ADGRAPH: ACCURATE, DISTRIBUTED TRAINING ON LARGE GRAPHS

Lizhi Zhang, Zhiquan Lai, Feng Liu and Zhejiang Ran

Parallel and Distributed Key Laboratory of National Defence Technology,
National University of Defence Technology, Changsha, China

ABSTRACT

Graph neural networks (GNNs) have been emerging as powerful learning tools for recommendation systems, social networks and knowledge graphs. In these domains, the scale of graph data is immense, so that distributed graph learning is required for efficient GNNs training. Graph partition-based methods are widely adopted to scale the graph training. However, most of the previous works focus on scalability other than the accuracy and are not thoroughly evaluated on large-scale graphs. In this paper, we introduce ADGraph (accurate and distributed training on large graphs), exploring how to improve accuracy while keeping large-scale graph training scalability. Firstly, to maintain complete neighbourhood information of the training nodes after graph partitioning, we assign l-hop neighbours of the training nodes to the same partition. We also analyse the accuracy and runtime performance of graph training, with different l-hop settings. Secondly, multi-layer neighbourhood sampling is performed on each partition, so that the mini-batch generated can accurately train target nodes. We study the relationship between convergence accuracy and the sampled layers. We also find that partial neighbourhood sampling can achieve better performance than full neighbourhood sampling. Thirdly, to further overcome the generalization error caused by large-batch training, we choose to reduce batchsize after graph partitioned and apply the linear scaling rule in distributed optimization. We evaluate ADGraph using GraphSage and GAT models with ogbn-products and Reddit datasets on 32 GPUs. Experimental results show that ADGraph achieves better performance than the benchmark accuracy of GraphSage and GAT, while getting 24-29 times speedup on 32 GPUs.

KEYWORDS

Graph neural networks; Distributed training; Multi-GPU; Deep learning; Parameter Server.

1. INTRODUCTION

Graph neural networks (GNNs) are becoming more and more influential in solving various challenges in many practical applications, such as social networks [1], paper citations [2], biological networks [3, 4], product customer relationships [5], recommendation systems [1], and knowledge graphs [6], which data can be naturally represented as graph structures. The graph data structure is widely used to model data with complex connections between elements because of good expressive ability. The powerful function of GNNs in modelling the dependency relationship between graph nodes has made a great breakthrough in the research field related to graph analysis, which is an emerging field in deep learning [7, 8].

Simultaneously, the scale of graphs in industry domains has developed rapidly [9]. For example, the social network maintained by Facebook has nearly 2 billion users, and Amazon's customer shopping network has hundreds of millions of nodes. Larger datasets and network structures can improve the accuracy of tasks. Technologies that can effectively analyse and process large-scale

David C. Wyld et al. (Eds): CoSIT, AIAPP, SIGL, CRIS, NLPML - 2021
pp. 97-115, 2021. CS & IT - CSCP 2021 DOI: 10.5121/csit.2021.110408

graph data have gradually become one of the research hotspots in academia and industry currently [10]. However, compared with real-world graphs, many optimizations on graph datasets mainly focus on small datasets. For example, Cora [11, 12], Citeseer [13, 14], Pubmed [15] and Blog [16]. Their specific parameters are shown in Table 1. Most of the evaluations are carried out for small graphs on a single machine, and there are only 2700 to 20000 nodes in the tasks of node classification. There are a few kinds of research on distributed training for large-scale graphs. Even though some authors aim at distributed graph learning, these small datasets are mainly used for training [17-19]. Because models are widely developed on these small datasets, most models cannot be extended to larger graphs. GraphSage [20] and Cluster-GCN [5] provide a method to perform random mini-batch training does not need to read graph features of all nodes into GPU or CPU memory. However, these two mini-batch training methods are still limited in accelerating the training of large-scale graph datasets in a single machine.

A few works have been developed to scale GNNs training on large graph data in the distributed clusters. However, they focus on the scalability other than the accuracy, such as NeuGraph [21] and PCGCN [22] aim to speedup GNNs training. However, there is no discussion about the changes in the accuracy of graph training [23]. Some GNNs frameworks [19, 24] built-in industrial scene adopt distributed mini-batch training. Nevertheless, none of these frameworks uses appropriate graph partitioning to maximize the accuracy of GNNs training. Moreover, for distributed deep learning training on multiple GPUs, it remains a problem that as the GPU number increases, the training accuracy decreases [25].

Table 1. Small datasets used in graph neural networks.

Datasets	vertex	edge	feature	label
Core	2708	5429	1433	7
Citeseer	3327	4732	3703	6
Pubmed	19417	44338	500	3
Blog	10400	678300	128	32

We introduce ADGraph, which uses neighbourhood-contained graph partition, multi-layers neighbourhood sampling and overcoming generalization error method in distributed GNNs training. On the one hand, the training of graph models on large-scale datasets can be accelerated through multi-GPU training. On the other hand, the distributed graph learning can still maintain high training accuracy through appropriate graph partition, neighbour sampling and distributed optimization methods. In order to verify our proposed methods, we train GraphSage and GAT model with two large-scale graph datasets (ogbn-products and Reddit) on GPU clusters, which significantly reduces training time. We also use a graph partition that includes neighbourhoods and a multi-layer neighbourhood sampling strategy. Even in a distributed environment, it can achieve the same accuracy as single-GPU training. In summary, our contributions are as follows:

- We use neighbourhood contained graph partition to ensure the completeness of training nodes information in each partition. Then, the mini-batch generated from multi-layers neighbourhood sampling can train nodes accurately.
- We combine a distributed optimization method with graph learning. We use the data synchronization method and linear scaling rule in the distributed training. This further improves the training accuracy of the GNN models.
- We test the GraphSage and GAT model on the ogbn-products and Reddit datasets. Experimental results show that the test accuracy on 32 GPUs is better than the benchmark accuracy using GraphSage and GAT models. The running time is accelerated 24-29 times on different datasets.

The remainder of this paper is organized as follows. Section 2 introduces the background of distributed GNNs training. We present ADGraph training methods in Section 3. Section 4 evaluates and analyses the key technologies to fulfil accurate and large-scale GNNs training. Finally, we conclude the paper in Section 5.

2. BACKGROUND

2.1. Graph Neural Networks

Graph neural networks (GNNs) are representative work in deep learning. Training neural network on graph data has been widely used because its model accuracy is much higher than that of traditional multi-layer perceptron [26]. GNNs layers generate intermediate embedding by aggregating the information from the in-edge neighbours of the target nodes. After superimposing several GNNs layers, the final embedding is obtained, which integrates the whole receptive field of the target node. Specifically, the graph neural networks iteratively update the node representation according to:

$$h_i^{l+1} = \sigma(W^l \frac{1}{|N_i|} \sum h_j^i) \tag{1}$$

Where h_i^{l+1} is the embedding of node i in the $(l+1)$-th layer. N_i is the node set connect with node i. N_i represents the in-edge number of node i. σ() represents the nonlinear activation function. W^l is the learnable parameters of layer l. GNNs first aggregate all values from the in-edge neighbours of each node to obtain new values for these nodes. After that, GNNs propagate this new value to target nodes throughout-edges. After l times of such aggregation and propagation, the calculation of GNNs is completed.

GraphSage [20] only needs to aggregate data sampled from the graph, without considering other nodes. GraphSage provides different ways to aggregate information of adjacent nodes. In the average version of GraphSage, the update formula of node embedding is Equation (2). GraphSage can form a mini-batch by sampling a specific size of neighbour nodes and does not need to get the adjacency matrix of the whole graph. This is very useful in training large-scale datasets.

$$h_i^{l+1} = \sigma(W^l Concat(h_i^l, \frac{1}{|N_i|} \sum_{j \in N_j} h_j^l)) \tag{2}$$

Graph Attention Network (GAT) [27] aggregates features of neighbouring vertices to the central node and learns new nodes features by using local stationary on Graph. GAT makes use of the attention coefficient and introduces anisotropy into the neighbourhood aggregation function. This network adopts a multi-head structure to increase the learning ability. Equation (3) is the updated formula of GAT, where W_l^k are k linear projections heads, and e is the attention coefficient of each head. GAT has stronger learning ability because the model can better capture correlation between node features [28].

$$h_i^{l+1} = Concat_{k=1}^{K}(\sigma(\sum_{j \in N_i} e_{ij}^{k,l} W^{k,l} h_j^l)) \tag{3}$$

2.2. Parameter Sever architecture and data parallelism

Parameter server (PS) is one of the commonly used frameworks for distributed deep learning [29]. PS aims to improve the training efficiency of big data and large models while maintaining accuracy. There are two parts in Parameter Server architecture: parameter server (PS) and worker. As shown in Figure 1, PS maintains a global shared parameter and updates all parameters together. Each worker is responsible for handling local training tasks, obtaining the latest model parameters from PS nodes, and sending model gradients generated by the local worker to PS [30].

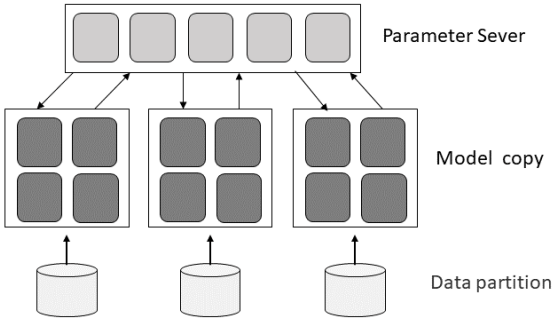


Figure 1. The architecture of the parameter server.

The mainstream method of distributed deep learning is data parallelism, which has higher training efficiency [31]. As shown in the Figure 2, In data parallel method, the whole dataset is divided into multiple machines. Each machine has a local copy of the model and updates the local model with the assigned data [32]. In the synchronous update, gradients of different batches are calculated at each worker. Gradients are averaged across machines to apply consistent updates to model copies in each worker [33]. This synchronization method is widely used in large-scale systems.

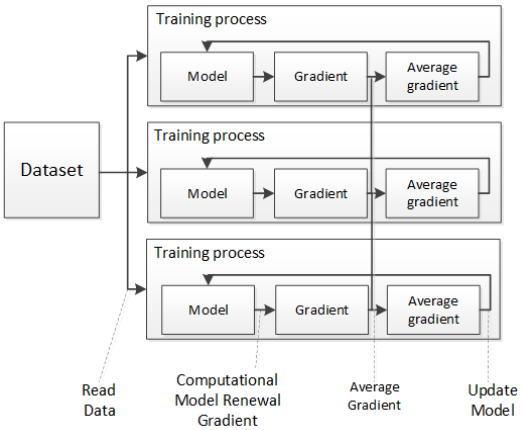


Figure 2. The process of data parallelism.

2.3. Large mini-batch Stochastic Gradient Descent

When mini-batch is used for GNNs training on multi-GPUs, linear scaling rule can reduce the training error caused by large mini-batch. After the mini-batch training is completed, stochastic gradient descent (SGD) [34] performs the following update:

$$w_{t+1} = w_t - \eta \frac{1}{n} \sum_{x \in \beta} \nabla l(x, w_t) \tag{4}$$

Here β is a mini-batch sampled from the neighbourhood of target nodes in partitioned graphs. t is the update times and η is the learning rate. $n=|\beta|$ is the mini-batch size. According to Equation (4), when learning rate η and batchsize are n, after k iterations of SGD, Equation (5) can be obtained:

$$w_{t+k} = w_t - \eta \frac{1}{n} \sum_{j<k} \sum_{x \in \beta_j} \nabla l(x, w_{t+j}) \tag{5}$$

On the other hand, using mini-batch of size kn and learning rate $\hat{\eta}$ to update once can get:

$$\hat{w}_{t+1} = w_t - \hat{\eta} \frac{1}{kn} \sum_{x \in \beta_j} \nabla l(x, w_t) \tag{6}$$

According to Equations (5), (6), it can be seen that the results of updating k times with a small-batch and updating once with a large-batch are different. Therefore, in order to keep the weights unchanged in both cases after SGD update. Set learning rate $\hat{\eta}=k\eta$ when updating large batches, and the updated results w_{t+k} and $\hat{w}_{t+1}$ will be approximately the same [35]. That is, linear scaling rule can reduce the distributed GNNs training error on Multiple GPUs.

3. ADGRAPH TRAINING METHODS

The difference between distributed training of graph neural networks and traditional distributed training lies in graph partition and mini-batch sampling. The overall structure of distributed training with four machines is shown in Figure 3. Specifically, there are three processes in graph distributed training: Graph Server, Sampler and Trainer. The Graph Server process needs to run on each machine to store the graph partitions (including graph structure, nodes features and nodes labels). Sampler process samples nodes from Graph Server and generate mini-batch required by trainer process. It can be noticed that a sampler process can obtain data from multiple Graph Servers. Trainer process can only obtain mini-batch from the sampler on its local machine. Then, the trainer calls the all-reduce primitive to update model parameters.

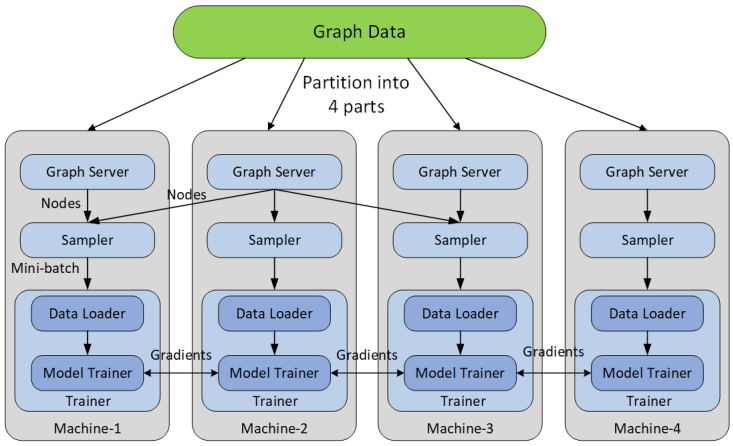


Figure 3 . Distributed graph neural networks training process.

The training process starts with the partition of graph data and then carries out mini-batch training. The steps performed in each mini-batch training process include: (1) Sampling the neighbourhood of target nodes from a local partition to generate mini-batch. (2) Obtaining features and labels involved in mini-batch from the global graph data. (3) Performing forward and backward propagation on the features to calculate the gradients of each layer. (4) Trainer process uses all-reduce to accumulate gradients. Then the trainer applies averaged gradients to update parameters of the model.

3.1. Graph partition containing neighbourhood information

Graph partitioning is the first step in distributed graph learning. Firstly, nodes are assigned to partitions using METIS [36] or random graph partitioning algorithm. Then the partitioned graph structure is constructed according to the result of node allocation. Finally, node features are segmented according to the partition results. We partition the graph structure, node features and labels, and distribute them on cluster machines in distributed training. There are two potential problems after graph partition: (1) Deleting some edges between nodes may affect performance. (2) Graph clustering algorithm (METIS partition) tends to cluster similar nodes together, resulting in the distribution of node categories different from the original dataset. Therefore, estimation of the gradient is biased when performing SGD updates.

We solve these two problems by two methods, namely partition graph with the intact neighbourhood of target nodes and increasing batch label entropy.

3.1.1. Keeping the neighbourhood information of the target nodes intact

According to Equation 1, nodes in the *l-hop* neighbourhood of the target nodes contain enough and necessary information for training the *l-layer* GNNs model. Therefore, in *l-layers* GNNs model, the embedding of target node only depends on its *l-hop* neighbourhood, rather than the entire graph.

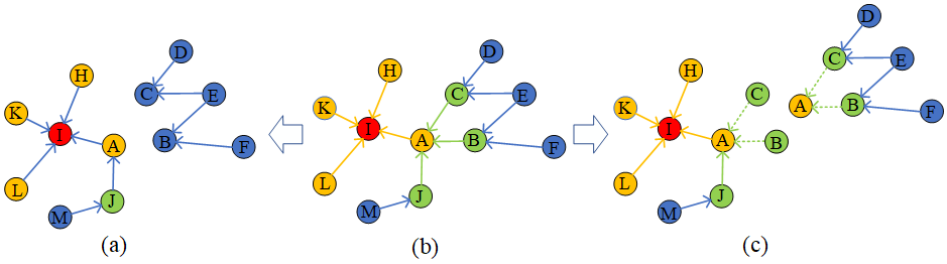


Figure 4. Example of graph partition(b) is the original graph, (a) is the direct partition of the graph without keeping the neighbourhood of I, the partition in (c) repeatedly stores the *2-hop* neighbour B and C.

After graph partition, nodes at the subgraph boundary are stored in two adjacent subgraphs. For completeness and efficiency, each partition contains not only nodes and edges belonging to itself, but also *l-hop* neighbourhoods of nodes in other partitions. As shown in Figure 4, (b) is the original graph. It is assumed that the GNNs model has two layers, thus, generating the embedding of node *I* needs the complete information of *1-hop* neighbour *HKL* and *2-hop* neighbour *BCJ*. Graph (a) is the result of partitioning Graph (b). Due to the lack of connection between *AB* and *AC*, the embedding of *I* on the partitioned subgraph will lack the information of *B* and *C*, resulting in inaccurate generating the embedding of *I*. By contraries, Graph (c) keeps the neighbourhood of *I*. Although subgraph becomes smaller after graph partition, the information to generate the embedding of *I* is intact. Therefore, this partition method ensures the accuracy of model training.

The *l-hop* neighbourhood of target nodes can provide enough information for the target nodes in the partition which avoids missing connection information after partition. Even after graph partition, the data in each partition is intact. This ensures the convergence accuracy when training *l-layer* GNNs, and it can achieve the same performance as that without partition.

3.1.2. Increase batch label entropy

Distributed graph learning is the training of graph neural networks by using the graph data to predict and simulate unknown large-scale graph data. Therefore, each mini-batch should be generally representative. We should generalize rules from existing graph data to make decisions on unknown graph data. If the training data is not representative, the rules will be poorly summarized, and significant deviations will be generated in the inference process on unknown graph data.

Vanilla Cluster-GCN [5] shows that METIS method partitions the graph into a large number of partitions, and nodes in the partitions tend to specific categories. Generating mini-batch from these partitions may lead to lack of representativeness of mini-batches. In this case, label entropy of most mini-batches is smaller than that of random partition. This indicates that the label distribution of mini-batches is biased towards some specific labels. This will increase the variance between different batch and may affect the convergence of SGD.

In order to avoid label deviation, we do not partition the graph into a large number subgraph. We set the number of partitions consistent with the number of machines. This can reduce network communication when the neighbourhood expands and make each batch have various labels that will not bias towards specific labels. In Figure 5, ogbn-products dataset is divided into eight partitions by METIS and random methods to show the example of label distribution. We calculate entropy according to the label distribution of each batch. It can be seen that the label

entropy of the mini-batches from METIS partition and random partition is similar. Therefore, the convergence of SGD will not be affected by METIS partition.

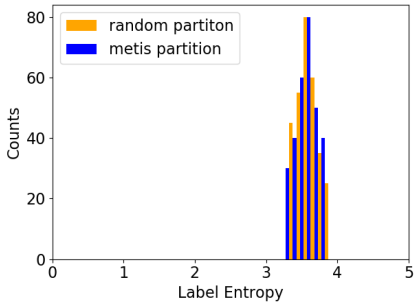


Figure 5. The label entropy of each batch when ogbn-products is partition into 8 parts.

3.2. Multi-layer neighbourhood sampling

Single-layer subgraph can only perform forward propagation once. Therefore, multi-layer neighbourhood sampling is required to generate the mini-batch when compute multi-layer GNNs. However, the degrees of nodes in a large graph are generally vast, multi-layer subgraph sampling will cause exponential expansion of neighbour nodes which consumes lots of memory. We use partial neighbourhood sampling for multi-layer GNN. For different *hop* neighbourhoods of the target nodes, a fixed number of neighbours are sampled. Due to the neighbours of the target nodes are randomly sampled, all neighbours will participate in training after multiple epochs. The clustering coefficient is a popular measure of how clustered a node's local neighbourhood is. GraphSage [20] has proved that partial neighbourhoods sampling is capable of approximating clustering coefficients to an arbitrary degree of precision, even when the node feature inputs are sampled from an absolutely continuous random distribution.

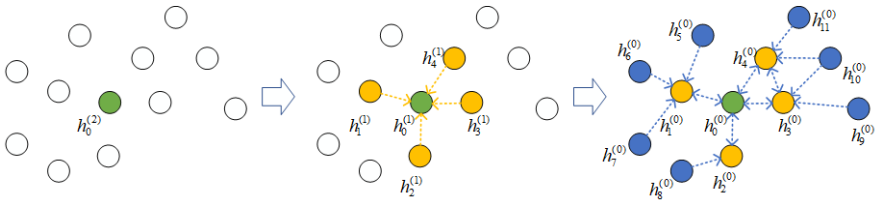


Figure 6. Multi-layer neighbourhood sampling to generate a mini-batch.

Figure 6 shows the process of multi-layer neighbourhood sampling: 1) For each gradient descent step, we select some target nodes to calculate their final representations at *l-th* layer. 2) Then, obtain part of *1-hop* neighbours of the target nodes at *l-1* layer, and obtain part of the *2-hop* neighbours of the target nodes at *l-2* layer. 3) This process continues till the input layer. The iteratively constructing dependency graph of multi-layer neighbourhood sampling generates a mini-batch. The forward calculation process is the opposite, which is calculated from (c) to (a).

Since the sampled mini-batch contains *l-hop* neighbourhood of the target nodes *v*, the information is intact when perform *l-layers* GNNs to calculate the embedding of target nodes. The embedding of the target nodes in *l-th* layer is calculated from the equation as follows:

$$h^{(l)} = A_{l-1}\sigma(\ldots A_1\sigma(A_0 X W^0)W^1\ldots)\,W^{l-1} \tag{7}$$

Where A is the subgraph represented in adjacency matrix in each mini-batch, X is the feature matrix, W is the weight parameter of each layer, and its loss function can be expressed as:

$$L = \frac{1}{|\mathrm{V}_A|} \sum loss(y_i, h_i^l) \tag{8}$$

In each step, we first sample the mini-batch of target nodes v, then perform SGD to update the parameters based on the gradient L. The gradient calculation and update only require the adjacency matrix A and node features X of the current mini-batch, ensuring the accuracy of the embedding in the last layer. Since the *k-hop* neighbourhood contains sufficient and necessary information for training GNNs model, trainers become independent of each other. They train mini-batch without additional communication with other trainers. Therefore, training of the GNNs model is similar to that of conventional deep learning.

3.3. Overcoming generalization errors of distributed GNNs training

In graph deep learning, it is stated that model trained with large batchsize is often inferior to that with small batchsize [37]. [38] found that when the training accuracy is consistent, the generalization performance of a model trained with large batchsize will be significantly lower than model trained with small batchsize.

Unlike single GPU training, data is sampled from the same dataset in each step. The batchsize is small compared to the entire data set, which can ensure sufficient parameter updates. However, in distributed GNNs training, the subset on each machine becomes smaller after graph partition. Each trainer samples mini-batch from the local partition. A larger batchsize will cause the parameter update insufficient, resulting in larger generalization errors. After graph partition, the batchsize should be reduced, which can increase the amount of model parameter updates. In this way, an accurate model can be trained efficiently under distributed training.

Although the generalization performance of the model can be guaranteed by increasing the amounts of updates, this will affect the benefits of distributed training. In order to maintain the accuracy of training and generalization while training on multiple machines, the linear learning rate scaling rule has a particularly important role in distributed learning. Because this allows data parallelism to be extended to more GPUs, it also improves the distributed training accuracy. Facebook large-scale training [25] has proved that small batch and large batch SGD updates not only get the same final precision model, but also match the training curves very well. We prove that the linear scaling rule is effective in the large-scale real-world graphs through experiments.

4. EVALUATION AND ANALYSIS

4.1. Experiment Setup

In this section, focusing on the task of node classification, we will evaluate the efficiency and performance of the proposed methods through experiments on several GNNs models and datasets.

GNNs models. We use the following two representative GNNs models in the experiment: GraphSage and GAT. Related concepts have been introduced in the background. Because they adopt different aggregation method of neighbours in the graph, we choose these models in experiment. Table 2 shows the default settings of parameters when training the two models.

Table 2. Parameter settings for GNN models in the experiment.

Models	layers	hiddens	Sampled neighbours	batchsize	Epochs
GraphSage	3	512	(10,10,10)	250	30
GAT	3	128	(10,10,10)	250	30

Table 3. Large-scale graph data statistics.

Datasets	vertex	edge	feature	label	Avg degree
ogbn-products	2,449,029	61,859,140	100	47	50
Reddit	232,965	1,606,919	602	41	100

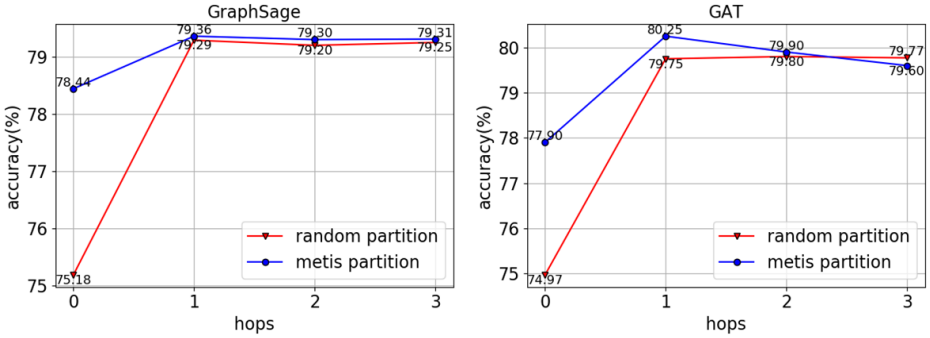
Datasets. The real-world graph datasets used in the experiment are listed in Table 1. The Reddit [20] dataset is formed by Reddit online discussion forum, and ogbn-product [9] comes from Amazon product co-purchasing network. The *feature* column in table 1 represents the feature dimensions of each node, and the *label* column indicates the number of label categories.

Experimental environment. We evaluated experimental results on GPU cluster and used up to 8 machines on the cluster for training. Each machine has four Nvidia Tesla GPU and two Intel Xeon CPU, and machines are connected through InfiniBand ConnectX FDR 56GB/s internet. Operating system version used is Redhat4.8, and libraries of CUDA10.0 are used. Our experiments are carried out on Pytorch, a deep learning framework, and Deep graph Library (DGL) [39]. DGL is a Python package that interfaces between tensor-oriented frameworks (such as Pytorch and MXNet) and graph structure data, which makes it easy to implement GNNs.

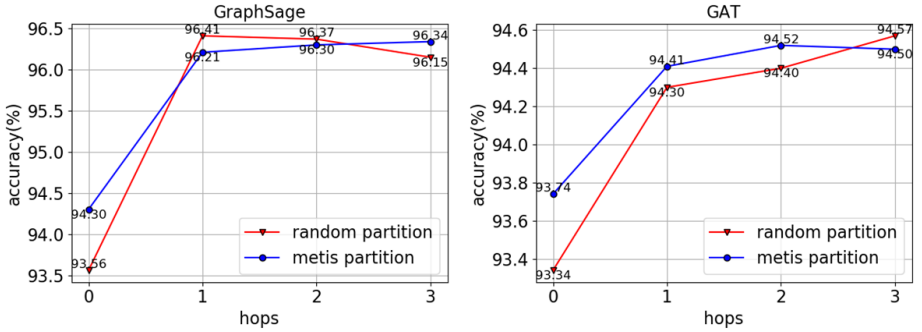
4.2. Comparison of Partition Methods

We use random and METIS [36] methods to partition the datasets, and compare convergence accuracy and running time of the two methods. By increasing neighbour layers around the target nodes in each partition (hop=k means expanding the neighbourhood of the target nodes to the k-th layers), the effect of neighbourhood partition with different layers is verified.

Accuracy. The convergence accuracy of the experiment is shown in Figure 7. It can be seen that when hop=0, the performance gap between these two methods is the largest. Figure 9 shows the number of edges cut by the random and METIS partition when the hop=0. It can be noticed that the edges cut by METIS partition is much less than random partition. This is due to random partition randomly assigned nodes into partitions, which will increase the randomness of edge segmentation and make many neighbours of nodes disappear. However, METIS divides into clusters, and nodes with many connections will be partitioned into the same partition. The segmentation is mainly between clusters, which will significantly reduce cut edges. Therefore, under the METIS partition, similar nodes can be clustered together to capture the clustering and graph structure better. METIS partition preserves the graph structure better, and results obtained during information aggregation are more accurate than random partition.



(a) Training on partitioned ogbn-products dataset.

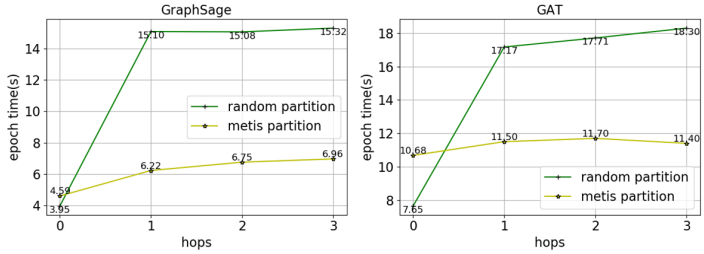


(b) Training on partitioned Reddit dataset.

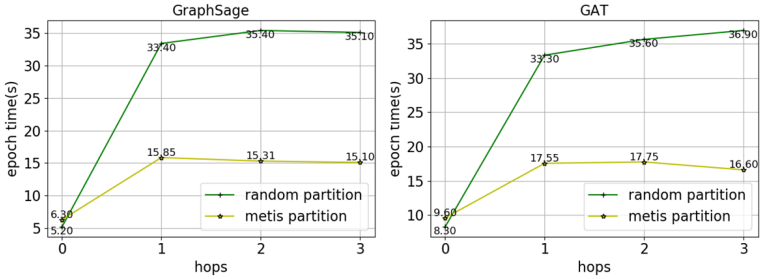
Figure 7. Convergence accuracy after partition datasets with different *hops*.

It can be seen from Figure 7 that when hop goes from 0 to 1, the convergence accuracy will increase significantly. The reason can be explained by Figure 10. Because of the strong connectivity among nodes in the graph, it will almost extend to the whole graph when expanding the 1-hop neighbourhood of the target nodes. When hops increase from 1 to 3, the convergence accuracy remains unchanged. Because the number of nodes in the extended neighbours is only slightly increased. Furthermore, when hops are greater than or equal to 1, the convergence accuracy of random partition and METIS partition is not much different. Because in these two methods, the number of nodes in the partitions is very similar, both can effectively expand the structure to the whole graph.

Run time. Figure 8 shows that the epoch time of the METIS partition is similar to random partition when *hop=0*. However, the epoch time of METIS is much shorter than that of random partition when *hops>0*. Although random and METIS partition can be extended to the full graph, nodes connectivity in local machines are different. METIS partition can generate more closely connected clusters, while nodes in the partitions generated by a random partition are randomly connected. Therefore, the graph data read during training is different. Random partition requires more communication with other machines for nodes data. Cross-machine communication takes more time, which leads to longer epoch time for random partition. This shows that graph clustering is significant, and the partitions of graph should not be randomly generated.



(a) Training on partitioned ogbn-products dataset.



(a) Training on partitioned Reddit dataset.

Figure 8. The epoch time after partition datasets with different *hops*.

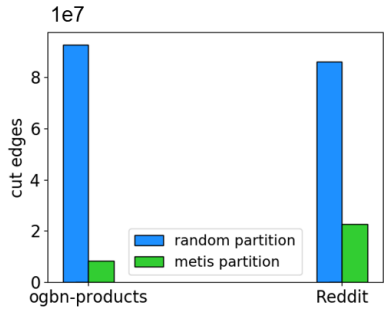


Figure 9. The number of cut edges after random and METIS partition.

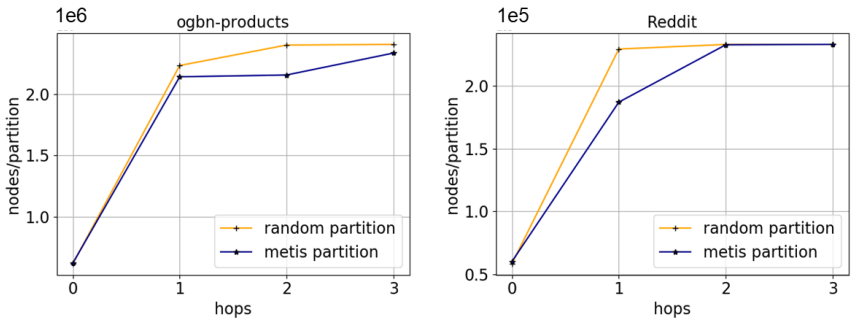


Figure 10. The average number of nodes in each partition as *hops* increase.

4.3. Multi-Layer Neighbourhood Sampling

When verifying the multi-layer sampling method, we fix the sampled neighbours for each node. We test convergence accuracy by increasing the sampled layers. Figure 11 shows the curves of training accuracy on 2 datasets, from which we can observe that the increase of layers can improve the training accuracy, but the improvement in accuracy after the third layer is not obvious. It is worth noting that the training accuracy of GAT failed to converge within 30 epochs

and get a dramatic loss of accuracy when 4 layers are used. A possible reason is that the structure of deeper GAT is more complex and has more parameters, resulting in the optimization for deeper GAT becomes more difficult.

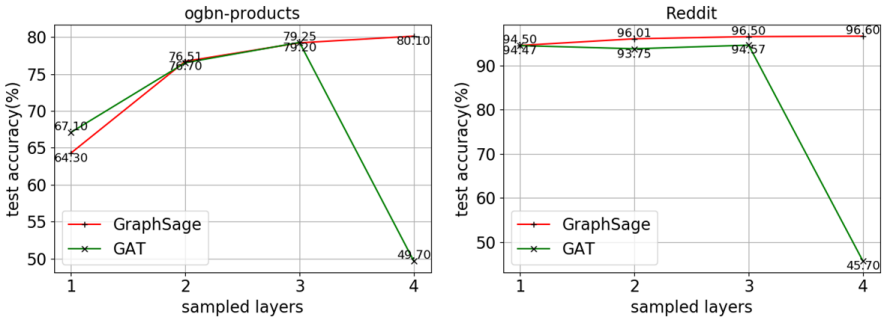


Figure 11: The model accuracy changes with the increasing sampled layers.

We test convergence accuracy and run time by changing the sampled neighbours from 2-32 (*sampled neighbours = k* means sampling *k* neighbour nodes of the current nodes). It can be seen from Figure 12 that the sampled neighbours have a great influence on the training performance (For ogbn-products training GAT, the memory will be exceeded when *sampled neighbours=32*, so we set sampled neighbours to 24). When the number of sampled neighbours increases from 2 to 16, the convergence accuracy can be greatly improved. However, when the number of sampled neighbours exceeds 16, there is almost no change in accuracy. This is because the neighbours are randomly sampled, and all nodes in the neighbourhood will participate in the training after several epochs. When *sampled neighbours=16*, the neighbourhood of target nodes can be captured effectively. In this situation, the training accuracy is the highest, and the accuracy is no more improved by increasing sampled neighbours.

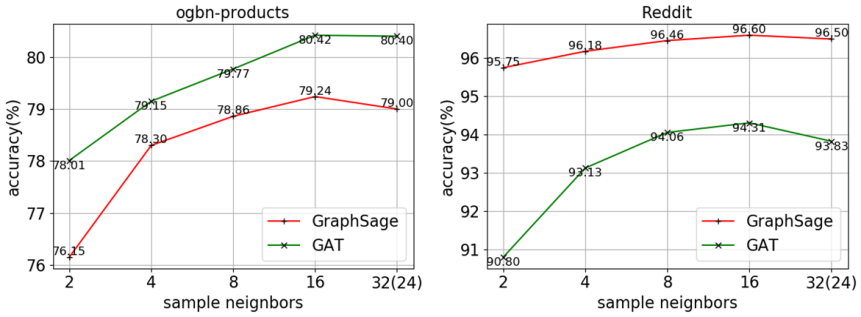


Figure 12. The model accuracy changes with the increasing sampled neighbours.

Figure 13 shows the training epoch time changes as the sampled neighbours increasing. We can notice that when the sampled neighbour doubles, the epoch time increases significantly. This is due to the increment of the sampling time and the model computation time caused by the adding of sampled neighbours. Balancing the performance gains and time consumption, the appropriate sampled neighbours can be chosen. In this way, ADGraph can optimize the convergence accuracy and guarantee training efficiency.

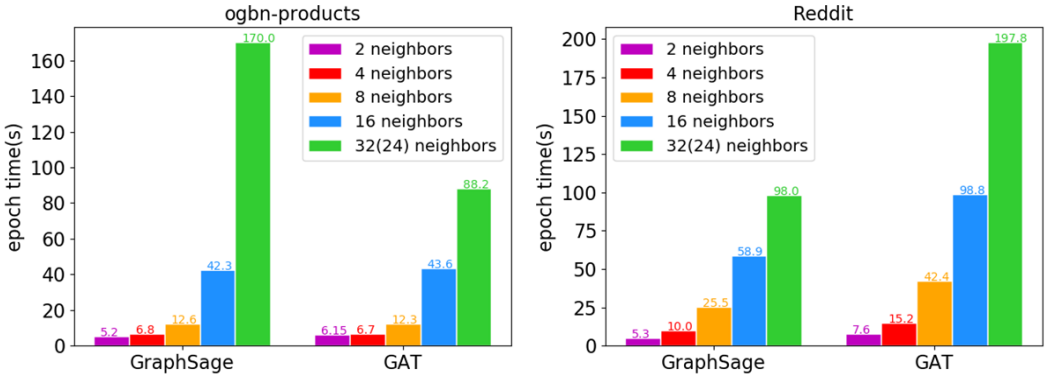


Figure 13. The increment of the run time with the adding sampled neighbours.

4.4. Change Batch Size To Overcome Generalization Error

In distributed GNNs training, the choice of batchsize is significant. In this experiment, we train the GraphSage model on ogbn-products and Reddit datasets with 1 to 32 GPUs. The batchsize on each GPU varies from 10 to 1000. Table 4 and Table 5 list the results. When there are few GPUs, and the batchsize is 10, convergence accuracy on the ogbn-products and Reddit datasets stops at 26.94% and 14.83%, respectively. It can be noticed that small batchsize in the distributed graph model training cannot converge. When there are 1 and 2 GPUs, higher model accuracy can be obtained with 500 and 1000 batchsize, because the large number of training nodes can guarantee enough updates. As the number of machines increases, the number of training nodes on each machine decreases after the graph partition. If larger batchsize is applied, parameter updates on each machine will be reduced, resulting in poor convergence accuracy. Therefore, after graph partitioned, ADGraph can efficiently obtain an accurate model on 8 machines by reducing batchsize (*batchsize = 250*) and increasing the parameters updates of the model.

Table 4. The test accuracy on ogbn-products with the increment of batchsize and GPUs.

Batchsize/GPU	**1**	**2**	**4**	**8**	**16**	**32**
10	26.94	26.94	26.98	30.84	55.83	56.74
50	73.04	76.66	78.11	78.31	76.94	76.36
100	75.84	78.51	79.31	78.19	78.87	78.68
250	79.2	79.34	79.45	79.36	79.36	79.13
500	79.4	79.35	79.02	78.61	77.86	74.52
1000	79.3	79.25	78.61	77.48	76.65	72.8

Table 5. The test accuracy on Reddit with the increment of batchsize and GPUs.

Batchsize/GPU	**1**	**2**	**4**	**8**	**16**	**32**
10	14.83	14.83	14.84	15.04	15.2	15.22
50	14.83	92.6	95.61	95.12	95.23	96.26
100	94.96	95.79	96.4	96.27	96.12	96.32
250	96.47	96.65	96.6	96.56	96.32	96.55
500	96.69	96.68	96.64	96.44	96.14	93.34
1000	96.67	96.54	96.36	96.13	95.45	92.39

4.5. Apply Learning Rate Scaling Rule

Table 6 and Table 7 show the effect of using linear learning rate rule in distributed graph training. GPU=1 is the result of model training with unpartitioned datasets on DGL [39]. When *GPU=32*, ADGraph use the linear scaled learning rate to compare with the fixed learning rate. It can be seen that when applying a fixed learning rate, the model convergence accuracy is poor on multiple GPUs. When training on 32 GPUs, it can achieve similar accuracy to that of a single GPU training with the linear scaled learning rate. Experiments prove that the linear learning rate is effective in real-world graphs.

It is worth noting that when training the GAT model on the ogbn-products dataset and the GraphSage model on the Reddit dataset, the convergence accuracy on 32 GPUs is even higher than that of a single GPU. This proves that distributed graph training of ADGraph has the same convergence performance as single GPU training on DGL.

Table 6. The performance gains of linear scaling rule on ogbn-products.

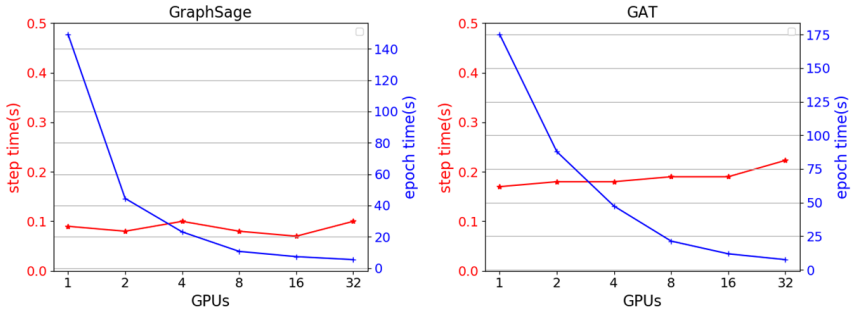
Models	Batchsize*GPUs	Learning rate	Accuracy (%)
GraphSage	250*1	0.003	79.2
	250*32	0.003	74.58
	250*32	0.096	79.13
GAT	250*1	0.0005	79.25
	250*32	0.0005	77.13
	250*32	0.016	80.18

Table 7. The performance gains of linear scaling rule on Reddit.

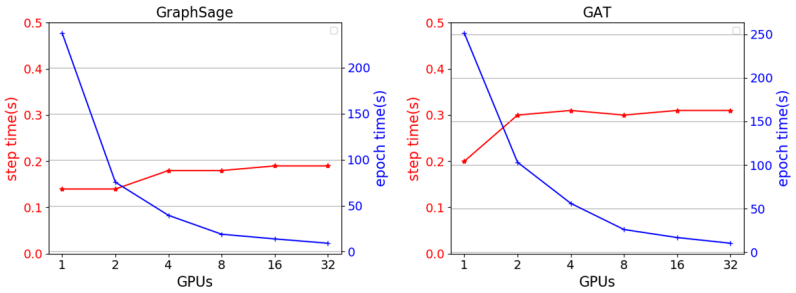
Models	Batchsize*GPUs	Learning rate	Accuracy (%)
GraphSage	250*1	0.0015	96.47
	250*32	0.0015	94.95
	250*32	0.048	96.55
GAT	250*1	0.0005	94.57
	250*32	0.0005	91.5
	250*32	0.016	94.41

4.6. Run Time

Figure 14 shows the curve of epoch time and step time as the GPU increases. The red curve is step time when GPUs changes from 1 to 32 (mini-batch size varies from 250 to 8000). The curve is relatively stable, and the increase in the number of GPUs did not significantly increase step time. The blue curve shows the reduction of each epoch time as the GPUs increases. The overall epoch time is continuously decreasing. In general, the epoch time of 32 GPUs is 24-29 times faster than the epoch time of a single GPU, which can significantly improve distributed graph training efficiency. The maximum scalability efficiency of ADGraph can reach 91%, which is higher than 83% of DistDGL [10] (The result was shown in the experiment of DistDGL).



(a) Training on Reddit dataset



(b) Training on ogbn-products dataset

Figure 14. The curve of epoch time and step time as the GPUs increase.

5. CONCLUSION

In this paper, we present ADGraph for accurate and distributed GNNs training on large graphs. We first used a graph partition method that contains the neighbourhood of the training nodes, and investigated the accuracy and time efficiency of the model training as the number of neighbourhood *hops* increases. Then the complete neighbourhood information of target nodes is obtained through multi-layer neighbourhood sampling. We also analyse the accuracy and runtime performance of graph training, with different l-hop settings. We found that the training time will increase dramatically as the increment of sampled neighbours and sampled layers, but the accuracy of the model not always increases. Then, we explored the influence of batchsize and the number of GPUs on the training of distributed GNNs. The results show that training on the graph partitions needs to reduce the batchsize appropriately. Finally, the linear scaling rule is applied to further improve the training accuracy. The distributed training accuracy can exceed the benchmark accuracy of GraphSage and GAT on DGL. The accuracy of training on 32 GPUs is the same as that of single GPU training, and there is a speedup of 24-29 times.

We have also noticed that there are some shortcomings in the proposed methods. Although the graph partition maintains the integrity of the neighbourhood information of the target nodes, the expanded neighbourhood range is too large, causing the graph partition to lose its meaning. We also found that due to the dependencies between the partitions, the sampling process may communicate with other machines, which will affect the sampling speed. Later, we will study more accurate graph clustering in order to achieve a more reasonable graph partition. Another problem is that the data transfer time from the sampler process to the trainer process often takes up most of the time, resulting in low utilization of computing resources. We are trying data prefetching and caching technology to speed up training and improve the scalability efficiency of distributed training.

ACKNOWLEDGEMENTS

We sincerely thank the anonymous reviewers for their insightful comments and suggestions. This work was supported in part by National Nature Science Foundation of China.

REFERENCES

[1] Z. Guo and H. Wang, "A deep graph neural network-based mechanism for social recommendations," *IEEE Transactions on Industrial Informatics,* vol. 17, no. 4, pp. 2776-2783, 2020.
[2] Z. Xinyi and L. Chen, "Capsule graph neural network," in *International conference on learning representations*, 2018.
[3] D. Szklarczyk *et al.*, "STRING v11: protein–protein association networks with increased coverage, supporting functional discovery in genome-wide experimental datasets," *Nucleic acids research,* vol. 47, no. D1, pp. D607-D613, 2019.
[4] W. Jin, R. Barzilay, and T. Jaakkola, "Junction Tree Variational Autoencoder for Molecular Graph Generation," 2018.
[5] 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.
[6] T. Dettmers, P. Minervini, P. Stenetorp, and S. Riedel, "Convolutional 2D Knowledge Graph Embeddings," 2017.
[7] J. Zhou *et al.*, "Graph Neural Networks: A Review of Methods and Applications," 2018.
[8] Z. Jia, Y. Kwon, G. Shipman, P. Mccormick, and A. Aiken, "A distributed multi-GPU system for fast graph processing," *Proceedings of the VLDB Endowment,* vol. 11, no. 3, pp. 297-310, 2017.
[9] W. Hu *et al.*, "Open Graph Benchmark: Datasets for Machine Learning on Graphs," 2020.
[10] D. Zheng, C. Ma, M. Wang, J. Zhou, and G. Karypis, "DistDGL: Distributed Graph Neural Network Training for Billion-Scale Graphs," 2020.
[11] W. Huang, T. Zhang, Y. Rong, and J. Huang, "Adaptive sampling towards fast graph representation learning," *arXiv preprint arXiv:1809.05343,* 2018.
[12] M. Chen, Z. Wei, Z. Huang, B. Ding, and Y. Li, "Simple and deep graph convolutional networks," in *International Conference on Machine Learning*, 2020: PMLR, pp. 1725-1735.
[13] F. Wu, T. Zhang, A. H. D. Souza, C. Fifty, T. Yu, and K. Q. Weinberger, "Simplifying Graph Convolutional Networks," 2019.
[14] J. Klicpera, A. Bojchevski, and S. Günnemann, "Predict then propagate: Graph neural networks meet personalized pagerank," *arXiv preprint arXiv:1810.05997,* 2018.
[15] J. Du, S. Zhang, G. Wu, J. M. Moura, and S. Kar, "Topology adaptive graph convolutional networks," *arXiv preprint arXiv:1710.10370,* 2017.
[16] H. Dai, Z. Kozareva, B. Dai, A. Smola, and L. Song, "Learning steady-states of iterative algorithms over graphs," in *International conference on machine learning*, 2018: PMLR, pp. 1106-1114.
[17] H. Gao, Z. Wang, and S. Ji, "Large-scale learnable graph convolutional networks," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 1416-1424.
[18] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," *arXiv preprint arXiv:1903.02428,* 2019.
[19] D. Zhang *et al.*, "AGL: a Scalable System for Industrial-purpose Graph Machine Learning," 2020.
[20] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive Representation Learning on Large Graphs," 2017.
[21] L. Ma *et al.*, "Neugraph: parallel deep neural network computation on large graphs," in *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, 2019, pp. 443-458.
[22] C. Tian, L. Ma, Z. Yang, and Y. Dai, "PCGCN: Partition-Centric Processing for Accelerating Graph Convolutional Network," in *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2020.
[23] W. Wei, Y. Wang, P. Gao, S. Sun, and D. Yu, "A Distributed Multi-GPU System for Large-Scale Node Embedding at Tencent," *arXiv preprint arXiv:2005.13789,* 2020.

[24] J. Li, J. Zhu, and B. Zhang, "Discriminative Deep Random Walk for Network Classification," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2016.

[25] P. Goyal *et al.*, "Accurate, large minibatch sgd: Training imagenet in 1 hour," *arXiv preprint arXiv:1706.02677,* 2017.

[26] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A Comprehensive Survey on Graph Neural Networks," *IEEE Transactions on Neural Networks and Learning Systems,* 2019.

[27] P. Velikovi, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph Attention Networks," 2017.

[28] C. Zhang, D. Song, C. Huang, A. Swami, and N. V. Chawla, "Heterogeneous graph neural network," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 793-803.

[29] M. Li *et al.*, "Scaling distributed machine learning with the parameter server," in *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, 2014, pp. 583-598.

[30] M. Li, D. G. Andersen, A. Smola, and K. Yu, "Communication efficient distributed machine learning with the parameter server," *Advances in neural information processing systems,* vol. 1, pp. 19-27, 2014.

[31] R. Johnson and T. Zhang, "Accelerating stochastic gradient descent using predictive variance reduction," in *Advances in neural information processing systems*, 2013, pp. 315-323.

[32] D. Li, Z. Lai, K. Ge, Y. Zhang, and H. Wang, "HPDL: Towards a General Framework for High-performance Distributed Deep Learning," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, 2019.

[33] 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.

[34] H. Robbins and S. Monro, "A stochastic approximation method," *The annals of mathematical statistics,* pp. 400-407, 1951.

[35] S. L. Smith, P.-J. Kindermans, C. Ying, and Q. V. Le, "Don't decay the learning rate, increase the batch size," *arXiv preprint arXiv:1711.00489,* 2017.

[36] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM Journal on scientific Computing,* vol. 20, no. 1, pp. 359-392, 1998.

[37] A. Krizhevsky, "One weird trick for parallelizing convolutional neural networks," *arXiv preprint arXiv:1404.5997,* 2014.

[38] T. Akiba, S. Suzuki, and K. Fukuda, "Extremely large minibatch sgd: Training resnet-50 on imagenet in 15 minutes," *arXiv preprint arXiv:1711.04325,* 2017.

[39] M. Wang *et al.*, "Deep Graph Library: A Graph-Centric, Highly-Performant Package for Graph Neural Networks," 2019.

AUTHORS

Lizhi Zhang, born in 1996, MS candidate, his research interests include distributed machine learning and graph neural network.

E-mail: zhanglizhi15@nudt.edu.cn

Zhiquan Lai, born in 1986, assistant professor, his research interests include distributed machine learning and highly performance system software.

E-mail: laizhiquan@nudt.edu.cn

Zhejiang Ran, born in 1997, MS candidate, his research interests include distributed machine learning and graph neural network.

E-mail: ranzhejiang@163.com

Feng Liu, born in 1977, associate Professor, his research interest includes big data and distributed computing.

E-mail: liufeng@nudt.edu.cn