Glint: Decentralized Federated Graph Learning with Traffic Throttling and Flow Scheduling

Tao Liu*, Peng Li*, and Yu Gu†
*The University of Aizu, Japan. Email: {d8212107, pengli}@u-aizu.ac.jp
†Hefei University of Technology, China. Email: yugu.bruce@ieee.org

2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS) | 978-1-6654-1494-4/21/$31.00 ©2021 IEEE | DOI: 10.1109/IWQOS52092.2021.9521331

***Abstract*—Federated learning has been proposed as a promising distributed machine learning paradigm with strong privacy protection on training data. Existing work mainly focuses on training convolutional neural network (CNN) models good at learning on image/voice data. However, many applications generate graph data and graph learning cannot be efficiently supported by existing federated learning techniques. In this paper, we study federated graph learning (FGL) under the cross-silo setting where several servers are connected by a wide-area network, with the objective of improving the Quality-of-Service (QoS) of graph learning tasks. We find that communication becomes the main system bottleneck because of frequent information exchanges among federated severs and limited network bandwidth. To conquer this challenge, we design Glint, a decentralized federated graph learning system with two novel designs: network traffic throttling and priority-based flows scheduling. To evaluate the effectiveness of Glint, we conduct both experiments on a testbed and trace-driven simulations. The results show that Glint can significantly outperform existing federated learning solutions.**

I. Introduction

Federated learning has shown great promise in enabling joint machine learning among distributed parties while preserving their data privacy [1]–[4]. There are a growing amount of research efforts on federated learning, but they all focus on convolution neural network (CNN) models that show superior learning accuracy on recognizing images and voices. However, many applications generate graph data (e.g., social graphs and protein structures) consisting of vertices and edges, which cannot be efficiently handled by CNN.

In this paper, we study federated graph learning (FGL) under a cross-silo setting including a set of servers maintained by different institutions (e.g., banks, hospitals, and universities). These servers may reside in public or private clouds, and they are connected by a wide-area network (WAN). Each server maintains a graph with edge connections to the graphs held by other servers. Specifically, we consider the graph convolutional network (GCN) model [5], which has been proposed to deal with graph learning by novel graph convolution operations. Similar to CNN's convolution operation that filters a small set of neighboring pixels, a graph convolution operation filters the features of neighboring vertices.

Existing work has shown that communication becomes the main bottleneck of the distributed machine learning, which seriously affects the Quality-of-Service (QoS) of learning tasks [1], [6], [7]. The communication challenge faced by FGL is more severe than traditional CNN-oriented federated learning for two reasons. First, FGL incurs more frequent information exchanges among servers due to its unique characteristics. In CNN-oriented federated learning, trainable model parameters are synchronized among participants only at the end of each training round. However, in addition to trainable parameter synchronization, servers in FGL need to exchange vertex embeddings in every graph convolutional layer during each training round. Our experimental results show that vertex embedding size is several times larger than that of trainable model parameters. Second, FGL relies on a wide-area network for communication. Since many applications share this network, the bandwidth allocated to FGL is very limited. Besides, the available network bandwidth for federated graph learning is usually unstable because of unpredictable network congestion.

There exist many research efforts on addressing expensive communication cost in similar scenarios with FGL. Unfortunately, they provide limited improvement or cannot be directly ported to FGL. The most related work is traditional CNN-oriented federated learning. Although various methods [8]–[11] have been proposed to optimize the communication, they differ from our problem studied in this paper. For example, many work focuses on scenarios where devices are connected by wireless networks and studies the tradeoff between learning speed and energy efficiency [8], [11]. There are some preliminary studies, e.g., [12], about federated learning under the cross-silo setting, but they do not exploit GCN's characteristics and leave a large optimization space unexplored. We also notice that some distributed systems, e.g., distributed machine learning and MapReduce, running across data centers also have communication challenges similar to ours, and they are conquered by joint data movement and task allocation schemes [13]–[15]. However, FGL does not allow graph data movement due to privacy protection. Moreover, FGL shows a different computational graph and traffic patterns from existing work. There also exist some general flow scheduling approaches, e.g., conventional first-in-first-out, shortest-flow-first and fair-sharing [16]–[18], but they bring limited improvement. Some more complex scheduling approaches claim a better performance [19]–[21]. However, they cannot fully exploit unique characteristics of FGL.

In this paper, we propose Glint, a distributed federated graph learning system with a series of novel designs to address the communication challenge. Instead of synchronizing via a parameter server, Glint lets training servers directly share

978-0-7381-3207-5/21/$31.00 ©2021 IEEE

978-1-6654-1494-4/21/$31.00 ©2021 IEEE

vertex embeddings and trainable parameters in a peer-to-peer manner. Furthermore, in order to protect vertex embeddings, Glint adopts cross-server graph convolutional operations that compress vertex embeddings before sharing, so that remote servers cannot recover the original embeddings. Glint conquers the communication challenge with two novel designs. First, it reduces the network traffic by eliminating the transmission of unimportant embeddings, which is also referred to as traffic throttling. Specifically, Glint lets each node first quickly evaluate its neighbors' contribution based on marginal loss, and then collect data from a subset of important neighbors. The second novel design is a priority-based dynamic flow scheduling strategy. Glint monitors the training speed of federated servers and dynamically assigns different priority levels to network flows according to their training stages. The main contributions of this paper are summarized as follows.

1) We propose a decentralized, federated graph learning system called Glint to enable collaborative training of GCN models among distributed servers without leakage of their local graph data. We have identified that communication is the main bottleneck of the decentralized federated graph learning system.
2) We analyze the graph training characteristics and evaluate the contributions of training participants. We design a traffic throttling module based on these observations. We further formulate the flow scheduling problem and propose a heuristic priority-based dynamic flow scheduling algorithm.
3) We implement a prototype of Glint and evaluate its performance using a 20-server setting. Extensive experimental results show that Glint can significantly outperform existing work.

The rest of this paper is organized as follows. In Section II, we first give some background about federated learning and graph convolutional networks. Some preliminary experimental results that motivate this work are also presented. Then we present the system design in Section III, which mainly contains the traffic throttling module and the flow scheduling module. The experimental results are presented in section IV, followed by related work in Section V. Finally, we conclude our work in Section VI.

II. Background and Motivation

A. Federated Learning

In many machine learning tasks, training data are distributed at different computing infrastructures that do not allow direct data sharing due to privacy concerns. Federated learning has been proposed to enable collaborative training among multiple servers without leaking any raw data. Its basic idea is to let servers share model parameters, instead of training data. A typical parameter synchronization scheme widely adopted by federated learning is the Parameter Server (PS) [2]. Specifically, a federated learning system consists of a parameter server and a set of computing servers. The training process contains multiple rounds. In each round, every server uses its local dataset to train a model. After local training, servers send their local models to the parameter server who then creates a new global model for the next-round training. Despite the simplicity, PS-based federated learning suffers from the global synchronization bottleneck at the parameter server.

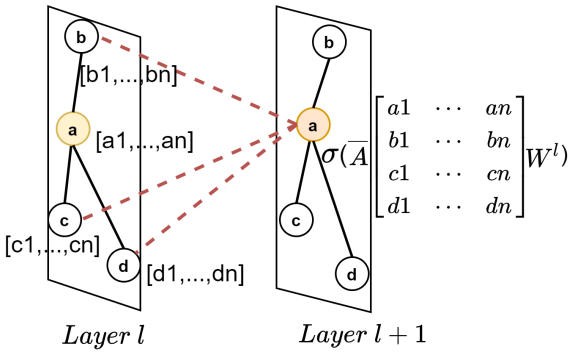


Fig. 1: Convolution operation on the graph.

This weakness motivates the study of decentralized federated learning, where servers directly share their model parameters in a peer-to-peer manner. To further release the communication burden, GossipGraD [22] uses a gossip protocol to deal with model aggregation. Combo [23] extends it with the segment operation to achieve higher utilization of network bandwidth. However, these decentralized schemes only focus on local models' aggregation stage, without exploiting the unique traffic patterns of FGL.

B. Graph Convolutional Network

Given a graph where each vertex is associated with a feature vector, graph convolutional network (GCN) aims to transfer vertex feature vectors into compressed ones, which are also called embeddings, by exploiting both graph structure and vertex features. GCN stacks multiple layers, and each layer has the same structure with the original graph. The graph convolution operation is defined as aggregating node embeddings of neighboring nodes, as shown in Fig. 1. We use A to denote the graph adjacent matrix, and H^l to denote the matrix of node embeddings in the l-th layer. The propagation rule of the GCN can be formally expressed by

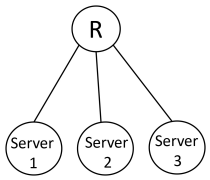
$$z^{l+1} = \overline{A} H^l W^l, \qquad H^{l+1} = \sigma(z^{l+1}), \tag{1}$$

where $\overline{A} = D^{-\frac{1}{2}}(A+I)D^{-\frac{1}{2}}$, W^l denotes trainable feature weights, D is the degree matrix and σ is the activation function.

Given a set V of nodes with labels, the GCN training goal is to minimize the loss function:

$$\mathcal{L} = \frac{1}{|\mathcal{V}|} \sum_{j \in \mathcal{V}} \mathcal{F}(y_j, \hat{y}_j), \tag{2}$$

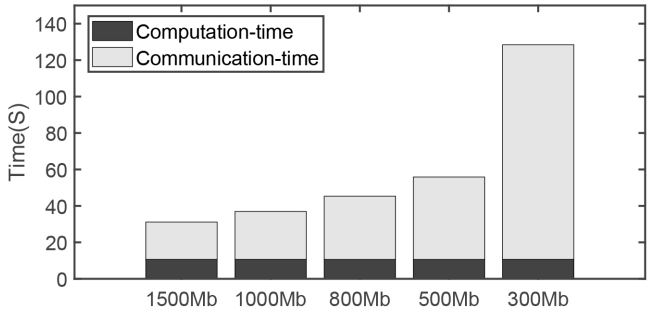
where y_j and $\hat{y}_j$ denote the true label and the predicted one, respectively. The loss function $\mathcal{F}$ usually uses the cross-entropy.



(a) Testbed topology.

		Server 1	Server 2	Server 3
Server 1	Embedding	0	443 MB	440 MB
	Local model	0	0.25 MB	0.25 MB
Server 2	Embedding	416 MB	0	431 MB
	Local model	0.25 MB	0	0.25 MB
Server 3	Embedding	408 MB	427 MB	0
	Local model	0.25 MB	0.25 MB	0

(b) The amount of transmitted data.



(c) The training time breakdown under different network bandwidth.

Fig. 2: Example of cooperate training among three servers.

C. Communication Bottleneck in Federated Graph Training

It has been well recognized that communication is the main bottleneck of distributed machine learning, especially in the WAN environment [12], [24], [25]. To better understand how communication affects FGL, we conduct some preliminary empirical studies on a 3-server cluster, where each server is equipped with Inter i7-10700 CPU, 16GB memory, and Geforce RTX 2080 GPU. As shown in Fig. 2(a), three servers are connected by a switch, and network bandwidths of all links are set to 1500Mbps. We divide the Reddit graph dataset, which has been widely used in graph learning research, into three parts stored by these servers, respectively. The Reddit dataset contains 232965 vertices, and each vertex has 602 features. Each server creats a two-layer GCN based on its local graph (including connected vertexes hold by others) and exchanges trainable parameters after each training round. The amount of data exchange averaged over training rounds are shown in Table 2(b). The size of trainable parameters is about only 0.25Mb, while the exchanged embedding size is over 400Mb.

We then change the network bandwidth and study how it affects training time. As shown in Fig. 2(c), we can see that the communication time increases dramatically as bandwidth decreases, while the computation time is almost the same. Other graph data in practice may involve even billions of nodes and edges [26], which makes the FGL more challenging.

III. System Design

A. Overview

We consider a typical federated setting consisting of a set C of distributed servers connected by a WAN with limited bandwidth. Each server $c_i \in C$ holds a local training dataset expressed as a graph $\mathcal{G}_i = (V_i, E_i)$, where V_i and E_i denote the vertex set and edge set, respectively. Every vertex $v \in V_i$ is associated with a feature vector x_v that cannot be exposed to other servers. Note that there exist some graph edges across servers. Each server is aware of the existence of connected vertices at other servers but cannot access their vertex feature vectors. For example, medical records of several hospitals can be modeled as a graph, where graph vertices represent patients and vertex features can be names, ages, and symptoms [27]. The interconnections among patients mean they have the same symptoms.

Each server constructs a GCN of L layers based on its local graph and trains this GCN. A unique feature of GCN training is to exploit graph structure information by graph convolution operations. Glint follows this idea to exploit both intra-server and inter-server graph structures. However, directly sharing vertex embeddings among servers leads to privacy exposure. To hide vertex embeddings in cross-server graph convolution operations, Glint lets each server aggregates only internal vertexes in the first GCN layer. The external vertexes are considered for aggregation from the second layer. Specifically, in the second layer and above, Glint lets each server multiply vertex embeddings with its local weights and then shares the results with others. After linear and non-linear transformations (e.g., activation function), the remote servers cannot recover the original vertex features even they know the weight matrix.

In many existing federated learning systems, all training servers send their local models to a parameter server that creates an updated global model for the next-round training. In contrast, as shown in Fig. 3, Glint adopts a decentralized coordination scheme that enables direct information sharing among training servers, eliminating the global barrier of the centralized parameter server. Specifically, at the beginning of each training round, each server collects weights from others and then averages them to get initial weights used in the current-round training. When a server conducts the convolutional graph operation, it requests vertex embeddings from other servers holding neighboring vertices. After local training, each server shares its updated weights by directly sending them to other servers. To ease the presentation, we refer to the training operation and its successive network flows as a training stage. Due to the difference of network bandwidth and latency, some servers who quickly collect required input information can start the next-stage computation earlier, avoiding waiting for slower servers in the same stage.

The above strategies are essential cornerstones of Glint system design. To further accelerate the training process, Glint has two novel designs called traffic throttling and flow scheduling, which aims to deal with the communication bottleneck in distributed graph learning. Their details are presented as follows.

B. Traffic Throttling

In Glint, each server collects feature embeddings from its neighbors. We have an important observation that embeddings from these neighbors have different contributions to the training process. To have a better understanding, we conduct

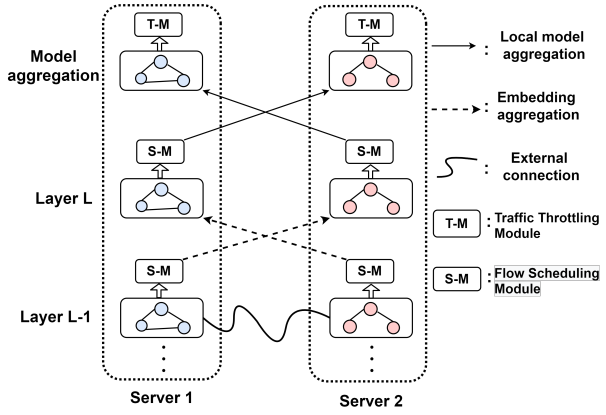


Fig. 3: Decentralized federated graph learning architectures.

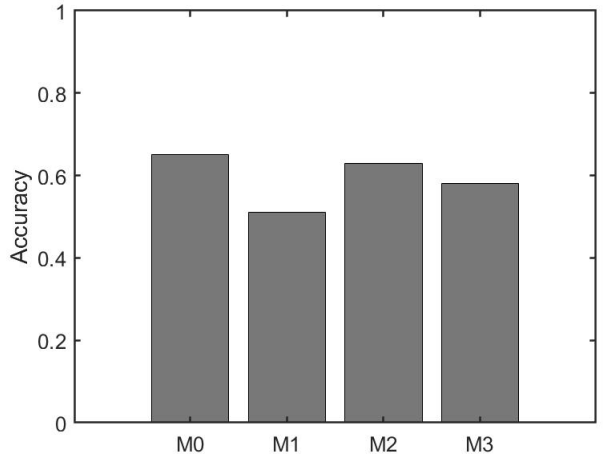


Fig. 4: Various contributions from neighbor servers.

Algorithm 1: Experience-based Traffic Throttling.

1: C is the set of all servers.
2: N_i contains the neighbors of server c_i.
3: Epo is the total training epochs.
4: # $The\ contribution\ evaluation\ phase$:
5: **for** server $c_i \in C$ in parallel **do**
6: **for** $r = 1, 2, \ldots, k$ **do**
7: Train a local model $M^r_{N_i}$ based on embeddings received from all neighbors;
8: For each neighbor j, c_i trains(or forward propagation) another model $M^r_{N_i-j}$.
9: **end for**
10: c_i calculates neighbors' embedding contributions scores with (3).
11: Replace embedding information with weight information and repeat 5-8 to calculate weight contributions scores.
12: **end for**
13: # $The\ neighbors\ selection\ phase$:
14: **for** each server $c_i \in C$ in parallel **do**
15: Sorted neighbors contribution in a non-increasing order.
16: Select the neighbors with larger contributions until their sum exceeds ψ.
17: **for** $r = k+1, k+2, \ldots, Epo$ **do**
18: Collect embedding and weight information from the selected neighbors.
19: Estimate the unselected neighbors' embedding and model weight using (4) and (5), respectively.
20: Conduct the local GCN training based on collected and estimated information.
21: **end for**
22: **end for**

experiments to show this phenomenon. Specifically, we divide the commonly used Cora dataset (containing 2708 nodes and 10556 edges) into four parts stored by different servers. The server 1 trains a two-layer GCN model (M0) that contains the complete embedding information from all neighbor servers. For comparison, it also trains 3 other models by eliminating some embedding information from neighbors. In each model training, a single neighbor is eliminated. The resulting models are referred to as M1, M2, and M3, respectively, and their accuracy is shown in Fig. 4. We can find that ignoring the embedding from different servers brings different levels of accuracy degradation, which means they have different contributions to the graph model training.

The basic idea of traffic throttling is to evaluate neighbors' contributions and to eliminate transmissions of unimportant data. Traffic throttling mainly contains two phases: contribution evaluation and neighbor selection, whose pseudo-codes are shown in Algorithm 1.

1) Contribution Evaluation: We follow the idea in [28] to quantify the contribution of shared embeddings. Note that the network flows associated with model sharing can be throttled in a similar way. Specifically, in each epoch r, server c_i first receives embeddings from all neighbors in set N_i and conducts training to generate a local model $M^r_{N_i}$ (Line 7). Then, for each neighbor $c_j \in N_i$, sever c_i also trains a model $M^r_{(N_i-j)}$ (Line 8), where $N_i - j$ means the neighbors set excluding the server c_j. The embedding contributions of c_j thus can be calculated as:

$$S_{j,i} = \frac{\sum_{r=1}^{k}[L(M^r_{N_i}) - L(M^r_{N_i-j})]}{\sum_{j\in N(i)}\sum_{r=1}^{k}[L(M^r_{N_i}) - L(M^r_{N_i-j})]}, \qquad (3)$$

where $L(M_{N_i})$ denotes the training loss of model M_{N_i}. Both models $M^r_{N_i-j}$ and $M^r_{N_i}$ are trained based on the $M^{r-1}_{N_i}$ in the previous epoch. Note that k is the number of epochs used for contribution evaluation. Since (3) does not involve the process of complete back propagation and verification, it can be quickly computed. We claim that a few epochs are enough to evaluate the contributions, which will be verified in Section IV.

2) Neighbor Selection: Based on contribution evaluation results, the neighbor selection phase selects neighbors with

high contributions and estimates the training input accordingly.

We let each server sort its neighbors according to their contributions scores, and select the ones with larger contributions until their sum scores exceeds the threshold ψ (Line 16). Our experimental results in Section IV show that an appropriate threshold can speed up the training process since less data are transmitted, while the final training accuracy has minor reduction. After neighbor selection, we estimate the embedding used for training input according to the collected ones from selected neighbors. For each server c_i with α_i flows, it extracts the embedding information contained in these flows and updates the local adjacent matrix $\overline{A}_{uv}$ as follows.

$$\overline{A}_{uv}^{(l-1)} = \begin{cases} \frac{|N(u)|}{|N'(u)|}\overline{A}_{uv}, & \text{if } v \in N(u), \\ 0, & \text{otherwise}, \end{cases} \tag{4}$$

where $N(u)$ denotes the set of neighbors of vertex u in the original graph, and $N'(u)$ denotes the set of neighbors obtained with α_i flows. Note that both $N(u)$ and $N'(u)$ contains internal neighbors and external neighbors maintained by other servers. For weight sharing stages, we estimate the initial weights with β_i flows, whose senders are included in set $\mathbb{C}_i$:

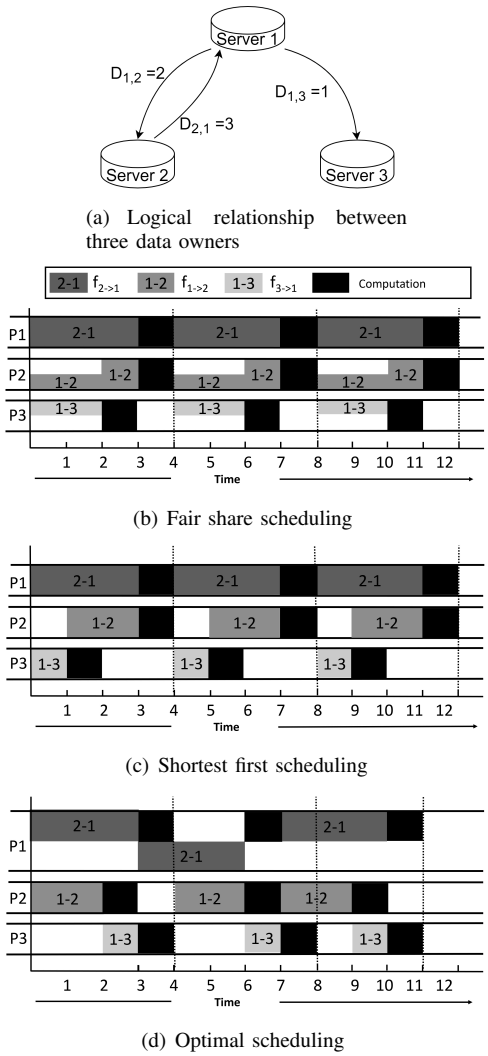
$$\omega_i = \sum_{c_j \in \mathbb{C}_i} \frac{|V_j| w_j}{\sum_{c_j \in \mathbb{C}_i} |V_j|}, \tag{5}$$

where V_j denotes the set of labeled vertices hold by server c_j.

C. Priority-based Dynamic Flow Scheduling

The transmissions of shared embeddings and model parameters can be modeled as network flows. It has been well recognized that flow scheduling is significant for communication efficiency [8]. We use the 3-server example in Fig. 2(a) to show the performance of different flow scheduling policies. Three servers are connected by a switch with limited bandwidth. Each network link's capacity is one data unit per second, and it works in the full-duplex mode. These servers collaboratively train a GCN model with three layers. The data sizes, denoted by $D_{i,j}$, of shared vertex embeddings among servers in each training round are illustrated in Fig. 5(a). We suppose all servers need one time unit to complete computation after receiving needed data for each layer.

We first consider the fair-sharing flow scheduling policy that flows fairly share the link bandwidth [18]. The scheduling results within a training round are shown in Fig. 5(b). The flows $f_{1\to2}$ and $f_{1\to3}$ need to share the bandwidth of link P_{1-R} during the first two time slots. Although $f_{1\to3}$ finishes earlier in the current stage due to less traffic, its transmission in the next stage can start only when server 1 completes its first layer's computation. As a result, three servers need 12 time slots to complete a training round. Another widely adopted flow scheduling policy is the shortest-flow-first that always schedules the flows with the minimum sizes [17]. As shown in Fig. 5(c), flow $f_{1\to2}$ competes with $f_{1\to3}$ on a link, and $f_{1\to3}$ is always scheduled first due to its smaller size. The completion time of a training round is still 12 time slots.



(a) Logical relationship between three data owners

(b) Fair share scheduling

(c) Shortest first scheduling

(d) Optimal scheduling

Fig. 5: Different scheduling schemes comparison.

After carefully examining the learning process, we find a better scheduling result as shown in Fig. 5(d). In contrast to shortest-flow-first, we schedule $f_{1\to2}$ first so that the communication of its counterpart in the next stage and computation of the first layer of server 1 can overlap in the 3-rd and 4-th time slots. Thanks to the benefit from communication-computation overlap, the total completion time is 11 time slots, less than the above two scheduling policies.

Although it is still challenging to find the best scheduling solution, the example has demonstrated the opportunities of training acceleration via flow scheduling. When we revisit the federated graph learning details in the example, we find that the learning process gives us two insights that are significant for the data flow scheduling. First, the flows belonging to the earlier stages should be transmitted first because they can enable the received server to finish the current stage computation faster. Second, we need to consider computation time of the servers after receiving flows. If a server needs longer time for training, its incoming flows should have a higher priority to

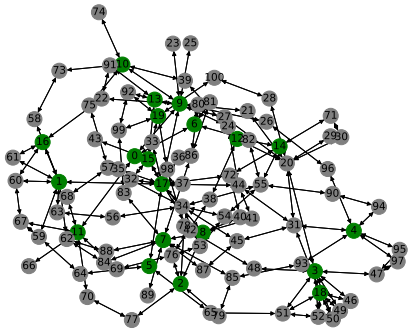


Fig. 6: The network topology used in performance evaluation.

avoid being the bottleneck. As the example shown in Fig. 5, for the first layer of the shortest first scheduling, both servers 1 and 2 need 4 time slots, longer than server 3. If the flows $f_{2\to1}$ and $f_{1\to2}$ are granted higher priorities, we can get the optimal result as shown in Fig. 5(d).

To solve the challenge, we propose a dynamic priority-based flow scheduling algorithm based on the above-mentioned insights. The pseudo-codes are shown in Algorithm 2. The system provides P priority levels that range from 0 to $P-1$, where 0 means the highest priority level. In each training epoch r, server c_i maintains a set of flows F_i to be transmitted to other servers. It also collects the information of corresponding receivers' current training stage and previous training stage time cost of the flows in F_i and maintains them in set St_i and T_i, respectively. We let Len_i denotes the size of St_i (Line 4). We first sort the flows in F_i according to their receivers' training stages in St_i. Specifically, the flows are ordered from the smallest training stage value to the largest one, and the smaller one means a larger priority (Line 6). For the flows whose receivers are in the same training stage, we describe them as a sub-flow set and further sort the flows in each sub-flow set according to the time cost information in T_i (Line 8). The sorted F_i is denoted as F_i'. For a flow f_j in F_i', we calculate its priority P_j as follows:

$$ind = \lceil Len_i/P \rceil, \tag{6}$$

$$P_j = \lceil j/ind \rceil. \tag{7}$$

Then the flow f_j is put into the queue of the P_j priority level. The calculation starts from $j = 1$, which guarantees that the training stage information and time cost information is fully taken into account. Note that servers have different training time costs that may changes across training stages, due to various computation capability and dynamic network bandwidth. Our algorithm considers both changing factors and lets each server dynamically calculate the priority levels at every training epoch.

Algorithm 2: Priority-based flow scheduling.

Input:
The set of output flows F_i for a server i.
Output:
The priority of flows in F_i.
1: There are P priority levels.
2: C is the server set.
3: **for** server $c_i \in C$ in parallel **do**
4: When c_i starts its $r-th$ training epoch, c_i collects stage set St_i, time set T_i and records their size Len_i.
5: # $The\ first-time\ range$:
6: c_i sorts flows in F_i according to the training stage information in St_i from smallest to largest.
7: # $The\ second-time\ range$:
8: c_i further sorts flows whose receivers are in the same training stage according to T_i from highest to lowest, and generates F_i'.
9: **for** j = 1,2,...,Len_i **do**
10: Get a flow f_j in F_i'.
11: $ind = \lceil Len_i/P \rceil$
12: $P_j = \lceil j/ind \rceil$
13: Put flow f_j into the candidate queue of the P_j priority level.
14: **end for**
15: **end for**

IV. Experiments and Evaluation

A. Experimental Settings

We implement Glint's prototype based on PyTorch and a python graph learning package, named Deep graph Library (DGL) [29]. The hardware includes Inter i7-10700 CPU, 16GB memory, and Geforce RTX 2080 GPU. We choose four widely used graph datasets: Cora, PubMed, Coauthor, and Reddit, whose details are summarized in Table I. We extract a network topology from a real peer-to-peer network [30], which contains 20 servers and 81 routers, as shown in Fig. 6. We set the network bandwidth randomly from 5Mbps to 500Mbps. Note that network bandwidth may fluctuate over time, and network congestion can also happen. Thus we randomly set the bandwidth values of all links according to a Gaussian distribution, where the standard deviation is one-tenth of the mean. The threshold k, which represents the number of epochs for contribution evaluation, is set to 10. The threshold ψ, which denotes the sum of transmitted flows' contribution score, is set to 0.9. Each server randomly chooses a subgraph from a given dataset and trains a two-layer GCN model with the ADAM optimizer.

We collect the real data-trace for performance evaluation, and we compare Glint with the following two baselines.

1) Centralized federated average graph learning (CFA): We extend FdeAvg [2], which has been widely used by many federated learning schemes, to implement a centralized federated average graph learning (CFA) scheme by adapting the same priority-based flow scheduling strategy in Glint for the embedding exchanging part.
2) Decentralized gossip-based federated graph learning (DGB): Combo [23] uses a segment operation and a gossip protocol to deal with local models' aggregation

TABLE I: Graph datasets

Dataset	Nodes	Edges	Features	Classes
Cora	2,708	10,556	1,433	7
PubMed	19,717	88,651	500	3
Coauthor physics	34,493	991,848	3,703	6
Reddit	232,965	114,848,857	602	41

under a decentralized-federated learning scenario. The basic idea is to let the models take random walks in the network topology and get updated when they arrive at a server, which is different from ours. We extend Combo to implement a decentralized gossip-based federated graph learning (DGB) baseline with the same flow scheduling strategy with Glint.

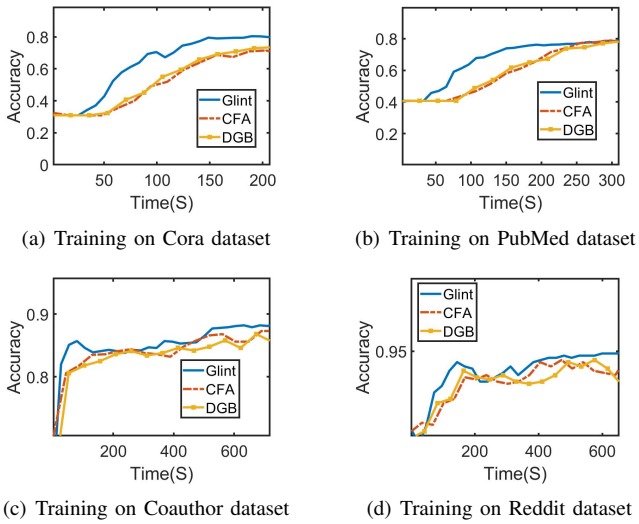
B. Results

The accuracy convergence of Glint over different datasets is shown in Fig. 7. We can see that Glint's convergence is faster than other two baselines. For example, in Fig. 7(a), Glint converges at around 150 seconds with near 80% testing accuracy, while CFA and DGB even do not achieve 70% testing accuracy. The improvement of Glint in training time is over 30% compared with DGB and over 40% compared with CFA. The performance of DGB is better than CFA because its local model aggregation needs less communication time. However, the improvement is limited because local models are a small percentage of the whole network traffic, as shown in Fig. 2.

1) The Effectiveness of the Traffic Throttling Module: Glint and DGB are both decentralized solutions with the same scheduling strategy. However, their model aggregation methods are different. To evaluate the effectiveness of the traffic throttling module, we further modify the local model aggregation strategy of DGB to make it work in a full exchange manner (each server gathers local models from all the others), and the corresponding baseline is referred to as DGB-F. We measure the completion time of different systems after 40 training epochs when they all have converged, and show the results in Fig. 8. The improvement of Glint is about 40% compared with DGB-F. Both DGB-F and CFA always have similar performance because they need to transmit the complete embedding data flows and model data flows.

2) The Effectiveness of the Flow Scheduling Module: To further evaluate the effectiveness of the flow scheduling module, we conduct experiments to modify the proposed scheduling algorithm in Glint by using the commonly used shortest-first (Glint-SF) and fair-share (Glint-FA) algorithms. As shown in Fig. 9, the benefit brought by the flow scheduling module is about 10%-15% compare with Glint-SF and Glint-FA.

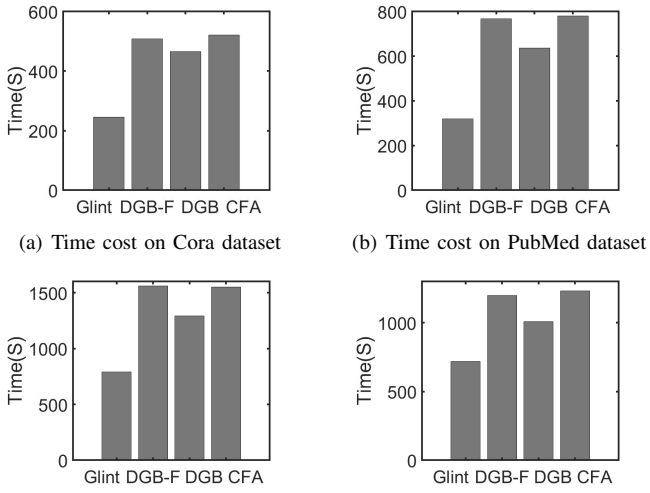
3) The Influence of Data Distribution: In practice, the data amount of different servers may be different. Thus we also study the influence of data distribution. We let the subgraph size obtained by each server be normally distributed. The variance is set as 0.1, 0.5, and 1, respectively, corresponding to low, middle, and high levels. The results of time cost under



(a) Training on Cora dataset (b) Training on PubMed dataset

(c) Training on Coauthor dataset (d) Training on Reddit dataset

Fig. 7: Time-accuracy performance comparison.



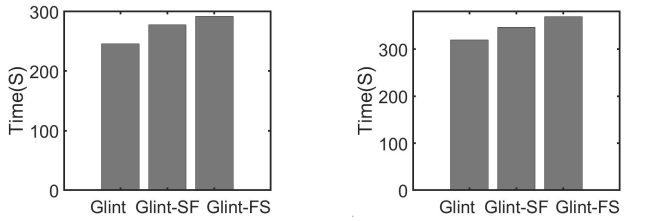
(a) Time cost on Cora dataset (b) Time cost on PubMed dataset

(c) Time cost on Coauthor dataset (d) Time cost on Reddit dataset

Fig. 8: Overall time cost comparison.

different variance levels after 40 training rounds are shown in Fig. 10. We have two observations. The first one is that all solutions' total training time decreases as the growth of the variance levels. Second, the performance gap between Glint and other schemes grows as the distribution variance increases. For example, when handling the Reddit dataset, the performance gap between Glint and CFA is about 40% at the low variance level, while increasing to about 50% at the high level.

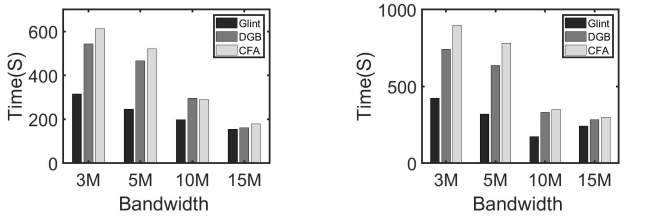
4) The Influence of Network Bandwidth: We study the influence of network bandwidth by changing their values in our emulated network. The results are shown in Fig .11. It is obvious that training time decreases as the network bandwidth grows under all systems. Interestingly, when the bandwidth is getting smaller, the gap between Glint and other schemes become more bigger. This benefit is mainly because



(a) Time cost for Cora dataset (b) Time cost for Coauthor dataset

(c) Time cost for Coauthor dataset (d) Time cost for Reddit dataset

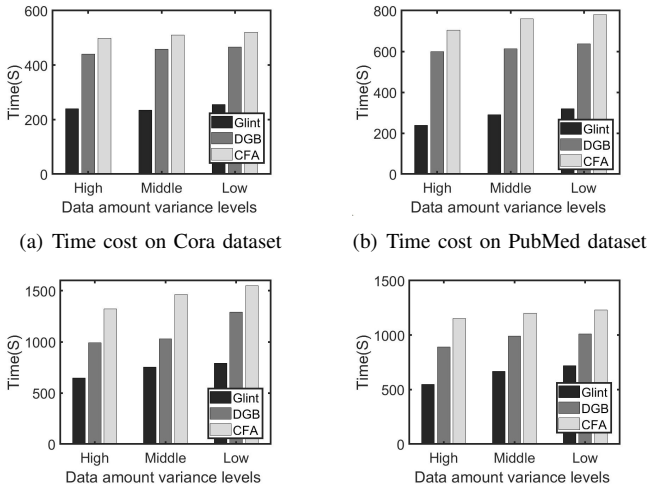
Fig. 9: The effectiveness of flow scheduling module.



(a) Time cost on Cora dataset (b) Time cost on PubMed dataset

(c) Time cost on Coauthor dataset (d) Time cost on Reddit dataset

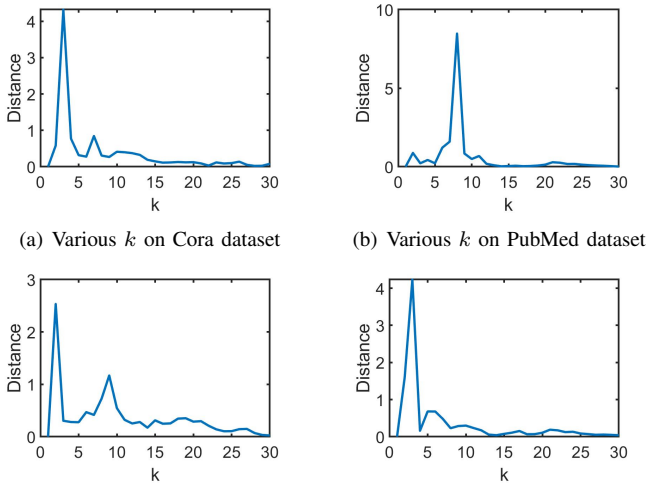
Fig. 11: Time cost with different network bandwidths.



(a) Time cost on Cora dataset (b) Time cost on PubMed dataset

(c) Time cost on Coauthor dataset (d) Time cost on Reddit dataset

Fig. 10: Time cost with different data amount distributions.



(a) Various k on Cora dataset (b) Various k on PubMed dataset

(c) Various k on Coauthor dataset (d) Various k on Reddit dataset

Fig. 12: The influence of threshold k.

of the traffic throttling module. When the bandwidth is getting larger, the gap between Glint and other schemes becomes smaller because of limited overlap between computation and communication. In addition, the effect of the traffic throttling model is also weakened under larger bandwidths.

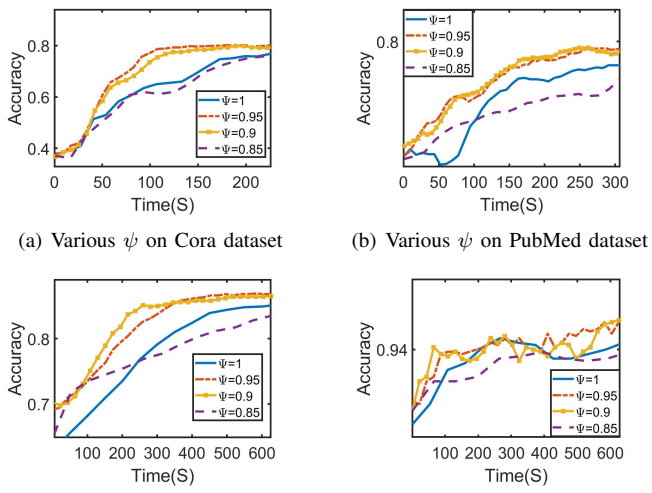
5) The Influence of System Parameters: Glint's performance is also affected by system parameters, k representing the number of training rounds for contribution evaluation and ψ denoting the threshold for neighbor selection in traffic throttling. The influence of different values of k to four datasets is shown in Fig. 12. We conduct experiments to calculate the embedding contribution scores of server 1's neighbors on four datasets. We first calculate the scores based on the complete training epochs. Then we compare the Euclidean distance between the above scores and the new scores calculated based on various k. When the value of k is larger than 10, their distance is stabilized and approaches 0. Although the contributions of different neighbors are still various with the training processing, their testing gaps are turning to be stable and smaller due to the model's converging. Thus we set k to 10 in previous experiments.

For the threshold ψ, we set it as 1, 0.95, 0.9, 0.85, respectively. The training results over four datasets are shown in Fig. 13. We can find that a smaller ψ may cause a large input error and further result in no coverage or slow down the training. We set ψ as 0.9, which has attractive performance in practice.

V. Related Work

A. Federated Learning

Federated learning has been proposed to enable joint learning among distributed data owners without privacy leaking



(a) Various ψ on Cora dataset (b) Various ψ on PubMed dataset

(c) Various ψ on Coauthor dataset (d) Various ψ on Reddit dataset

Fig. 13: The influence of threshold ψ.

concerns [2]. Due to its great promise, substantial growth has occurred in this research field. For example, to address data non-IID issues, Zhao et al. [31] explain the impact with mathematical and try to release the problems by sanding a set of uniform distribution data among servers. Mehryar et al. [32] proposed an agnostic federated learning scheme to avoid the distribution bias. Recently, the DRL algorithm has been adapted to dynamically select a subset of training participants by Wang et al. [33], which can accelerate the model converge and alleviate the non-IID issue. Considerations from the system efficiency perspective, Konecny et al. [1] try to compress model updates to reduce communication costs. CMFL [6] has a different insight to ignore the irrelevant model update and further reduce cost. Tran et al. [34] consider the wireless scene and formulate an optimization problem to trade-off training speed, accuracy, and energy consumption. Zhan et al. [35] further consider lower CPU frequency of mobile devices to improve energy efficiency. However, all of these works focus on training conventional Euclid-space data with a neural network like CNN, which is different from our research.

To cooperative learning among graph data owners with privacy perceiving, SGNN [36] proposed a novel method to protect graph structure information and APGE [37] utilizes GAN to against inference attack. However, compare with our work, the scene-setting is different, and the research issues are various. For non-IID graph data scene, ASFGNN [38] training multiple models and PPGNN [39] further consider feature heterogeneity situation. Their scenarios are also different from ours.

B. Flow Scheduling

Some general flow scheduling schemes try to make the system work efficiency, like FIFO [16], shortest-flow-first [17] and fair mechanisms [18]. These heuristic solutions simplify the problem and discard the space for further optimization. The more complex scheduling approaches claim a better performance [19]–[21], but they only focus on a single communication stage, while our training process contains multiple rounds. Some works express the flow scheduling process as a DAG and utilize the reinforcement learning-based method to learning a scheduling solution [40], [41]. However, these learning-based methods only consider a single agent that knows all the communication situation, which is impossible in our case. Besides, they do not combine the unique characteristics of federated GCN learning.

C. Graph Convolutional Networks

After been proposed, GCN shows its effective in the various fields like recommendation systems [26], temporal link prediction [42], and spam review detection [43]. A realistic situation is that the graph's size is usually too large to load in the memory. Thus various sampling strategies have been proposed to make training efficient. GraphSAGE [44] mandates each node to samples a fixed number of neighbors in each layer, which may lead to data explodes when layers get deeper. To release the issue, VR-GCN [45] utilizes history neighbors' embedding to control variance and reduce the sampled neighbors to two nodes. Cluster-GCN [46] cluster the large graph into subgraphs and then complete convolution separately.

VI. Conclusion

In this paper, we propose a decentralized, federated graph learning system, named Glint, to jointly training a global GCN model among distributed graph data owners. We consider a cross-server graph convolution operation to compress shared embedding and thus avoid privacy exposure. We adopt a series of designs to alleviate the communication burden from embedding sharing, including the traffic throttling module and flow scheduling model. The former evaluates the embedding contribution of other servers and select partial flows for transmission while estimating others. The later dynamic tuns the priority of flows according to the training stage and training completion time. We conducted multi-dimensional comparison experiments with comparison objects, including a centralized solution and a decentralized solution. The experimental results have shown the superiority of Glint over the state-of-the-art.

Acknowledgment

This research is partially supported by the JSPS Grants-in-Aid for Scientific Research (19K20258 and 21H03424), and National Science Foundation of China (61772169). Peng Li is the corresponding author.

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," in *NIPS Workshop on Private Multi-Party Machine Learning*, 2016.

[2] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proceedings of the Artificial Intelligence and Statistics Conference (AISTATS)*, 2017, pp. 1273–1282.

[3] Y. Zhan, J. Zhang, Z. Hong, L. Wu, P. Li, and S. Guo, "A survey of incentive mechanism design for federated learning," *IEEE Transactions on Emerging Topics in Computing*, pp. 1–1, 2021.

[4] Y. Zhan, P. Li, Z. Qu, D. Zeng, and S. Guo, "A learning-based incentive mechanism for federated learning," *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 6360–6368, 2020.

[5] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[6] W. Luping, W. Wei, and L. Bo, "Cmfl: Mitigating communication overhead for federated learning," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2019, pp. 954–964.

[7] F. Sattler, S. Wiedemann, K.-R. Müller, and W. Samek, "Robust and communication-efficient federated learning from non-iid data," *IEEE transactions on neural networks and learning systems*, 2019.

[8] H. H. Yang, Z. Liu, T. Q. Quek, and H. V. Poor, "Scheduling policies for federated learning in wireless networks," *IEEE Transactions on Communications*, vol. 68, no. 1, pp. 317–333, 2019.

[9] M. Chen, Z. Yang, W. Saad, C. Yin, H. V. Poor, and S. Cui, "A joint learning and communications framework for federated learning over wireless networks," *IEEE Transactions on Wireless Communications*, 2020.

[10] W. Shi, S. Zhou, and Z. Niu, "Device scheduling with fast convergence for wireless federated learning," in *ICC 2020-2020 IEEE International Conference on Communications (ICC)*. IEEE, 2020, pp. 1–6.

[11] Y. Zhan, P. Li, and S. Guo, "Experience-driven computational resource allocation of federated learning by deep reinforcement learning," in *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2020, pp. 234–243.

[12] Z. Li, H. Zhou, T. Zhou, H. Yu, Z. Xu, and G. Sun, "Esync: Accelerating intra-domain federated learning in heterogeneous data centers," *IEEE Transactions on Services Computing*, pp. 1–1, 2020.

[13] T.-Y. Liu, W. Chen, and T. Wang, "Distributed machine learning: Foundations, trends, and practices," in *Proceedings of the 26th International Conference on World Wide Web Companion*, Republic and Canton of Geneva, CHE, 2017, p. 913–915.

[14] S.-J. Yang and Y.-R. Chen, "Design adaptive task allocation scheduler to improve mapreduce performance in heterogeneous clouds," *Journal of Network and Computer Applications*, vol. 57, pp. 61–70, 2015.

[15] J.-w. Lee, G. Jang, H. Jung, J.-G. Lee, and U. Lee, "Maximizing mapreduce job speed and reliability in the mobile cloud by optimizing task allocation," *Pervasive and Mobile Computing*, vol. 60, p. 101082, 2019.

[16] F. R. Dogar, T. Karagiannis, H. Ballani, and A. Rowstron, "Decentralized task-aware scheduling for data center networks," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 431–442, 2014.

[17] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, "pfabric: Minimal near-optimal datacenter transport," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 435–446, 2013.

[18] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica, "Managing data transfers in computer clusters with orchestra," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 98–109, 2011.

[19] M. Chowdhury, Y. Zhong, and I. Stoica, "Efficient coflow scheduling with varys," in *Proceedings of the 2014 ACM conference on SIGCOMM*, 2014, pp. 443–454.

[20] H. Zhang, L. Chen, B. Yi, K. Chen, M. Chowdhury, and Y. Geng, "Coda: Toward automatically identifying and scheduling coflows in the dark," in *Proceedings of the 2016 ACM SIGCOMM Conference*, 2016, pp. 160–173.

[21] H. Tan, S. H.-C. Jiang, Y. Li, X.-Y. Li, C. Zhang, Z. Han, and F. C. M. Lau, "Joint online coflow routing and scheduling in data center networks," *IEEE/ACM Transactions on Networking*, vol. 27, no. 5, pp. 1771–1786, 2019.

[22] J. Daily, A. Vishnu, C. Siegel, T. Warfel, and V. Amatya, "Gossipgrad: Scalable deep learning using gossip communication based asynchronous gradient descent," *arXiv preprint arXiv:1803.05880*, 2018.

[23] C. Hu, J. Jiang, and Z. Wang, "Decentralized federated learning: A segmented gossip approach," *arXiv preprint arXiv:1908.07782*, 2019.

[24] K. Hsieh, A. Harlap, N. Vijaykumar, D. Konomis, G. R. Ganger, P. B. Gibbons, and O. Mutlu, "Gaia: Geo-distributed machine learning approaching {LAN} speeds," in *14th Symposium on Networked Systems Design and Implementation (NSDI)*, 2017, pp. 629–647.

[25] F. Chen, P. Li, D. Zeng, and S. Guo, "Edge-assisted short video sharing with guaranteed quality-of-experience," *IEEE Transactions on Cloud Computing*, pp. 1–1, 2021.

[26] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 974–983.

[27] L. Li, P. Wang, J. Yan, Y. Wang, S. Li, J. Jiang, Z. Sun, B. Tang, T.-H. Chang, S. Wang *et al.*, "Real-world data medical knowledge graph: construction and applications," *Artificial intelligence in medicine*, vol. 103, p. 101817, 2020.

[28] T. Nishio, R. Shinkuma, and N. B. Mandayam, "Estimation of individual device contributions for incentivizing federated learning," *arXiv preprint arXiv:2009.09371*, 2020.

[29] M. Wang, L. Yu, D. Zheng, Q. Gan, Y. Gai, Z. Ye, M. Li, J. Zhou, Q. Huang, C. Ma *et al.*, "Deep graph library: Towards efficient and scalable deep learning on graphs," *arXiv preprint arXiv:1909.01315*, 2019.

[30] https://snap.stanford.edu/data/p2p-Gnutella08.html.

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

[32] M. Mohri, G. Sivek, and A. T. Suresh, "Agnostic federated learning," *arXiv preprint arXiv:1902.00146*, 2019.

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

[34] N. H. Tran, W. Bao, A. Zomaya, N. M. NH, and C. S. Hong, "Federated learning over wireless networks: Optimization model design and analysis," in *IEEE INFOCOM*, 2019, pp. 1387–1395.

[35] Y. Zhan, P. Li, and S. Guo, "Experience-driven computational resource allocation of federated learning by deep reinforcement learning," in *Proc. of IPDPS*, 2020.

[36] G. Mei, Z. Guo, S. Liu, and L. Pan, "Sgnn: A graph neural network based federated learning approach by hiding structure," in *2019 IEEE International Conference on Big Data (Big Data)*. IEEE, 2019, pp. 2560–2568.

[37] K. Li, G. Luo, Y. Ye, W. Li, S. Ji, and Z. Cai, "Adversarial privacy preserving graph embedding against inference attack," *IEEE Internet of Things Journal*, 2020.

[38] L. Zheng, J. Zhou, C. Chen, B. Wu, L. Wang, and B. Zhang, "Asfgnn: Automated separated-federated graph neural network," *arXiv preprint arXiv:2011.03248*, 2020.

[39] J. Zhou, C. Chen, L. Zheng, X. Zheng, B. Wu, Z. Liu, and L. Wang, "Privacy-preserving graph neural network for node classification," *arXiv preprint arXiv:2005.11903*, 2020.

[40] H. Mao, M. Schwarzkopf, S. B. Venkatakrishnan, Z. Meng, and M. Alizadeh, "Learning scheduling algorithms for data processing clusters," in *Proceedings of the ACM Special Interest Group on Data Communication*, 2019, pp. 270–288.

[41] P. Sun, Z. Guo, J. Wang, J. Li, J. Lan, and Y. Hu, "Deepweave: Accelerating job completion time with deep reinforcement learning-based coflow scheduling," in *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, 2020.

[42] K. Lei, M. Qin, B. Bai, G. Zhang, and M. Yang, "Gcn-gan: A non-linear temporal link prediction model for weighted dynamic networks," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 388–396.

[43] A. Li, Z. Qin, R. Liu, Y. Yang, and D. Li, "Spam review detection with graph convolutional networks," in *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, 2019, pp. 2703–2711.

[44] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in neural information processing systems*, 2017, pp. 1024–1034.

[45] J. Chen, J. Zhu, and L. Song, "Stochastic training of graph convolutional networks with variance reduction," *arXiv preprint arXiv:1710.10568*, 2017.

[46] W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C.-J. Hsieh, "Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 257–266.