2022 IEEE 28th International Conference on Parallel and Distributed Systems (ICPADS) | 978-1-6654-7315-6/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICPADS56603.2022.00021

FedDGIC: Reliable and Efficient Asynchronous Federated Learning with Gradient Compensation

Zaipeng Xie*†, Junchen Jiang†, Ruifeng Chen†, Zhihao Qu†, and Hanxiang Liu‡
*Key Laboratory of Water Big Data Technology of Ministry of Water Resources, Hohai University, Nanjing, China
†College of Computer and Information, Hohai University, Nanjing, China
‡Department of Computer Science, University of Southern California, Los Angeles, USA
Email: {zaipengxie, junchenjiang, ruifeng.chen, quzhihao}@hhu.edu.cn, liuhanxi@usc.edu

***Abstract*—Asynchronous federated learning is a distributed machine learning paradigm that may alleviate the impact of straggler nodes and improve the efficiency of federated training. However, some nodes can become sluggish, and node dropout may frequently happen for various reasons, such as network connection constraints, energy deficits, and system faults. Consequently, the global model may deviate from the desired convergence direction and lead to suboptimal results. This work proposes an asynchronous federated learning framework, FedDGIC, to mitigate the impact of the node dropout problem. The proposed framework can improve training efficiency by utilizing a dynamic grouping algorithm with gradient compensation. Experiments are performed in a real federated learning environment using two datasets, i.e., MNIST and CIFAR-10. Compared with three state-of-the-art methods, the proposed FedDGIC can significantly improve training efficiency and provide reliable asynchronous federated learning.**

***Index Terms*—Asynchronous federated learning, node dropout, model convergence, dynamic grouping, gradient compensation**

I. Introduction

Federated learning (FL) is a machine learning paradigm [1] where many clients collaboratively train a model under the coordination of the parameter server while keeping the training data decentralized. However, straggler devices, especially the dropouts, can frequently appear due to many factors [2] such as device heterogeneity, network constraints, and energy deficit. Hence, a decrease in training efficiency may happen due to device dropouts [3]. Asynchronous federated learning [3] may alleviate the impact of straggler devices and improve the efficiency of federated training by eliminating the waiting for stragglers during model aggregation. However, the data among devices are often non-independently and identically distributed (non-IID) [1] in an actual FL implementation. The device dropout may lead to a discrepancy between the original data distribution and the remaining training data distribution. Consequently, the model convergence's direction can be biased from the original data distribution, resulting in suboptimal training results. In addition, the global model may lean toward representing the data from the living nodes, which may negatively impact the model fairness [4]. So far, most of the existing asynchronous federated learning methods [5]–[9] do not provide an active mechanism to compensate for the model convergence's direction caused by device dropouts. Hence, it is desired to devise an improved federated learning method for mitigating this issue.

This work introduces a novel asynchronous federated learning method, FedDGIC, that incorporates the dynamic grouping algorithm with a gradient compensation algorithm. The proposed method can improve the training efficiency in terms of the training time and model accuracy when the data distribution deviates due to device dropout in the asynchronous federated learning framework. This method first categorizes active working nodes into groups based on similarity matrices, and the nodes within each group are considered to exhibit a similar data distribution. When a node becomes a straggler and fails to respond, our proposed method can resort to the gradient compensation method to compensate for the lost gradient using its group members, thus mitigating the impact of device dropout and improving the training efficiency. The main contributions of our research are as follows:

- We propose a novel method, FedDGIC, to compensate for the lost gradient information caused by dropped devices. Our method can correct the convergence direction of the global model and thus significantly improve the training efficiency and provide a reliable and efficient asynchronous federated learning framework.
- Experiments are performed on the actual implementation of the asynchronous federated learning framework using the MNIST and CIFAR-10 datasets. A random number of dropped devices are simulated, and the impact of the varying non-IID data distribution is evaluated. Experimental results demonstrate that our proposed FedDGIC method outperforms the state-of-the-art algorithms in all experiment scenarios.

II. Related Work

Device heterogeneity exists in many federated learning frameworks, where working devices may be slow to respond and even drop out spontaneously during the training stage. Several methods [10]–[13] are proposed to mitigate the impact caused by stragglers in recent years. Li et al. [10] propose an optimization algorithm, FedProx, by adding a proximal term to help improve the stability of the federated learning process.

The corresponding author is Zaipeng Xie. This work is supported by The Belt and Road Special Foundation of the State Key Laboratory of Hydrology-Water Resources and Hydraulic Engineering under Grant 2021490811, the National Natural Science Foundation of China under Grant 62102131, and the Natural Science Foundation of Jiangsu Province under Grant BK20210361.

978-1-6654-7315-6/23/$31.00 ©2023 IEEE
DOI 10.1109/ICPADS56603.2022.00021

This algorithm allows each participating device to perform a variable amount of work to tackle device heterogeneity. However, since devices in federation learning may join or drop out frequently, it may still produce a deadlock by dropped devices that are supposed to participate in training. Chen et al. [11] propose a method to speed up the training process under communication constraints. The method can effectively reduce the communication rounds by selecting clients with a significant update norm. However, fairness may not be ensured for worker nodes with poor computational performance. Lai et al. [12] propose choosing the best available nodes at each round using the online exploration-exploitation strategy. Although this method can select high-performance worker nodes to avoid the delay caused by straggler or dropped nodes, low-performance nodes may be ignored in the training process. Chai et al. [13] propose a tier-based federated learning system that can group devices into multiple tiers based on their performance. Clients from the same tier can be selected at each round to mitigate the straggler problem. However, the individual client can become a straggler or dropout sporadically. Hence, it may require an additional scheme to reorganize the tiers and degrade the system performance.

The research of asynchronous federated learning [5]–[9] has attracted much attention in recent years. Asynchronous federated learning can improve training efficiency by eliminating node synchronizations. However, existing methods may not be able to mitigate the missing gradients [14] caused by dropout devices. It is desired to explore methods that can prevent suboptimal training results and ensure the model fairness [4]. Personalized federated learning (PFL) is a recent technique [15] that exploits personalized models for devices with data heterogeneity. Sattler et al. [16] propose a method to group asynchronous PFL clients based on their similarity of the gradient matrix using the model parameters, and the PFL training can be implemented in a cluster-based manner. Ghosh et al. [17] propose to cluster the working nodes using a K-means method. This method, however, is less flexible and requires specifying the number of groups by the users. Liu et al. [18] introduce a sparse representation of the CNNs to facilitate cluster-based PFL training, where the similarity of the representations is used. However, due to the data heterogeneity among nodes, it is challenging to set a reasonable number of groups for large-scale PFL systems. The hierarchical clustering method [19], [20] has been proposed to justify a reasonable number of groups. However, its efficacy as a solution for mitigating the dropped device problems is still unclear.

III. Proposed Method

A. Problem Analysis

Federated learning aims to obtain a model [1] that matches the overall data distribution, and the model loss, $F(W)$, can be given by

$$F(W) = \frac{1}{N} \sum_{i \in [N]} E_{z^i \sim D^i} f(W; z^i) \tag{1}$$

where W is the model weight and z^i is the sampling of the local dataset D^i on $node_i$, and $E_{z^i \sim D^i} f(W; z^i)$ denotes the loss expectation of the global model for the training data D^i on $node_i$. If k nodes are dropped, the loss function of federated learning can be written as

$$F'(W) = \frac{1}{N-k} \sum_{i \in [N-k]} E_{z^i \sim D^i} f(W; z^i). \tag{2}$$

Because the data in each node can be non-IID, the loss expectation in each node can be different, as described by

$$E_{z^i \sim D^i} f(W; z^i) \neq E_{z^j \sim D^j} f(W; z^j), \forall i \neq \forall j. \tag{3}$$

Hence, node dropouts may impose changes in the overall data distributions across the working nodes, resulting in an unexpected loss function for the global model, i.e., $F(W) \neq F'(W)$. Consequently, the optimization objective can be disfigured in this scenario.

This work proposes a gradient compensation-based asynchronous federated learning framework, FedDGIC, to establish a similarity matrix of the models. Based on this matrix, the working nodes are clustered into groups, and gradient compensation can be accomplished using the fellow worker nodes within the group.

B. Method Description

The general process of the FedDGIC framework is divided into two steps: (1) Dynamic grouping: the nodes are grouped dynamically based on the similarity matrix of models. (2) Gradient compensation: the system monitors the nodes' status and corrects the global model's convergence direction, maintaining a reliable model performance even under unexpected node dropouts. Before we start the method description, we summarize the important notations in Table I.

TABLE I
Notations

Notation	Description
n	Number of work nodes
$H(\cdot)$	Hierarchical clustering method
n_{cal}	Number of grouping times
$g(i)$	Group to which $node_i$ belongs
$V_{g(i)}$	Lastest model update version in group $g(i)$
V_i	Updated version submitted by $node_i$
T	The threshold to limit the number of groups
$W_{g(i)}$	A model of group $g(i)$
W_{new}	A model submitted from a node
α_{max}	Maximum update ratio of group model aggregation
α_{group}	Update ratio of group model aggregation
α_{global}	Update ratio of global model aggregation without gradient compensation
$N_{g(i)}$	Number of nodes in group $g(i)$
$D_{g(i)}$	Number of dropped nodes in group $g(i)$
r^c	Compensation ratio
α_{comp}	Update ratio of global model aggregation after gradient compensation
W_{global}	Global model
$P_{disorder}$	A hyperparameter to control the degree of data non-IID

1) Dynamic Grouping Algorithm: This algorithm consists of two steps: First, we employ the cosine similarity method [16] to generate the similarity matrix of models. At the same time, the algorithm minimizes the impact of the varying scales [20] of neural network layers. Second, we utilize a hierarchical clustering method to divide the working nodes into groups that can generate a tree structure, where the distance between groups can gradually increase. During each iteration, two groups with the smallest distance are clustered into one group, and the inter-group distance, $d(G_x, G_y)$, between group G_x and G_y, as given by

$$d(G_x, G_y) = \frac{1}{|G_x||G_y|}\left(\sum_{i\in G_x}\sum_{j\in G_y} d(i,j)\right), \tag{4}$$

where $i \in G_x, j \in G_y$, and the distance $d(i,j) = w_i \cdot w_j/(\|w_i\| \cdot \|w_j\|)$ is the cosine similarity between $node_i$ and $node_j$. The cosine distance [16] of the weights measures the difference in data distributions across clients and is not affected by the scaling effects. Since the distance between two merging groups increases proportionally at each clustering iteration, we propose that the hierarchical clustering method can dynamically calculate the number of groups. We define d^k as the minimum inter-group distance at the k-th aggregation and $d^k \geq d^{k-1}$. Therefore, The minimum inter-group distance at the n-th clustering iteration is d^n, where n is the number of nodes. The increment of d^k at each clustering iteration is set to be $(d^{n-1} - d^1)/(n-2)$. The hierarchical clustering algorithm, as illustrated in Fig.1, can stop the clustering and return the lists of the grouping result, given that the following inequation satisfies:

$$\frac{(d^k - d^{k-1}) \cdot (n-2)}{d^{n-1} - d^1} > p. \tag{5}$$

where p is a constraint pre-defined as the maximum ratio of the minimum inter-group distance increment ($p = 0.8$ in default).

The grouping process can require some extra computational cost on the parameter server due to the increased volume of gradient exchange. On the other hand, the one-shot grouping method [20], [21] may produce an inaccurate estimate, leading to a degraded global model. Hence, we develop a dynamic grouping tactic that can make a trade-off between the computational cost and the fitness of grouping by choosing the number of grouping operations, n_{cal}, based on the number of nodes n, where n_{cal} is defined as in (6). We use the average of the similarity matrix obtained after n_{cal} grouping iterations as the similarity matrix of the grouping result.

$$n_{cal} = \begin{cases} 2, & n \leq 8 \\ \lfloor \log_2 n - 1 \rfloor, & n > 8. \end{cases} \tag{6}$$

The dynamic grouping algorithm is described in Algorithm 1 and its time complexity is $O(n^3)$.

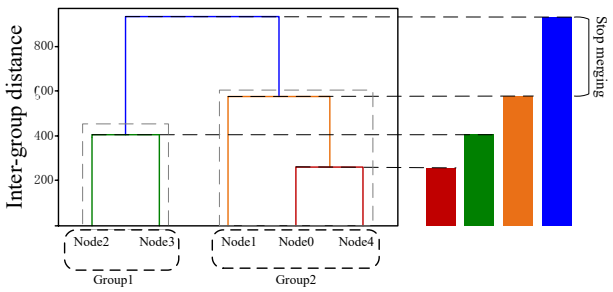

Fig. 1. An example of the dynamic grouping algorithm. The hierarchical clustering method stops grouping when the inter-group distance between Group1 and Group2 exceeds the threshold p.

Algorithm 1: Dynamic Grouping Algorithm

Input: Number of worker nodes n
Output: Grouping result r

Initial grouping times n_{cal} according to Eq.(6);
Initialize a list of similarity matrices S;
for i *in* $\{1, 2, \cdots, n_{cal}\}$ **do**
    Accept updated weights from all nodes;
    Calculate the similarity matrix $S[i]$ among nodes in round i;
end
Calculate mean of the similarity matrix $S_{avg} \leftarrow \frac{1}{n_{cal}} \sum_{i=1}^{n_{cal}} S[i]$;
Determine the number of groups according to Eq.(5);
Use the hierarchical clustering method to determine grouping result $r \leftarrow H(S_{avg})$;
return r;

2) Gradient Compensation Method: The exponential moving average algorithm [5] is usually employed to perform the model aggregation in asynchronous federated learning. The global model W is updated by aggregating the model W_{new} submitted by $node_i$, and the updated model W_{t+1} is given by

$$W_{t+1} \leftarrow (1-\alpha) \cdot W_t + \alpha \cdot W_{new}, \tag{7}$$

where hyperparameter $\alpha \in [0, 1)$ is the aggregate update ratio of the global model and α determines the global model's convergence. We propose a gradient compensation method to decide α adaptively. When a node submits its model update, it first updates the group model. Then, the global model is updated based on the group model using an α preselected based on the number of dropped nodes within the group. We utilize a version control algorithm to manage the stale updates that can produce reliable training results. The version control can be implemented via the update ratio of the group model aggregation, α_{group}, and the group model, $W_{g(i)}$, as given by

$$\begin{cases} \alpha_{group} = \dfrac{\alpha_{max}}{1 + \beta \cdot max\left(V_{g(i)} - V_i - N_{g(i)}, 0\right)} \\ W_{g(i)} \leftarrow \alpha_{group} \cdot W_{new} + (1 - \alpha_{group}) \cdot W_{g(i)} \end{cases} \tag{8}$$

where α_{max} is the maximum update ratio of the group model aggregation, $g(i)$ represents the group ID of $node_i$, $V_{g(i)}$ is the latest model version in group $g(i)$, V_i is the version of $node_i$, $N_{g(i)}$ is the number of members in group $g(i)$, and β is a positive parameter. Equation (8) indicates that the group model $W_{g(i)}$ is updated using the model from $node_i$ by the ratio α_{group}. As the difference between the group version $V_{g(i)}$ and the node version V_i grows, the ratio α_{group} decreases, leading to a reduced contribution by the model weights W_{new} provided by $node_i$ and reducing the impact of the stale updates.

After the group model updates, our algorithm can adaptively choose the aggregation update ratio, α_{global}, of the global model to compensate for the missing gradient. It utilizes a version monitoring algorithm to check the status of each working node within the group. In this algorithm, we define T as a timeout threshold. When $V_{g(i)} - V_i \geq T$, $node_i$ is considered a dropout node. Once the parameter server detects a node dropout, our proposed gradient compensation method can take the fellow worker nodes in the group and compensate for the gradient. The update ratio of the global model aggregation after gradient compensation α_{comp} can be described as given by

$$\alpha_{comp} = 1 - (1 - \alpha_{global})^{r_c} \tag{9}$$

where the hyperparameter α_{global} is the update ratio of the global model aggregation without gradient compensation, $r_c = N_{g(i)}/(N_{g(i)} - D_{g(i)})$ represents the compensation ratio, $N_{g(i)}$ is the number of nodes in group $g(i)$, $D_{g(i)}$ is the number of dropped nodes in group $g(i)$. The aggregated updates to the global model W_{global} can be described as given by

$$W_{global} \leftarrow \alpha_{comp} \cdot W_{g(i)} + (1 - \alpha_{comp}) \cdot W_{global}. \tag{10}$$

When the number of dropped nodes in the group increases, W_{global} is updated using the model from group members $W_{g(i)}$ by a modified contribution ratio α_{comp}.

For instance, consider a group of three nodes (i.e., $N_{g(i)} = 3$). If each node provides one update to the global model, the proportion of the original global model in the latest update is $(1 - \alpha_{global})^3$, the proportion of model weights provided by the group is $1 - (1 - \alpha_{global})^3$. When a node becomes a dropout (i.e., $D_{g(i)} = 1$), the compensation ratio r_c of its fellow members is $r_c = N_{g(i)}/\big(N_{g(i)} - D_{g(i)}\big) = 1.5$. If both of the remaining two nodes provide one update to the global model with $\alpha_{comp} = 1 - (1 - \alpha_{global})^{1.5}$, the proportion of the original global model in the latest update is

$$(1 - \alpha_{comp})^2 = (1 - \alpha_{global})^3. \tag{11}$$

And the proportion of model weights provided by the other two nodes is $1-(1-\alpha_{global})^3$. Hence, the group's contribution to the global model does not change in the presence of dropping nodes.

An example of the proposed gradient compensation mechanism is illustrated in Fig.2. In this example, our FedDGIC algorithm divides four working nodes into two groups, and the Nodes within each group update in a similar direction. Node1 and Node4 are trained to update the Group Model 1. Meanwhile, Node2 and Node3 are trained to update the Group Model 2. When Node3 drops, our FedDGIC algorithm can utilize Node2 as an interim agent to compensate for the missing gradient of Node3, hence maintaining the global direction of model convergence.

The pseudo-code for the proposed Gradient Compensation method is summarized in Algorithm 2.

Algorithm 2: Gradient Compensation Method

Input: The grouping result r, the number $N_{g(i)}$ of members in each group, α_{global}, α_{max}

Output: Global model W_{target}

```
Initialize model W_global = W_0;
Initialize the group model W_{g(i)} = W_0, i ∈ {1, ··· , k};
Initialize worker nodes version V_i = 0, i ∈ {1, ··· , n};
Initialize group version V_{g(i)} = 0, i ∈ {1, ··· , k};
Broadcast the initial parameter W_global to all nodes;
while f(W_global) − f(W_*) > ε do
    Receive the update weight W_new from node_i;
    Determine the group g(i) which node_i belongs to;
    Update the group model W_{g(i)} according to Eq.(8);
    V_{g(i)} ← V_{g(i)} + 1;
    Calculate the compensation ratio
      r_c ← N_{g(i)} / (N_{g(i)} − N_{d(g(i))});
    α_comp ← 1 − (1 − α_global)^{r_c};
    Update the global model according to Eq.(10);
    V_i ← V_{g(i)};
    Send W_global to node_i;
end
W_target ← W_global;
return W_target;
```

IV. Experiment

A. Experimental Setup

1) Environment Settings: The experiment is performed on an actual implementation of a federated learning framework, Parallel-SGD [22]. Our FL system consists of one parameter server and twenty clients. Each client holds the same number of training samples, and the number of dropout clients is varied during the training to simulate actual node dropout. We further quantify the degree of the data distribution to verify the effectiveness of our FedDGIC, i.e., we sort the training samples by their labels and then take out $P_{disorder}$ percent of the samples for each label. We follow the FedAvg method [1] to distribute data samples to twenty clients. We randomly assign an equal amount of the remaining samples to the twenty clients. When $P_{disorder}$ is 100, the result of the sample distribution is fully non-IID. When $P_{disorder}$ is 0, the distribution of samples at each client can be considered IID.

2) Datasets and Models: We use two popular datasets, MNIST [23], and CIFAR-10 [24]. The MNIST dataset contains 60,000 samples for training and 10,000 samples for testing, in which each sample is a 28×28 bilevel image. The

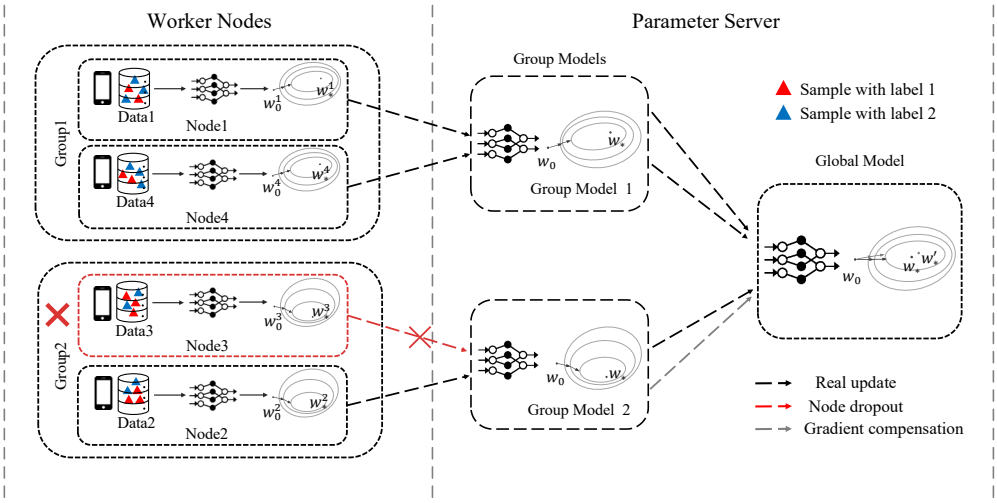

Fig. 2. An example of Gradient Compensation Method.

CIFAR-10 dataset contains 50,000 samples for training and 10,000 samples for testing, and each sample in CIFAR-10 is a 32×32 RGB image. Both datasets contain labels for ten categories. For the MNIST dataset, we train a simple neural network with three fully-connected layers. For the CIFAR-10 dataset, we employ a convolutional neural network that has four convolutional layers and two fully-connected layers. For simplicity, both networks do not utilize dropout layers and max-pooling layers.

3) Baseline Methods: We compare FedDGIC against three state-of-the-art asynchronous federated learning methods:

- FedAsync [6]: This method is an asynchronous FL method using the exponential moving average algorithm to update the global model.
- FedFG [19]: This method is implemented using the Federated Fixed Grouping method based on the idea in [19]. In this method, the number of groups is fixed, and the group model does not use the gradient compensation strategy when updating the global model.
- Sageflow [25]: This method can perform periodic global aggregation and groups clients based on their model's staleness. The representative models are aggregated among groups first and then contribute to generating a global model using the exponential moving average algorithm.

4) Hyperparameters: For both datasets, we use the SGD optimizer with a learning rate $\eta = 3 \times 10^{-3}$ for the four asynchronous federated learning methods. For the MNIST dataset, the batch size is set to 30, and the local epoch is 5. For the CIFAR-10 dataset, the batch size is 25, and the local epoch is 4. We adopt cross-entropy as the loss function. We set the global update ratio for all four methods as $\alpha_{global} = 0.5$. For FedDGIC and FedFG, we set the group update ratio as $\alpha_{group} = 0.8$. We perform the training for 100 epochs for MNIST and 150 epochs for CIFAR-10.

5) Evaluation Criteria: We adopt three evaluation metrics for our proposed method as follows:

- Reduction Ratio on Loss: This metric represents the average improvement in the accuracy of our method compared to the other methods on the test set in n epochs ($n = 100$ for MNIST and $n = 150$ for CIFAR-10).
- Improvement Ratio on Accuracy: This metric represents the average reduction in the loss of our FedDGIC method compared to the other methods on the same test dataset.
- Speed-up Ratio: We set a desired accuracy and loss value as the goal. Then, the required number of epochs to reach that goal is utilized to estimate the training speed-up $S(m1, m2, t)$ of our method as compared to the other methods given by

$$S(m1, m2, t) = \frac{Epochs(m1, t) - Epochs(m2, t)}{Epochs(m1, t)} \quad (12)$$

where $m1$ represents the proposed FedDGIC method, $m2$ represents the compared methods, including FedAsync, FedFG, and Sageflow, and the $Epochs(m, t)$ represents the number of epochs in time t.

We perform the experiments in two settings: In the first setting, we compare the performance by varying the number of dropped nodes under the same degree of non-IID data distribution across the working nodes. In the second setting, we compare each method's performance when the number of dropout nodes is the same while the degree of the non-IID data distribution varies.

B. Experimental Results

Before the experiments start, we allocate an equal number of training samples to the worker nodes. Then, after each training round, the global model is utilized to evaluate the model performance on the test dataset. Figure 3 describes the experimental results on MNIST when three, five, and seven nodes are dropped with $P_{disorder} = 40$. Our FedDGIC method is the best among all four methods, even though the

three compared methods update the global model using the exponential moving average algorithm. We observe that the performance difference among the three compared methods is trivial because of the gradient loss caused by the dropping node. Table II summarizes the experimental results, showing that our proposed FedDGIC method has an average speed-up ratio of 19.59%, 17.65% and 20.45% under the various number of node dropouts when $P_{disorder} = 40$.

TABLE II
IMPROVEMENT OF FEDDGIC COMPARED TO STATE-OF-THE-ART METHODS BY VARYING THE NUMBER OF DROPPED NODES ON MNIST

Method	Number of Dropped Nodes	Reduction Ratio on Loss	Improvement Ratio on Accuracy	Speed-up Ratio
FedAsync	3	5.57%	6.51%	16.86%
	5	6.43%	6.91%	17.26%
	7	6.87%	9.95%	24.65%
FedFG	3	3.89%	5.03%	14.29%
	5	5.07%	5.41%	18.15%
	7	5.20%	7.87%	20.50%
Sageflow	3	5.33%	7.54%	17.49%
	5	5.87%	7.91%	18.57%
	7	6.43%	9.64%	25.30%

Figure 4 illustrates the experiment results on CIFAR-10 when three, five, and seven dropout nodes are evaluated with $P_{disorder} = 50$. We observe that there is little difference between the four methods in the early stage of the training process, which may be caused by the underfitting of the model. The FedDGIC method, however, performs the best among all methods after 20 rounds of training. This is because FedAsync, FedFG, and Sageflow methods do not handle the dropout nodes well. Meanwhile, FedDGIC employs the gradient compensation and alleviates the model's deviation caused by dropout nodes. Table III shows the comparison results of various methods on CIFAR-10 when the number of dropped nodes is 3, 5, and 7. We conclude that the FedDGIC method has an average speed-up ratio of 21.82%, 16.22%, and 21.3% as compared to three other methods under the different number of node dropouts and $P_{disorder} = 50$.

TABLE III
IMPROVEMENT OF FEDDGIC COMPARED TO STATE-OF-THE-ART METHODS BY VARYING THE NUMBER OF DROPPED NODES ON CIFAR-10

Method	Number of Dropped Nodes	Reduction Ratio on Loss	Improvement Ratio on Accuracy	Speed-up Ratio
FedAsync	3	1.58%	4.74%	20.37%
	5	1.71%	5.48%	23.96%
	7	1.67%	5.20%	21.14%
FedFG	3	1.06%	3.86%	13.13%
	5	1.69%	5.27%	18.22%
	7	1.63%	4.83%	17.30%
Sageflow	3	1.29%	3.96%	20.37%
	5	1.67%	4.96%	22.57%
	7	1.66%	5.24%	20.96%

We also compare the performance of each method by varying the degree of non-IID data distribution when a certain number of nodes are dropped. Under the condition that five nodes are dropped, the comparison results on MNIST when $P_{disorder}$ is 20, 40, and 60 are presented in Table IV. The results show that our proposed FedDGIC method significantly outperforms the FedAsync, FedFG, and Sageflow in terms of loss and accuracy performance. For the MNIST dataset, the FedDGIC method has an average speed-up of 15.98%, 17.25%, and 17.44% under the different data distribution ($P_{disorder}$ = 20, 40, and 60), when the number of node dropout is 5. Table V shows the improvement of the FedDGIC method on CIFAR-10 when $P_{disorder}$ is 30, 50, and 70. The average speed-up is 23.55%, 18.65%, and 23.87%.

TABLE IV
IMPROVEMENT OF FEDDGIC COMPARED TO STATE-OF-THE-ART METHODS USING VARIOUS DATA DISTRIBUTIONS ON MNIST

Method	$P_{disorder}$	Reduction Ratio on Loss	Improvement Ratio on Accuracy	Speed-up Ratio
FedAsync	20	6.51%	7.13%	18.82%
	40	6.43%	6.91%	16.26%
	60	5.42%	4.12%	12.85%
FedFG	20	5.59%	6.93%	20.76%
	40	5.07%	5.41%	18.15%
	60	3.88%	4.10%	12.85%
Sageflow	20	7.50%	8.36%	20.50%
	40	5.87%	7.91%	18.57%
	60	4.84%	5.19%	13.27%

TABLE V
IMPROVEMENT OF FEDDGIC COMPARED TO STATE-OF-THE-ART METHODS USING VARIOUS DATA DISTRIBUTIONS ON CIFAR-10

Method	$P_{disorder}$	Reduction Ratio on Loss	Improvement Ratio on Accuracy	Speed-up Ratio
FedAsync	30	1.95%	7.49%	25.66%
	50	1.71%	5.48%	23.96%
	70	1.65%	5.29%	21.03%
FedFG	30	1.74%	5.74%	21.64%
	50	1.69%	5.27%	18.22%
	70	1.62%	5.13%	16.08%
Sageflow	30	1.96%	7.07%	23.64%
	50	1.67%	4.96%	22.57%
	70	2.10%	7.32%	25.40%

In summary, the experimental results demonstrate that the proposed FedDGIC method can mitigate the negative impact of dropout nodes, providing an improved loss and accuracy performance compared with three state-of-the-art methods. Although all methods are based on the exponential moving average algorithm, our proposed FedDGIC method can serve as a reliable and efficient asynchronous federated learning framework in actual implementations.

V. CONCLUSION

Devices participating in federated learning are often unreliable, where dropout nodes and straggler problems can significantly impact performance. Existing asynchronous federated learning methods can improve the efficiency of model aggregation. However, node dropout may lead to varying data distribution, causing the global model to deviate and reducing the accuracy and training efficiency of the global model. This work proposes an asynchronous federated learning framework based

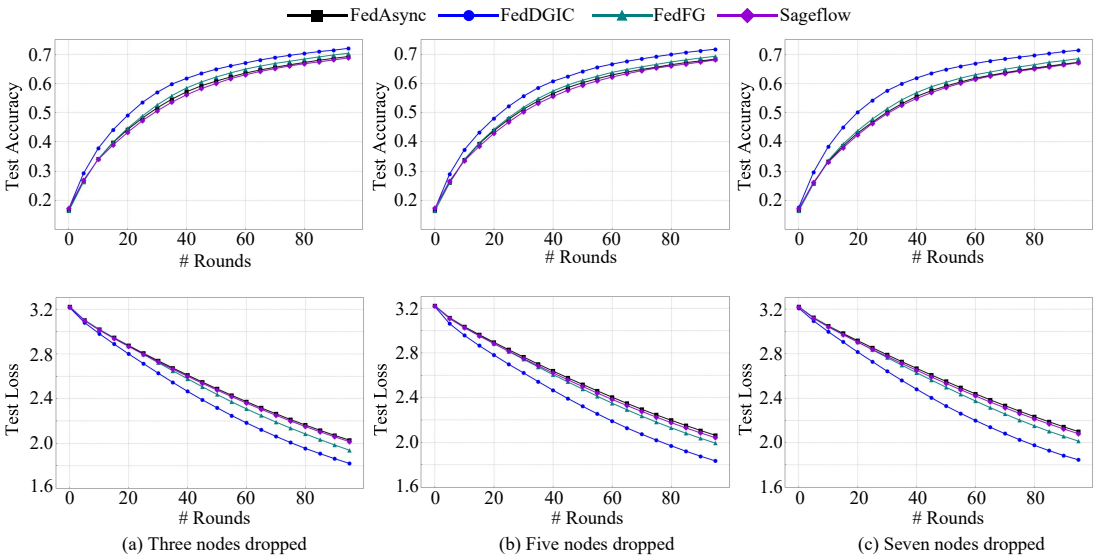

Fig. 3. Loss and accuracy of different methods on MNIST test set when three, five, and seven nodes are dropped.

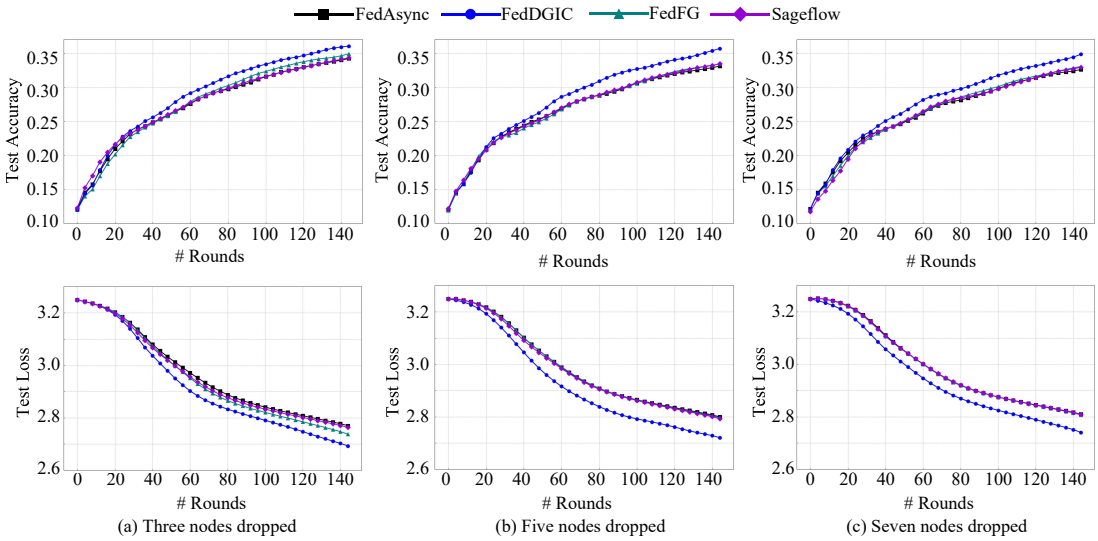

Fig. 4. Loss and accuracy of different methods on CIFAR-10 test set when three, five, and seven nodes are dropped.

on dynamic grouping and gradient compensation (FedDGIC) to address this problem. The proposed FedDGIC dynamically groups nodes based on the similarity matrix of models among nodes. It uses the nodes alive within the same group for gradient compensation when a node drops out. In order to verify the effectiveness of FedDGIC, we perform experiments with different numbers of dropped nodes in a certain degree of data non-IID distribution. We also conduct experiments with various degrees of non-IID distribution of data, but the number of dropped nodes is fixed. The proposed FedDGIC method is compared with some of the state-of-the-art methods, including FedAsync, FedFG, and Sageflow. The results demonstrate that FedDGIC outperforms the three methods on both MNIST and CIFAR-10 datasets in all experimental scenarios.

REFERENCES

[1] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and A. y. B. Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Conference on Artificial Intelligence and Statistics*. PMLR, 2017, pp. 1273–1282.

[2] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Processing Magazine*, vol. 37, pp. 50–60, 2020.

[3] C. Xu, Y. Qu, Y. Xiang, and L. Gao, "Asynchronous federated learning on heterogeneous devices: A survey," *arXiv preprint arXiv:2109.04269*, 2021.

[4] Z. Zhou, L. Chu, C. Liu, L. Wang, J. Pei, and Y. Zhang, "Towards fair federated learning," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. Association for Computing Machinery, 2021, p. 4100–4101.

[5] C. Xie, S. Koyejo, and I. Gupta, "Asynchronous federated optimization," *arXiv preprint arXiv:1903.03934*, 2019.
[6] Y. Li, S. Yang, X. Ren, and C. Zhao, "Asynchronous federated learning with differential privacy for edge intelligence," *arXiv preprint arXiv:1912.07902*, 2019.
[7] Y. Chen, Y. Ning, and H. Rangwala, "Asynchronous online federated learning for edge devices," *arXiv preprint arXiv:1911.02134*, 2019.
[8] Y. Chen, X. Sun, and Y. Jin, "Communication-efficient federated deep learning with layerwise asynchronous model update and temporally weighted aggregation," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 10, pp. 4229–4238, 2020.
[9] Z. Chai, Y. Chen, L. Zhao, Y. Cheng, and H. Rangwala, "FedAT: A communication-efficient federated learning method with asynchronous tiers under non-iid data," *arXiv preprint arXiv:2010.05958*, 2020.
[10] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," in *Proceedings of Machine Learning and Systems (MLSys)*, vol. 2, 2020, pp. 429–450.
[11] W. Chen, S. Horvath, and P. Richtarik, "Optimal client sampling for federated learning," *arXiv preprint arXiv:2010.13723*, 2020.
[12] F. Lai, X. Zhu, H. V. Madhyastha, and M. Chowdhury, "Oort: Efficient federated learning via guided participant selection," in *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2021, pp. 19–35.
[13] Z. Chai, A. Ali, S. Zawad, S. Truex, A. Anwar, N. Baracaldo, Y. Zhou, H. Ludwig, F. Yan, and Y. Cheng, "TiFL: A tier-based federated learning system," in *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing (HPDC)*, 2020, p. 125–136.
[14] S. Dhakal, S. Prakash, Y. Yona, S. Talwar, and N. Himayat, "Coded federated learning," in *2019 IEEE Globecom Workshops*, 2019, pp. 1–6.
[15] V. Kulkarni, M. Kulkarni, and A. Pant, "Survey of personalization techniques for federated learning," *2020 Fourth World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4)*, pp. 794–797, 2020.
[16] F. Sattler, K.-R. Müller, and W. Samek, "Clustered federated learning: Model-agnostic distributed multitask optimization under privacy constraints," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, pp. 3710–3722, 2020.
[17] A. Ghosh, J. Chung, D. Yin, and K. Ramchandran, "An efficient framework for clustered federated learning," *Annual Conference on Neural Information Processing Systems (NeurIPS)*, vol. 33, pp. 19 586–19 597, 2020.
[18] B. Liu, Y. Guo, and X. Chen, "PFA: Privacy-preserving federated adaptation for effective model personalization," in *Proceedings of the Web Conference*, 2021, pp. 923–934.
[19] C. Briggs, Z. Fan, and P. Andras, "Federated learning with hierarchical clustering of local updates to improve training on non-iid data," in *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2020, pp. 1–9.
[20] W. Y. B. Lim, J. S. Ng, Z. Xiong, J. Jin, Y. Zhang, D. Niyato, C. Leung, and C. Miao, "Decentralized edge intelligence: A dynamic resource allocation framework for hierarchical federated learning," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 3, pp. 536–550, 2022.
[21] A. Ghosh, J. Hong, D. Yin, and K. Ramchandran, "Robust federated learning in a heterogeneous environment," *arXiv preprint arXiv:1906.06629*, 2019.
[22] EngineerDDP, "Parallel-SGD," https://github.com/EngineerDDP/Parallel-SGD, 2021, [Online; accessed 20-July-2022].
[23] L. Deng, "The MNIST database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.
[24] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," *Technical report, University of Toronto.*, 2009.
[25] W. J. Park, D.-J. Han, M. Choi, and J. Moon, "Sageflow: Robust federated learning against both stragglers and adversaries," *Annual Conference on Neural Information Processing Systems (NeurIPS)*, vol. 34, pp. 840–851, 2021.