** $i = 2$ to m //match R_{i-1} **do**
17: find an unmatched ring R' with the maximum similarity to R
18: set R' to R // R' becomes the head of the ring of rings
19: connect the head and tail of the ring of rings to form a 2-D torus

6 Reducing Communication Frequency

In addition to the network topology design, adjusting the communication frequency among training devices can reduce the communication volume and efficiently speed up the training process. Existing studies mainly follow either a synchronous or asynchronous approach. In synchronous federated learning, all participating devices need to synchronize their model parameters in every l iteration, where $l \geq 1$ is a hyper-parameter representing the staleness limitation. A large l can efficiently reduce the communication frequency, but may also break the convergence of machine learning models [13].

In an asynchronous scheme, training devices no longer need to wait for neighbors for model synchronization. However, the overall communication volume is not significantly reduced in the asynchronous scheme. Different from existing approaches, we present a batch synchronization scheme for distributed federated learning over the hypercube topology. Intuitively, our proposed scheme can fine-tune the synchronization frequency of nodes in each dimension in the hypercube graph, which helps to reduce the network traffic during training and to improve the training efficiency.

To reduce the communication cost, we first present a sequential communication scheme for decentralized federated learning in hypercube topology. In traditional federated learning, all participating devices perform communication in parallel. For example, if there are 4 devices. they perform parallel communication in each synchronization round, which can be represented by $0||1||2||3$, where $||$ denotes the parallel communications. For a d-dimensional hypercube, each device needs to communicate with d neighbors in each synchronization round. The traditional synchronization scheme would introduce a large communication cost. Differently to setting up a fixed synchronization barrier, we propose letting training devices synchronize their model parameters in sequence by each dimension in the hypercube connectivity graph. In our sequential communication scheme, each device only synchronizes with one neighbor in each communication round. The neighbor selection sequence of each device is sorted by dimension. For example, the commu-

Algorithm 3 **Sequential Communication Scheme**

Input: The dimension d of the hypercube graph
Output: A sequential communication schedule

1: **while** training process is not completed **do**
2: $i \leftarrow 0$
3: **for** every device $v \in V$ **do**
4: Select the neighbor at the $(i \mod d)$-th dimension for synchronization
5: $i \leftarrow i + 1$ //iteration i

nication of 4 devices is organized as $0||1, 2||3$ in the first round, and $0||2, 1||3$ in the second round. Specifically, the device 0 only communicates with device 1 in the first round, and synchronizes with device 2 in the following round. In the sequential communication scheme, the communication cost is reduced by $1/d$ compared to the traditional federated learning scheme.

Detailed steps of our proposed sequential communication scheme is shown in Algorithm 3. In particular, while the training process is not completed. Every training device would perform model synchronization with one neighbor node in a communication round. Line 2 initializes a counter to keep a record of the number of iterations. The loop in lines 3-4 would select a neighbor node for every training device. At iteration i, the neighbor at the $i \mod d$ dimension would be selected for synchronization, as shown in line 4. Line 5 would increment the iteration counter i. With the sequential communication scheme, the overall communication cost is reduced by $1/d$.

In addition to the sequential communication scheme, we present a more flexible communication scheme for decentralized federated learning over the hypercube topology. In particular, we can fine-tune the communication cost in each synchronization round. For training with n devices, we can use a $\log_2 n$-bit binary mask $\mathbf{b}$ to indicate which dimensions the synchronization should be performed on. In the binary mask, 0 represents skipping the synchronization in the corresponding dimension and 1 means performing the synchronization in this round. For example, the sequential communication scheme for a 3-dimensional hypercube with eight devices can be encoded as $001, 010, 100$, and repeat. The communication pattern can be fine-tuned by adjusting the binary mask for each synchronization round.

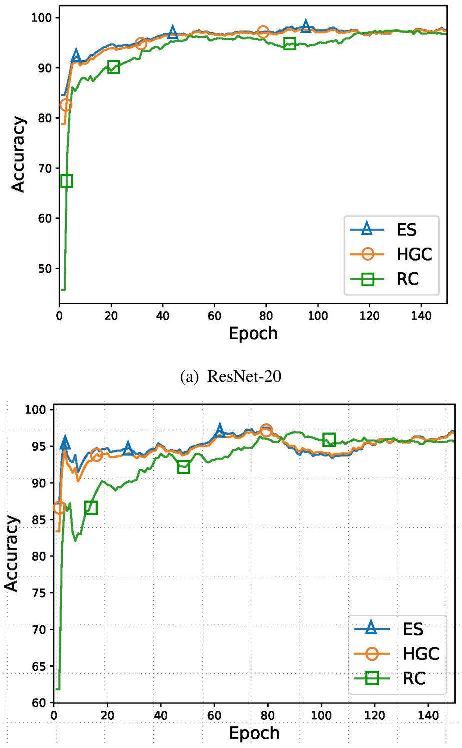
7 Experiment

7.1 Experiment Setup

Out testbed is built based on a computing cluster that has 8 NVIDIA Tesla V100 GPUs with 448 GB RAM and 5.9 TB storage space. All GPUs are connected by NVLinks that provide 300GB/s bandwidth per GPU. We have evaluated our proposed methods with different network sizes. When there are more than eight nodes in the network, multiple worker nodes would be assigned to a GPU device. Each device can handle multiple training processes if the overall workload does not exceed the GPU memory limitation.

We implement our proposed schemes in PyTorch. We use the OpenMPI package as the backend that coordinates the communication among training devices. In particular, we modify the topology module of PyTorch to integrate our graph construction methods. The network topology during training is adjusted by the connectivity matrix generated by our algorithms. We utilize the CIFAR-10 data set to simulate the heterogeneous data distribution among training devices. We treat the images from the same class as homogeneous data and mix images from different classes to construct the heterogeneous data. During the experiment, we assume the data similarity matrix is known. Given a data similarity matrix, we assign heterogeneous data from different image classes to training devices to fulfill the data heterogeneity indicated by the matrix. When testing our proposed synchronization scheme, we use both CIFAR-10 and CIFAR-100 data sets.

In our experiments, we compared the performance of different graph construction methods. In particular, our proposed hypercube and torus graph construction methods are denoted as HGC and TGC, respectively. We also implement an exhaustive search method to find the optimal graph where the sum of data similarities over edges in the graph is maximized. The exhaustive search method is denoted as ES. ES finds the optimal solution in non-polynomial time, and cannot be applied to solve large-scale problems. In the graph construction experiment, we set n to 16. In addition, we implement a random construction method, which is denoted as RC. The RC method can only guarantee the topology



(a) ResNet-20

(b) ResNet-50

Fig. 7 Evaluation of hypercube construction methods.

of the generated graph, but it cannot reduce the data heterogeneity of the generated graph. Moreover, we also compare different synchronization schemes. We denote our sequential synchronization scheme as SS and the traditional full synchronization scheme as FS. Based on our batch synchronization scheme, we also implement a hybrid synchronization scheme that reduces the communication frequency of the full synchronization scheme by $1/3$, i.e., performing a full synchronization in every 3 iterations such that the communication cost is equivalent to SS. The hybrid synchronization scheme is denoted as HS. When evaluating different synchronization schemes, the number of devices n is set to 8.

7.2 Experiment Results

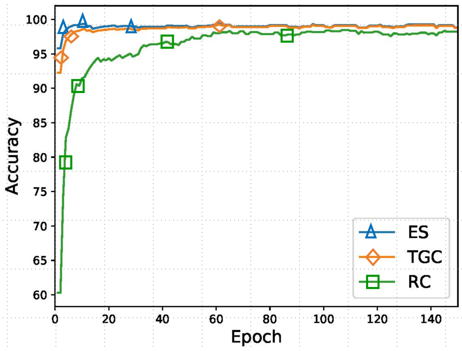
The evaluation results of hypercube construction algorithms are shown in Fig. 7. In particular, Fig. 7(a) shows the experiment results of the ResNet-20 model and Fig. 7(b) shows the results of the ResNet-50 model. From 7(a) and 7(b), we can find that our proposed HGC method achieves a better convergence rate compared to randomly constructing a hypercube graph in RC. The experiment results show that finding a hypercube graph that increases the sum of data similarities over edges in the graph can improve the training efficiency. In addition, our HGC method has a similar convergence trace as ES. This shows that our proposed method can find the near-optimal graph in terms of data similarity maximization. Compared to ES, our proposed method has polynomial time complexity and is more time efficient when constructing hypercube graphs.

Experiment results of torus graph construction methods are shown in Fig. 8. The figure illustrates the model convergence property when training with the communication graph constructed by different graph construction methods. From the figure, we find that our TGC method outperforms RC in both the ResNet-20 and ResNet-50 models. Our proposed TGC method can reduce the data heterogeneity in the generated torus graph, which helps improve the convergence rate of machine learning models. For example, TGC takes about 30 epochs less than RC to achieve 95% model accuracy with training with ResNet-20. In addition, the performance of TGC is close to the optimal graph construction method ES.

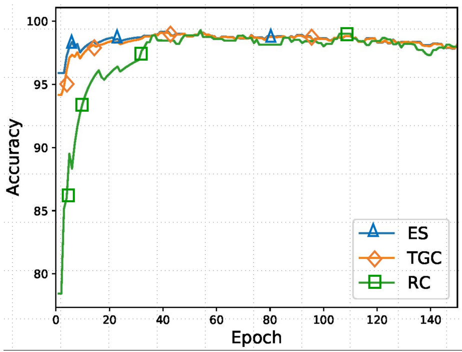
Fig. 9 shows the evaluation results of different synchronization schemes on the ResNet-50 model. Fig. 9(a) shows the experiment results over the CIFAR-10 data set and Fig. 9(b) illustrates the results over the CIFAR-100 data set. From the figure, we can observe that our sequential synchronization scheme SS requires lower communication cost to achieve the same model accuracy as the traditional FS scheme. For example, when reaching 80% accuracy for CIFAR-10, SS has a 19% lower communication cost compared to FS, and the saving is more significant when reaching the same higher level of accuracy. In addition, the model trained with the sequential synchronization scheme converges to the same accuracy as FS. This shows that our SS scheme can efficiently reduce the communication cost during training without harming the model convergence.

8 Conclusion

In this paper, we investigate the topology design problem for decentralized federated learning with heterogeneous training data. We demonstrate the advantage of hypercube topology by showing its spectral gap and theoretical convergence rate. To reduce the data heterogeneity

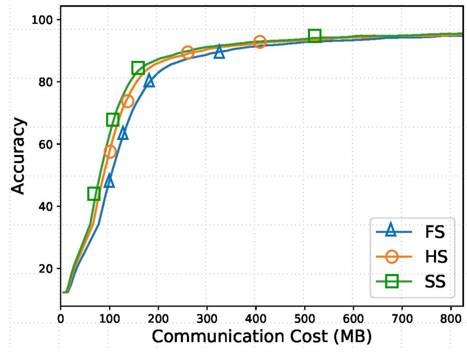


(a) ResNet-20

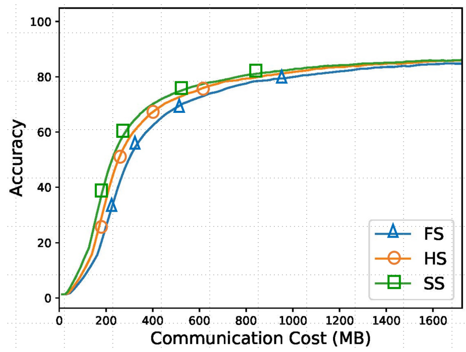


(b) ResNet-50

Fig. 8 Evaluation of torus construction methods.



(a) CIFAR-10



(b) CIFAR-100

Fig. 9 Evaluation of synchronization schemes.

during the training process, we present graph construction methods for both hypercube and torus topologies to carefully select neighbors for each training device and increase the overall data similarities in the generated graph. Our hypercube graph construction method is $1/d$-approximate. In addition to the topology design, we propose a sequential synchronization scheme for training in hypercube graphs. Also, a batch synchronization scheme is proposed to fine-tune the communication patterns during training. To evaluate our proposed methods, we conduct experiments over CIFAR-10 and CIFAR-100 data sets. Training traces of ResNet models show that our proposed graph construction methods can accelerate the training process. Moreover, our proposed synchronization schemes can significantly reduce the overall communication cost during training.

References

[1] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, Federated learning: Strategies for improving communication efficiency, *arXiv preprint arXiv:1610.05492,* 2016.

[2] 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, *Advances in Neural Information Processing Systems,* vol. 30, 2017.

[3] R. I. Ansari, C. Chrysostomou, S. A. Hassan, M. Guizani, S. Mumtaz, J. Rodriguez, and J. J. Rodrigues, 5g d2d networks: Techniques, challenges, and future prospects, *IEEE Systems Journal,* vol. 12, no. 4, pp. 3970–3984, 2017.

[4] I. Hegedűs, G. Danner, and M. Jelasity, Decentralized learning works: An empirical comparison of gossip learning and federated learning, *Journal of Parallel and Distributed Computing,* vol. 148, pp. 109–124, 2021.

[5] K. Scaman, F. Bach, S. Bubeck, Y. T. Lee, and L. Massoulié, Optimal algorithms for smooth and strongly convex distributed optimization in networks, in *Proc. of International Conference on Machine Learning. PMLR*, 2017, pp. 3027–3036.

[6] A. Koloskova, T. Lin, S. U. Stich, and M. Jaggi, Decentralized deep learning with arbitrary communication compression, in *ICLR 2020 - International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=SkgGCkrKvH

[7] G. Neglia, C. Xu, D. Towsley, and G. Calbi, Decentralized gradient methods: does topology matter?, in *Proc. of International Conference on Artificial Intelligence and Statistics. PMLR*, 2020, pp. 2348–2358.

[8] S. Wang, M. Lee, S. Hosseinalipour, R. Morabito, M. Chiang, and C. G. Brinton, Device sampling for heterogeneous federated learning: Theory, algorithms, and implementation, in *Proc. of IEEE INFOCOM 2021- IEEE Conference on Computer Communications. IEEE*, 2021, pp. 1–10.

[9] F. T. Leighton, *Introduction to parallel algorithms and architectures: Arrays, trees, hypercubes*, Morgan Kaufmann Publishers, Inc. 1992.

[10] A. Krizhevsky and G. Hinton, Learning multiple layers of features from tiny images, Technical Report, Computer Science Dept., Univ. of Toronto, CA, 2009.

[11] S. Duan, D. Wang, J. Ren, F. Lyu, Y. Zhang, H. Wu, and X. Shen, Distributed artificial intelligence empowered by end-edge-cloud computing: A survey, *IEEE Communications Surveys & Tutorials,* vol. 25, no. 1, pp. 591–624, 2023.

[12] M. Li, L. Zhou, Z. Yang, A. Li, F. Xia, D. G. Andersen, and A. Smola, Parameter server for distributed machine learning, in *Big learning NIPS workshop*, vol. 6, no. 2, 2013.

[13] Q. Ho, J. Cipar, H. Cui, S. Lee, J. K. Kim, P. B. Gibbons, G. A. Gibson, G. Ganger, and E. P. Xing, More effective distributed ml via a stale synchronous parallel parameter server, in *Advances in neural information processing systems*, 2013, pp. 1223–1231.

[14] M. Li, D. G. Andersen, A. J. Smola, and K. Yu, Communication efficient distributed machine learning with the parameter server, *Advances in Neural Information Processing Systems,* vol. 27, 2014.

[15] O. Dekel, R. Gilad-Bachrach, O. Shamir, and L. Xiao, Optimal distributed online prediction using mini-batches. *Journal of Machine Learning Research,* vol. 13, no. 1, 2012.

[16] V. Smith, S. Forte, M. Chenxin, M. Tak´aˇc, M. I. Jordan, and M. Jaggi, Cocoa: A general framework for communication-efficient distributed optimization, *Journal of Machine Learning Research,* vol. 18, p. 230, 2018.

[17] J. Wang and G. Joshi, Adaptive communication strategies to achieve the best error-runtime trade-off in local-update sgd, in *Proc. of Machine Learning and Systems*, vol. 1, pp. 212–229, 2019.

[18] A. Reisizadeh, A. Mokhtari, H. Hassani, A. Jadbabaie, and R. Pedarsani, Fedpaq: A communication-efficient federated learning method with periodic averaging and quantization, in *International Conference on Artificial Intelligence and Statistics. PMLR*, 2020, pp. 2021–2031.

[19] E. Ozfatura, K. Ozfatura, and D. Gündüz, Time-correlated sparsification for communication-efficient federated learning, in *2021 IEEE International Symposium on Information Theory (ISIT). IEEE*, 2021, pp. 461–466.

[20] C. Chen, H. Xu, W. Wang, B. Li, B. Li, L. Chen, and G. Zhang, Communication-efficient federated learning with adaptive parameter freezing, in *Proc. of IEEE ICDCS. IEEE*, 2021, pp. 1–11.

[21] J. Wang and G. Joshi, Cooperative sgd: A unified framework for the design and analysis of communication-efficient sgd algorithms, *arXiv preprint arXiv:1808.07576,* 2018.

[22] S. U. Stich, Local sgd converges fast and communicates little, *arXiv preprint arXiv:1805.09767,* 2018.

[23] H. Sun, Y. Shao, J. Jiang, B. Cui, K. Lei, Y. Xu, and J. Wang, Sparse gradient compression for distributed sgd, in *DASFAA. Springer*, 2019, pp. 139–155.

[24] N. Ivkin, D. Rothchild, E. Ullah, V. Braverman, I. Stoica, and R. Arora, Communication-efficient distributed sgd with sketching, *arXiv preprint arXiv:1903.04488,* 2019.

[25] Y. Li, J. Park, M. Alian, Y. Yuan, Z. Qu, P. Pan, R. Wang, A. Schwing, H. Esmaeilzadeh, and N. S. Kim, A network-centric hardware/algorithm co-design to accelerate distributed training of deep neural networks," in *MICRO. IEEE*, 2018, pp. 175–188.

[26] Y. Yu, J. Wu, and J. Huang, Exploring fast and communication efficient algorithms in large-scale distributed networks, *arXiv preprint arXiv:1901.08924,* 2019.

[27] H. Wang, S. Sievert, Z. Charles, S. Liu, S. Wright, and D. Papailiopoulos, Atomo: Communication-efficient learning via atomic sparsification, *arXiv preprint arXiv:1806.04090,* 2018.

[28] T. Vogels, S. P. Karimireddy, and M. Jaggi, Powersgd: Practical lowrank gradient compression for distributed optimization, *Neur IPS,* 2019.

[29] M. Cho, V. Muthusamy, B. Nemanich, and R. Puri, Gradzip: Gradient compression using alternating matrix factorization for large-scale deep learning, *Neur IPS,* 2019.

[30] S. P. Karimireddy, Q. Rebjock, S. Stich, and M. Jaggi, Error feedback fixes signsgd and other gradient compression schemes, in *International Conference on Machine Learning. PMLR*, 2019, pp. 3252–3261.

[31] C. Xie, S. Zheng, O. O. Koyejo, I. Gupta, M. Li, and H. Lin, Cser: Communication-efficient sgd with error reset, *Advances in Neural Information Processing Systems,* vol. 33, 2020.

[32] N. Mhaisen, A. A. Abdellatif, A. Mohamed, A. Erbad, and M Guizani, Optimal User-Edge Assignment in Hierarchical Federated Learning Based on Statistical Properties and Network Topology Constraints, *IEEE Transactions on Network Science and Engineering,* vol. 9, no. 1, pp.55-66, 2021.

[33] Y. Arjevani and O. Shamir, Communication complexity of distributed convex learning and optimization, *Advances in neural information processing systems*, vol. 28, 2015.

[34] A. Nedic, A. Olshevsky, and W. Shi, Achieving geometric convergence for distributed optimization over time-varying graphs, *SIAM Journal on Optimization*, vol. 27, no. 4, pp. 2597–2633, 2017.

[35] A. Koloskova, S. Stich, and M. Jaggi, Decentralized stochastic optimization and gossip algorithms with compressed communication, in *Proc. of International Conference on Machine Learning. PMLR*, 2019, pp. 3478–3487.

[36] H. Gao, M. T. Thai, and J. Wu, When decentralized optimization meets federated learning, *IEEE Network*, 2023.

[37] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konečnỳ, S. Mazzocchi, H. B. McMahan et al., Towards federated learning at scale: System design, *arXiv preprint arXiv:1902.01046*, 2019.

[38] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, Federated learning: Challenges, methods, and future directions, *IEEE Signal Processing Magazine,* vol. 37, no. 3, pp. 50–60, 2020.

[39] L. Zhu, H. Lin, Y. Lu, Y. Lin, and S. Han, Delayed gradient averaging: Tolerate the communication latency for federated learning, *Advances in Neural Information Processing Systems,* vol. 34, 2021.

[40] A. B. Ozyurt and W. O. Popoola, Lifi-based d2d communication in industrial IoT, *IEEE Systems Journal*, 2022.

[41] B. L. Bars, A. Bellet, M. Tommasi, E. Lavoie, and A. marie Kermarrec, Refined convergence and topology learning for decentralized optimization with heterogeneous data, in Proc. of The Workshop on Federated Learning: Recent Advances and New Challenges (in Conjunction with NeurIPS 2022), 2022. [Online]. Available: https://openreview.net/forum?id=WBm2z8jrgtP.

[42] Y. Hua, K. Miller, A. L. Bertozzi, C. Qian, and B. Wang, Efficient and reliable overlay networks for decentralized federated learning, *SIAM Journal on Applied Mathematics,* vol. 82, no. 4, pp. 1558–1586, 2022.

[43] S. Li, T. Zhou, X. Tian, and D. Tao, Learning to collaborate in decentralized learning of personalized models, in *Proc. of 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). IEEE*, 2022, pp. 9756–9765.

[44] J. Edmonds, Maximum matching and a polyhedron with 0, 1-vertices, *Journal of research of the National Bureau of Standards B,* vol. 69, no. 125-130, pp. 55–56, 1965

Yubin Duan is currently a research scientist at Meta Platform, Inc. He received his Ph. D. degree in the Department of Computer and Information Sciences at Temple University in 2022. He obtained his B.S. degree in Mathematics and Physics from the University of Electronic Science and Technology of China in 2017. Currently, he is working at Meta Platforms, Inc. His research focuses on scheduling algorithms for distributed systems and parallel computing. He has published nearly thirty papers in high-impact conferences and journals, such as IEEE ICDCS, ICPP, IWQoS, TMC, JPDC, etc. He has served as a program committee member for top international conferences such as the Web Conference, ACM WSDM, etc., and a reviewer for premier journals such as IEEE TMC, IoT journal, TNSE, etc.

Xiuqi Li is an Associate Professor of Instruction at Department of Computer and Information Sciences, Temple University, USA. She was a tenured Associate Professor at Department of Computer Science and Mathematics in University of North Carolina at Pembroke, USA. She worked as a senior instructor in Florida Atlantic University, USA, where she earned her Ph.D. in 2006. She is a PI/Co-PI of multiple grants. She was an NSF session chair and panelist. She served as a chair/co-chair/program committee member/referee for a number of conferences and journals. She holds thirtytwo peer-reviewed journal and conference papers. Her research interests include federated learning, blockchain, security, cloud computing, and computer science education. She is a member of ACM and IEEE.

Jie Wu is Laura H. Carnell Professor at Temple University and the Director of the Center for Networked Computing (CNC). He served as Chair of the 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, where he received his Ph.D. in 1989. His current research interests include mobile computing and wireless networks, routing protocols, network trust and security, distributed algorithms, applied machine learning, and cloud computing. Dr. Wu regularly published in scholarly journals, conference proceedings, and books. He serves on several editorial boards, including IEEE/ACM Transactions on Networking, IEEE Transactions on Service Computing, and Journal of Computer Science and Technology. Dr. Wu is/was general chair/co-chair for IEEE DCOSS'09, IEEE ICDCS'13, ICPP'16, IEEE CNS'16, WiOpt'21, ICDCN'22, IEEE IPDPS'23, and ACM MobiHoc'23 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. He is a Member of the Academia Europaea (MAE).