Grouping Synchronous to Eliminate Stragglers with Edge Computing in Distributed Deep Learning

*Zhiyi Gui, *Xiang Yang, †Hao Yang, *Wei Li, *Lei Zhang, *Qi Qi, *Jingyu Wang, *Haifeng Sun, *Jianxin Liao
*The State Key Laboratory of Networking and Switching Technology,
Beijing University of Posts and Telecommunications, Beijing, China
†Huawei, Beijing, China
{guizhiyi, yangxiang, liwei, zhanglei, qiqi8266, wangjingyu, hfsun, liaojx}@bupt.edu.cn, yanghao30@huawei.com

***Abstract*—With the development of artificial intelligence(AI) applications, a large number of data are generated from mobile or IoT devices at the edge of the network. Deep learning tasks are executed to obtain effective information in the user data. However, the edge nodes are heterogeneous and the network bandwidth is limited in this case, which will cause general distributed deep learning to be inefficient. In this paper, we propose *Group Synchronous Parallel* (GSP), which uses a density-based algorithm to group edge nodes with similar training speeds together. In order to eliminate stragglers, group parameter servers are responsible for coordinating communication of nodes in the group with Stale Synchronous Parallel and aggregating the gradients of these nodes. And a global parameter server is responsible for aggregating the gradients from the group parameter servers to update the global model. To save network bandwidth, we further propose *Grouping Dynamic Sparsification* (GDS). It adjusts the gradient sparsification rate of nodes dynamically based on GSP so as to differentiates the communication volume and makes the training speed of all nodes tend to be the same. We evaluate GSP and GDS's performance on LeNet-5, ResNet, VGG, and Seq2Seq with Attention. The experimental results show that GSP speedups the training by 45%$\sim$120% with 16 nodes. GDS on top of GSP can make up for some test accuracy loss, up to 0.82% for LeNet-5.**

***Index Terms*—distributed deep training, gradient compression, parameter server, gradient sparsification**

I. Introduction

With the development of deep neural networks (DNNs), the application of artificial intelligence in mobile devices with limited resources is developing, such as face recognition [1] [2], virtual/augmented reality (AR/VR) [3] [4], and so on. These applications are computationally intensive and delay-sensitive. If a large amount of data generated from the users' devices is uploaded to the data center, it will bring obvious delay and user privacy leakage. Fortunately, edge computing can solve these concerns by collaborating edge computing nodes (such as base stations, home gateways, edge servers, and roadside units) with computing power.

Distributed deep learning trains the model in parallel by deploying the model to distributed nodes. Among all parallel strategies, data parallelism is a common method. It distributes data equally to each node, and each node trains only a part of the data to overcome the limitations of the computing power and hardware storage of a single device, allowing training tasks executed in parallel. However, in distributed training, especially synchronous training, the most widely used optimization method, stochastic gradient descent (SGD), requires the synchronization of parameters by transmitting gradients among nodes. Therefore, the considerable communication overhead in data parallelism has become an important bottleneck for accelerating training. Simply adding machines will not achieve linear training acceleration. Therefore, the parallelization strategy becomes extremely important in distributed training.

Communication strategies in data parallelism are divided into synchronous and asynchronous. In synchronous data parallelism, all nodes compute gradients based on the same model parameters. This means that in each training iteration, each node has to wait for others until all nodes have completed their computation. Take the most typical synchronous architecture Parameter Server (PS) architecture as an example. Whenever a node completes the gradient computation of the current batch, it will upload the computation result to the parameter server. The parameter server will store all the information until all nodes have completed the model computation and uploaded the gradients. After that, the parameter server will apply a specific update mechanism to merge and then send the gradients. In order to overcome the serious delay problem caused by the high-load nodes in the synchronization strategy, the asynchronous method removes the constraints of the synchronization method [5] [6] [7]. There is no need to wait for each other between nodes and can be updated freely. However, the problem with this is that the parameters are out of date. Parameter staleness refers to the delay time between a node obtaining parameters from the parameter server and submitting parameters to the parameter center next time. Stale Synchronous Parallel [8] [9] [10] is a solution that balances the delay of synchronous methods and parameter staleness in asynchronous methods to a certain extent.

However, in an edge environment, the computing speeds of nodes are different, thus forming a hybrid heterogeneous cluster based on the existing edge devices. None of the above communication strategies can fully utilize the characteristics of the hybrid cluster. What's more, many experiments show that communication overhead is also the bottleneck of large-scale distributed training. Taking data-parallel training of ResNet-50 as an example, the parameter size of ResNet-50 is about 100MB. Tensorflow training on Tesla K80 requires 181.40ms for each iteration, which means 550MB/s bandwidth is required to avoid bandwidth saturation. Considering VGG-16, a bandwidth of 2965MB/s or more is needed. Therefore, in order to overcome this bottleneck, large-scale distributed training requires gradient compression to reduce the number of communication parameters.

In this paper, we focus on how to make full use of heterogeneous computing and limited communication resources to achieve the best distributed training performance. First, we propose Group Synchronous Parallel (GSP), which uses density clustering to select the optimal number of groups, and groups nodes with similar training speeds together. The group parameter servers coordinates the nodes with SSP, and a global parameter server coordinates the group parameter server asynchronously. Moreover, the group parameter server accumulates the gradients of the nodes several steps and transmits the merged gradients to further reduce the communication frequency. After each iteration, a node needs to upload its local gradient to the group server and pull the latest global model. We further propose Grouping Dynamic Sparsification(GDS), which reduces the number of parameters in communication through gradient sparsification. In addition, differentiating the number of parameters uploaded by nodes with different training speeds reduces the difference in training speeds between groups and reduces the deviation of model convergence.

Our critical contributions are summerized as follows:

- We use a density-based clustering method to cluster all edge nodes, which can compute the best number of groups. The clustering speed of this method is faster than traditional iterative-based methods, including k-means. GSP eliminates stragglers through clustering, thereby increasing the training speed of the heterogeneous clusters.
- Based on GSP, we then proposed a grouped dynamic sparsification method, which effectively reduces network bandwidth through gradient sparsification and reduces the difference in training speed between nodes.
- Compared with other distributed training communication strategies, our method achieves acceleration of edge heterogeneous clusters.

The rest of this paper is organized as follows. In Sec.II, we present the background and motivation of this paper. Sec.III presents the architecture of GSP and its mechanism in detail. In Sec.IV, we present GDS method based on GSP. We evaluate the performance of GSP and GDS experimentally in Sec.V. Related work and summary are in Sec.VI and Sec.VII.

II. Background

Stochastic Gradient Descent. Given a neural network model, the goal of deep learning is to find the model parameters ω^* that minimizes the loss function $F(\omega)$ over the training dataset, i.e.,

$$\min F(\omega) = \frac{1}{|D|} \sum_{B \in D} f(B, \omega) \tag{1}$$

here, D is the labeled training dataset, and $f(B, \omega)$ is the loss value when using the parameters ω to predict sample B. For mini-batch SGD [11], we have $\omega_{t+1} = \omega_t - \eta_t g_t$ for iteration t, here η_t is the learning rate.

$$g_t = \frac{1}{|B_t|} \sum_{b \in B_t} \nabla f(b, \omega_t). \tag{2}$$

where g_t is the gradients computed from batch B at iteration t.

In data-parallel distributed training, mini-batch SGD is:

$$g_t = \frac{1}{N} \sum_{n=1}^{N} \cdot \frac{1}{|B_t^n|} \sum_{b_n \in B_t^n} \nabla f(b_n, \omega_t^n) \tag{3}$$

where N is the number of nodes.

Distributed communication strategy. Dai *et al.* [12] studied the convergence of a large number of models and algorithms in the case of delayed updates during asynchronous training, revealed the multiple effects of parameter staleness on the convergence of the algorithm, and proposed a new non-convex optimization SGD convergence analysis method, which can reach the convergence rate of $\frac{1}{\sqrt{T}}$. To compensate for the outdated parameters, Zheng *et al.* [13] proposed Delay Compensation Asynchronous SGD (DC-ASGD), which makes the optimization behavior of asynchronous SGD closer to sequential SGD. They realize this method by using the Taylor expansion of the gradient function and the effective approximation of the Hessian matrix of the loss function. They have performed comparative experiments on a variety of models and datasets and get good performance.

Communication overhead. To reduce the communication overhead caused by transmitting gradients in distributed training, a widely adopted method is gradient compression. Gradient compression is mainly divided into two directions: gradient sparsification [14] [15] and gradient quantization [16] [17]. In gradient sparsification, only part of the gradients is uploaded after filtering all the gradients. Since most of the gradients are close to 0 during the training process, gradient communication frequently is unnecessary. In gradient sparsification, each node only transmits the important gradients in each training step, accumulates the unimportant gradients locally, and waits for the next step. In PS architecture, each node only sends important gradients to the server after backpropagation, that is, *Sparse Uploading Gradient*. The server will aggregate sparse upload gradients from all nodes, and then send the aggregated gradients to the nodes. Locally at each node, unimportant gradients are accumulated and become *Storing Gradients*. After accumulation, the old storage gradients will be uploaded together with the newly generated gradients to form a new sparse upload gradient. Using this method, the compression ratio has been greatly increased. Because in each training step, only a small part of the gradient is involved in the communication. Different from reducing overhead by simply reducing the number of transmitted elements of gradient sparsification, gradient quantization will transmit each element and compress each element. After quantization, the number of bits per element will be reduced. Since quantization still needs to transmit each gradient element, the compression ratio is constrained.

III. Grouping Synchronous Parallel

In this section, we propose Group Synchronous Parallel (GSP) scheme. The edge nodes are density clustered according to the training speed to accelerate the model training in PS architecture. In order to reduce the synchronization waiting time, nodes in the same group execute iteration within a relaxed stale threshold. At the same time, a global parameter server is set to reduce the frequency of communication between different groups. In this way, all nodes communicate in a semi-synchronous manner to achieve a balance between waiting time and staleness.

A. Overview

Our solution decouples the relevance of iteration between nodes with relatively training speed to overcome the serious synchronous delay problem. This is achieved by a two-layer parameter server architecture. Firstly, we use a density-based clustering method to cluster edge nodes to calculate the required number of groups. Then we set P group parameter servers according to the calculated number, and they are responsible for aggregating gradients in the current group and transmitting weights between groups. We also set a global parameter server for refining global model weights. Then, the group parameter servers merge the gradients they receive from the intra-group nodes, and then sends their sum to the global parameter server to obtain the global model to further reduce the frequency of communication. The architecture of GSP is shown as Fig. 1.

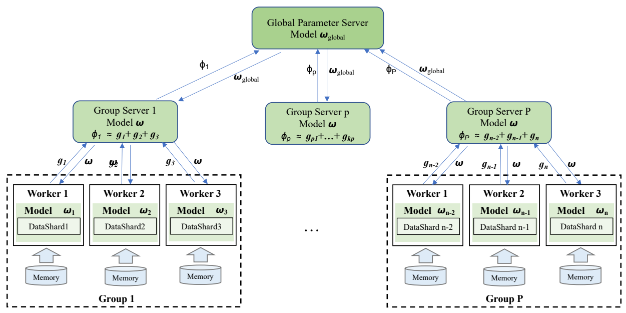


Fig. 1. The two-layer parameter server architecture of grouping synchronous. The intra-group nodes communicate at the pace of SSP, and the different groups communicate asynchronously. For node n, there are local model ω^n and local data D^n, $\{n = 1, ..., N\}$. Due to the existence of the global parameter server, the models of all nodes and servers are the same after synchronizing with the global parameter server.

B. Grouping Algorithm

In a cluster with N edge nodes, we group the nodes according to a density-based clustering method. We define the difference in training speed between nodes as distance dis. For each node n, we compute two metrics: its local density ρ_n and its distance s_n to nodes with higher density. And these metrics only depend on the distance between nodes. The local density ρ_n of node n is defined as:

$$\rho_n = \sum_j \chi(dis_{nj} - dis_{thr}) \tag{4}$$

where dis_{nj} represents the distance between node n and node j. dis_{thr} is the threshold distance. Only if the distance between nodes is less than dis_{thr} will they be considered as neighboring nodes in the same group. We define the function χ as:

$$\chi(x) = \begin{cases} 1, & if\ x < 0; \\ 0, & otherwise. \end{cases} \tag{5}$$

ρ_n is basically equal to the number of nodes whose distance from node n is less than dis_{thr}. s_n is determined by calculating the minimum distance between node n and any other node with higher density:

$$s_i = \min_{j:\rho_j > \rho_n} (dis_{nj}) \tag{6}$$

For node with the highest density, we usually take $s_i = max_j(dis_{nj})$. Therefore, the center of the cluster is a node with an abnormally large s.

we can set any number of groups. If the number is less than P, the node with higher ρ and s will be preferentially set as the group center. If the number is greater than P, all workers with $s > s^0$ will be set as the center of the cluster, and other workers with higher ρ will be given priority to become the the center of the cluster. After obtaining the centers of cluster, the remaining points are allocated to the same cluster as its nearest neighbor, and the density is higher. Algorithm 1 shows the details of finding the optimal number of groups P and the detail information of the grouping process.

Based on density clustering, we get the optimal number of groups P. The p_{th}, $\{p = 1, 2, ..., P\}$ group parameter server is responsible for k_p nodes in its group, and the total number of nodes meets $\sum_{p=1}^{P} k_p = N$. The global parameter server aggregates the models from all the group parameter servers asynchronously to obtain the global model. Moreover, the number of communications between groups should be reduced to further reduce communication overhead. For the group parameter server of group p, the gradient of this group is uploaded to the global parameter server every k_p times, so that each node has approximately the same gradient contribution to the global model. Through merging gradients, the number of gradient transmissions of the p_{th} group server can be reduced to $\frac{1}{k_p}$ of the original. Extending to the cluster, the communication times of all groups can be reduced to the $\frac{p}{n}$ of the original, where N is the number of nodes, and P is the number of the optimal group obtained by Algorithm 1. And we summarize GSP as Algorithm 2.

C. Convergence Analysis

We prove the convergency rate of GSP. In Sequential SGD, $g(\omega_t)$ should be added to ω_t at step t, that is, $\omega_t = \omega_{t-1} - \eta g(\omega_t)$. However, in GSP, the global models some may have updated by other edge nodes τ times. So the update formula

Algorithm 1 Finding optimal P and grouping process

```
1: Finding optimal $P$
2: input: $G = (K, L), dis_{thr}$, $G(K, L)$ represents a set of workers $K$ and all connections $L$ between each two nodes.
3: $P$ = 0
4: for $k$ in $K$ do
5:     $\rho_k$ = GetNodesCountWithinDistance($k, dis_{thr}, G$)
   end for
6: for $k$ in $K$ do
7:     $s_k$=MinDistanceToHigherDensityNodes($k, \rho, G$)
8:     $s^0 = \frac{1}{|K|}\sum_{k \in K} s_k$
9:     if $s_k > s^0$ then
10:        $P$ += 1
   end for
11: return $P$
12:
13: Grouping process
14: input: $G = K, L, \rho, s$
15: input: the number of groups P
16: $\max_P K$ = FindNodesAsClusterCenter($P, \rho, s, G$)
17: for $k$ in $K$ do
18:     if $k \in \max_P K$ then
19:         $k$ belongs to a new group
20:     else
21:         $k$ belongs to the nearest high-density node group
   end for
```

Algorithm 2 Grouping Synchronous Parallel (GSP)

```
Node $n$ in group $p$
1: Once receiving Ready from its group PS $p$
2: pull group $p_th$ weights $\omega^p$ from group PS
3: $\omega_t^n$ = $\omega^p$
4: $g_t^n = \frac{1}{|B_t^n|}\sum_{b_n \in B_t^n} \nabla f(b_n, \omega_t^n)$
5: push $g_t^n$ to group PS
Group PS $p$
  - $\phi^p$ is the accumulated gradients for $p_{th}$ group
  - $counter^p$ counts local accumulations for $p_{th}$ group
1: receive $g_t^n$ from node $n$
2: wait until SSPScheduler(group $p$, node $n$) is Ready
3: if $k_p$ = $counter^p$ then
4:     $\phi^p$ = $\phi^p + g_t^n$
5:     send($\phi^p$) to global PS
6:     set $\phi^p = 0$ and $counter^p = 0$
7:     $\omega^p$ = $\omega^{global}$
8: else
9:     $\phi^p$ = $\phi^p + g_t^n$
10:    $counter^p = counter^p + 1$
   end if
11: return Ready
12:
13: procedure SSPScheduler(group $p$, node $n$)
14:     if node $n$ is not the fastest or within the threshold then
15:         Return Ready
        end if
```

is $\omega_{t+\tau+1} = \omega_{t+\tau} - \eta_{t+\tau} g(\omega_t)$. To analyze convergence, we make the following assumptions.

Assumption 1 (smoothness and strongly convex): Loss function $F(\omega)$ is $L_2-smooth$ and μ-strongly convex about ω. $\nabla F(\omega)$ is $L_3-smooth$ about ω and a constant H is the upper bound of the expection of the $\|\cdot\|_2^2$ norm of the gradients.

Assumption 2: Since most of the gradient elements are close to 0 [14], we bound $\|\nabla F(\omega_t) - g(\omega_t)\| \leq G/t$, where G is some constant.

Theorem 1: Set $\eta_t = \frac{L_2}{\mu^2 t}$. With Assumptions 1 and 2, the convergence rate of GSP is

$$\mathbb{E}F(\omega_t) - F(\omega^*) \leq \frac{L_3 L_2^3 H^3 \tau^2}{\mu^6 t^2} + \frac{L_2 H G}{\mu^2 t} + \frac{L_2^3 H^2 \tau^2}{\mu^4 t} \quad (7)$$

Proof:

Using $\mathbb{E}g(\omega_t) = \nabla F(\omega_t)$ and the smoothness condition, we have

$$\begin{aligned}&\mathbb{E}F(\omega_{t+\tau+1}) - F(\omega^*)\\ &\leq F(\omega_{t+\tau}) - F(\omega^*) + F(\omega_{t+\tau+1}) - F(\omega_{t+\tau})\\ &\leq F(\omega_{t+\tau}) - F(\omega^*) + \langle \nabla F(\omega_{t+\tau}), \omega_{t+\tau+1} - \omega_{t+\tau}\rangle\\ &\quad + \frac{L_2}{2}\|\omega_{t+\tau+1} - \omega_{t+\tau}\|^2\\ &\leq F(\omega_{t+\tau}) - F(\omega^*) - \eta_{t+\tau}\langle \nabla F(\omega_{t+\tau}), g(\omega_t)\rangle + \frac{L_2 \eta_{t+\tau}^2 H^2}{2}\end{aligned} \quad (8)$$

The term $-\langle \nabla F(\omega_{t+\tau}), g(\omega_t)\rangle$ in (8) can be expressed as

$$\begin{aligned}&-\langle \nabla F(\omega_{t+\tau}), \nabla F(\omega_{t+\tau})\rangle + \langle \nabla F(\omega_{t+\tau}), \nabla F(\omega_{t+\tau})\\ &\quad - \nabla F(\omega_t)\rangle + \langle \nabla F(\omega_{t+\tau}), \nabla F(\omega_t) - \nabla g(\omega_t)\rangle\end{aligned} \quad (9)$$

we analyze each term in (9) one by one.

Using Assumption 1, we have

$$\begin{aligned}&-\langle \nabla F(\omega_{t+\tau}), \nabla F(\omega_{t+\tau})\rangle \leq -\mu^2\|\omega_{t+\tau} - \omega^*\|^2\\ &\leq -\frac{2\mu^2}{L_2}(F(\omega_{t+\tau}) - F(\omega^*))\end{aligned} \quad (10)$$

Using $\nabla F(\omega_t)$ is L_3-smooth (i.e., $\|\nabla F(\omega_{t+\tau}) - \nabla F(\omega_t)\| \leq \frac{L_3}{2}\|\omega_{t+\tau} - \omega_t\|$), we have

$$\begin{aligned}&\langle \nabla F(\omega_{t+\tau}), \nabla F(\omega_{t+\tau}) - \nabla F(\omega_t)\rangle\\ &\leq \|\nabla F(\omega_{t+\tau})\|\|\nabla F(\omega_{t+\tau}) - \nabla F(\omega_t)\|\\ &\leq \frac{L_3 H}{2}\|\omega_{t+\tau} - \omega_t\|^2 \leq \frac{L_3 \tau H^3}{2}\sum_{j=0}^{\tau-1}\eta_{t+j}^2\\ &\leq \frac{L_3 L_2^2 H^3 \tau^2}{2\mu^4 t(t+\tau)}\end{aligned} \quad (11)$$

Under Assumption 2, we have

$$\begin{aligned}&\langle \nabla F(\omega_{t+\tau}), \nabla F(\omega_t) - g(\omega_t)\rangle\\ &\leq \|\nabla F(\omega_{t+\tau})\| \; \|\nabla F(\omega_t) - g(\omega_t)\|\\ &\leq H\|\nabla F(\omega_t) - g(\omega_t)\| \leq \frac{HG}{t}\end{aligned} \quad (12)$$

Putting inequality (10)-(11) in inequality (8), we have

$$\mathbb{E}F(\omega_{t+\tau+1}) - F(\omega^*) \leq (1 - \frac{2}{t+\tau})(\mathbb{E}F(\omega_{t+\tau}) - F(\omega^*)) + \frac{L_3 L_2^3 H^3 \tau^2}{2\mu^6 t(t+\tau)^2} + \frac{L_2 HG}{\mu^2 t(t+\tau)} + \frac{L_2^3 H^2 \tau^2}{2\mu^4 (t+\tau)^2}. \tag{13}$$

In the end, we get

$$\mathbb{E}F(\omega_t) - F(\omega^*) \leq t(\frac{L_3 L_2^3 H^3 \tau^2}{2\mu^6 t^3} + \frac{L_2 HG}{\mu^2 t^2} + \frac{L_2^3 H^2 \tau^2}{2\mu^4 t^2}) \leq \frac{L_3 L_2^3 H^3 \tau^2}{\mu^6 t^2} + \frac{L_2 HG}{\mu^2 t} + \frac{L_2^3 H^2 \tau^2}{\mu^4 t} \tag{14}$$

by induction. Theorem 1 is proved.

IV. Grouping Dynamic Sparsification

In distributed training, in addition to the difference in computation speed between nodes, the communication overhead when multiple machines synchronize parameters is also a bottleneck for accelerating training. Gradient compression is used to reduce the number of parameters that need to be transmitted. Under the traditional synchronization method, all distributed nodes have the same sparsification ratio. However, with GSP, the nodes are relatively asynchronous, and the training speed among different groups is very different, and the convergence of the model will be affected by outdated parameters. We propose Grouping Dynamic Sparsification (GDS) to differentiate their communication time by setting different sparsification ratios for nodes in different groups so that they have similar total iteration time, including computation time, compression time, and communication time. For compression time, it is necessary to upload the gradients and corresponding coordinates according to the calculated threshold, regardless of the sparsification ratio. Therefore, the compression time is only related to the computation performance. For communication time, under the same network bandwidth conditions, a larger sparsification ratio means shorter communication time.

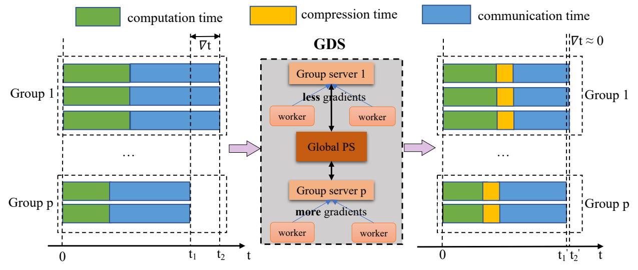


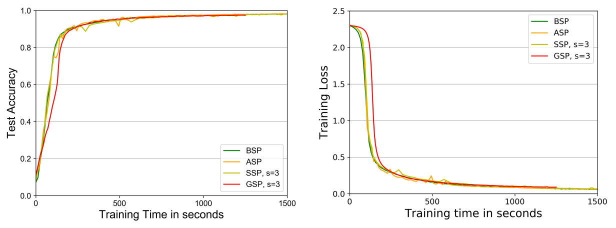
Fig. 2. Grouping Dynamic sparsification shorten the iteration gap among groups and reduce communication time. Take group 1 and group p as an example. The nodes in group 1 are stragglers for other nodes because the computation speed of group 1 is the slowest. GDS on top of GSP eliminates stragglers by reducing the communication time of group 1. After GDS, the iteration time of all nodes is nearly the same.

More specifically, we set the basic sparsification ratio as s_ϵ, and the maximum sparsification ratio as s_ϵ^{max}. The maximum sparsification ratio is to ensure that each node uploads its local gradients. For the group with the slowest computation speed, we set its sparsification ratio to s_ϵ with computation time $time_\epsilon^{slowest}$. For other group, the computation time is $time_\epsilon^p$. Thus, their sparsification ratio is $min(s_\epsilon * time_\epsilon^p / time_\epsilon^{slowest}, s_\epsilon^{max})$. Since uploading sparse gradients and its coordinates, the uploaded parameter amount is $\sum_{p=1}^{P} 2 \cdot s_\epsilon^p \cdot |g_t^n| + C_0$, where $|g_t^n|$ is the size of gradient elements and C_0 is a small constant representing the shape of the original gradients.

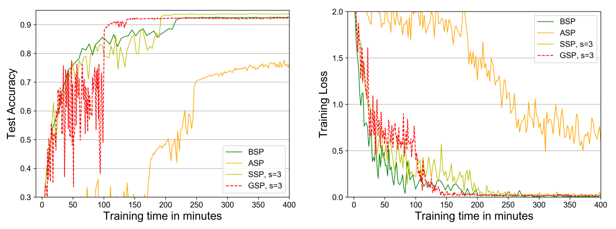
The training speed is unstable affected by the fluctuation of node physical performance and network bandwidth. So we add a momentum to estimate the training time of the next iteration based on the historical training time. Moreover, in order to improve the efficiency of gradient compression, we first sample a fixed proportion of nodes to compute the threshold for each group roughly. GDS on top of GSP sparses out some unimportant and stale local parameters and solve the problem of model convergence deviation.

Algorithm 3 Grouping Dynamic Sparsification (GDS)

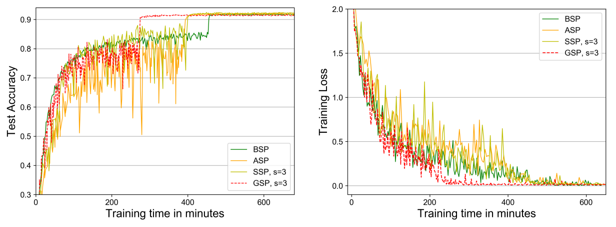
```
1:  $m_t^p$ is the number of gradients for nodes in $p_{th}$ group at iteration $t$;
2:  $s_\epsilon$, $s_\epsilon^{max}$ are the basic upload ratio and the maximum upload ratio respectively at epoch $\epsilon$;
3:  $time_\epsilon^p$ is the average training time in $p_{th}$ group at epoch $\epsilon$;
4:  $time_\epsilon^{slowest}$ is the slowest group respectively at epoch $\epsilon$.
5:  Set $g_0^n = 0$
6:  for $t = 1 \to T$ do
7:      $g_t^n = g_{t-1}^n + \nabla f(b, \omega_t^n)$;
8:      Set threshold $thr$ = the $(m_t^p)^{th}$ lagest value of $|g_t^n|$
9:      $\widetilde{Mask} = |g_t^n| > thr$
10:     $\widetilde{g_t^n} = g_t^n \odot Mask$;
11:     $g_t^n = g_t^n \odot \neg\widetilde{Mask}$;
12:     $\omega^p = \omega^p - \eta\widetilde{g_t^n}$
    end for
13: CompressController
14: if $\epsilon == 0$ then
15:     $\widetilde{time_\epsilon^p} = time_\epsilon^p$
16: else
17:     $\widetilde{time_\epsilon^p} = \widetilde{time_{\epsilon-1}^p} * \beta + time_\epsilon^p * (1-\beta)$        ▷ $\beta$ is a momentum factor
    end if
18: $\widetilde{time_\epsilon^{slowest}} = max(\widetilde{time_\epsilon^1}, ..., \widetilde{time_\epsilon^P})$
19: if $\widetilde{time_\epsilon^p} = \widetilde{time_\epsilon^{slowest}}$ then
20:     $s_\epsilon^p = s_\epsilon$                    ▷ group $p$ is the slowest
21: else
22:     $s_\epsilon^p = min(s_\epsilon * \widetilde{time_\epsilon^p} / \widetilde{time_\epsilon^{slowest}}, s_\epsilon^{max})$
23:                                     ▷ group $p$ is not the slowest
    end if
24: return $m_t^p = s_\epsilon^p \cdot |g_t^n|$
25:                  ▷ $|g_t^n|$ is the size of gradient elements
```



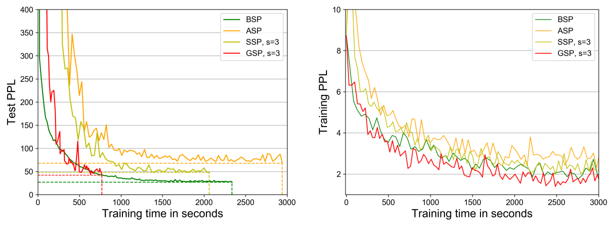
(a) LeNet-5 on MNIST. The accuracy of GSP is slightly lower than other distributed communication strategies.



(b) ResNet-18 on CIFAR-10. GSP is about 2x faster than BSP and SSP to reach 92.0% accuracy, with an accuracy drop of about 0.9%.



(c) VGG-19 on CIFAR-10. GSP reaches 91% accuracy faster than other distributed communication strategies.



(d) Test PPL of Seq2Seq with Attention on Multi30k. Compared with other distributed paradigms, GSP is the fastest and has lower PPL than ASP ans SSP.

Fig. 3. Distributed communication paradigms comparison on multiple deep learning tasks.

V. Experiments and Evaluation

A. Experimental Setup

1) Hardware and software: To evaluate the effectiveness of GSP, we use the clusters with edge nodes varying from 4 to 32. The node group consists of mobile devices with GPU and laptop computers, which are connected with 1Gbps Ethernet. We implement all distributed communication strategies in PS architecture with Ray [18].

2) Models and datasets: We trained our methods on deep learning tasks, including computer version and natural language processing. For image classification tasks, we use momentum SGD in all models. We set the hyperparameters as Table.I.

TABLE I
Parameters of GSP

Model	Dataset	Batch size / node	learning rate	lr decay	epochs
LeNet-5	MNIST	64	0.001	–	200
ResNet-18	Cifar-10	64	0.1	0.1 at epoch 50 and 70	200
VGG-19	Cifar-10	64	0.01	0.1 at epoch 70 and 100	200
Seq2seq with Attention	Multi30k	32	–	–	until convergence

3) Metrics: We trained all DL tasks on BSP, ASP, SSP and our method. For stale synchronous, we set the staleness threshold to 3. For the image recognition tasks, we compare the training time and test accuracy; For machine translation tasks, we compare training time and test perplexity (PPL). We compared the iteration gap among groups, acceleration speedup and gradient compression ratio (GCR) for GDS experiments.

B. Results in GSP

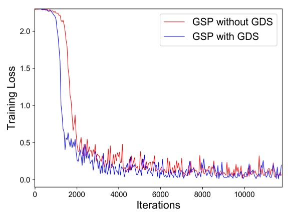
The training curves of different communication methods on 16 edge nodes are shown in Fig.3. Fig.3(a) shows the experimental effect of LeNet-5 on MNIST dataset, comparing baseline (BSP), ASP, SSP and our GSP method. Fig.3(b) shows the experimental effect of Resnet-18 on Cifar-10 dataset, comparing baseline, ASP, SSP and GSP. Fig.3(c) shows the performance of VGG-19 in cifar-10 dataset when baseline, ASP, SSP and GSP are used. Fig.3(d) shows the performance of Seq2Seq with Attention on the Multi30k dataset when using baseline, ASP, SSP and GSP.

Let's look at the accuracy and convergence speed of the model. Compared with baseline, GSP achieved 45%∼120% convergence acceleration on the premise that the test accuracy decreased by no more than 1.1%. Our explanation is that the improvement of convergence speed may come from two aspects. Firstly, it reduces the negative impact of stragglers on the training speed of other nodes, so as to make full use of the computing resources of nodes. Secondly, the group parameter server accumulates gradients, thus reducing the frequency of communication, and total communication time.

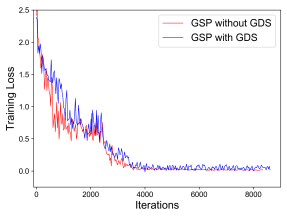
C. GSP with GDS

To evaluate GDS on top of GSP, we set the same hyper-parameters in Table.I as GSP. Similar to [15], We also set the sample ratio to 0.01 for sampling about 1% gradients and set the basic sparsification ratio and the maximum maximum sparsification ratio as 99.6% and 98.5% respectively. 99.6% exponentially increases from 75%, 93.75%, 98.4375% in the warm-up period. And 98.5% exponentially increases from 65%, 87.75%, 95.7125% in the warm-up period.

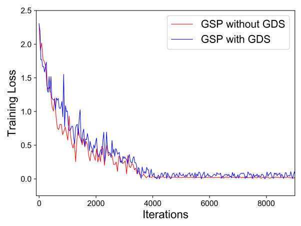
1) group gap with GDS. Fig.4 depicts that for LeNet-5, ResNet-18 and VGG-19, the group gaps can be reduced to 75%, 8% and 14% respectively. For LeNet-5, GDS on top of GSP has faster convergence speed. For ResNet-18 and VGG-19, the training curves jitter slightly. For Seq2Seq with Attention, the group gaps and training loss with GDS both are almost the same as GSP.



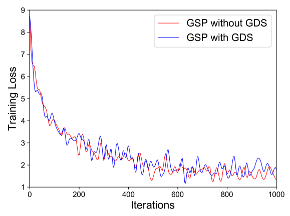
(a) LeNet-5 on MNIST



(b) ResNet-18 on CIFAR-10



(c) VGG-19 on CIFAR-10



(d) Seq2Seq with Attention on Multi30k

Fig. 4. Training curves of all models

2) Test accuracy. Table II demonstrated GDS improves test accuracy and reduces test PPL compared to simple GSP in all experiments. We attribute the increase in accuracy and the decrease in PPL to one reason: most of the gradients generated in each step of training are useless for the convergence of the model, and even some outdated parameters may affect the convergence of the model. Gradient sparsification did not upload these gradients to help generalization.

3) Convergence speed. Compared to SSP, GSP speedup

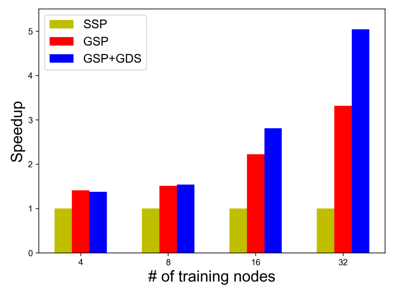


Fig. 5. Training speedup with 1Gbps Ethernet

training process with 1Gbps Ethernet on multiple training nodes, as shown in Fig.5. GSP achieved a certain acceleration because it improves the utilization of computing resources. After using GDS, the communication computation ratio $\frac{communication\ time}{computation\ time}$ is greatly reduced. Therefore, the acceleration and expansion ability of distributed training are greatly improved, especially when the network condition is bad.

4) Compression ratio. We calculate the gradient compression ratio (GCR) to evaluate the reduction of communication volume:

$$GCR = \frac{size[\sum_{p=1}^{P}\sum_{i=0}^{k_p}\widetilde{g_t^i} + C_0]}{size(\sum_{n=1}^{N} g_t^n)},$$

where C_0 is a small constant related to the model, describing the number of model layers and the shape of each layer before sparsification, so that the gradients can be restored on the group PS by the sparse gradients and corresponding coordinates.

TABLE II
COMPRESSION RATIO OF DIFFERENT TRAINING TASKS.

Model	Method	Accuracy/PPL	Gradient size	GCR	Group gap with GDS
LeNet-5	GSP	97.76%	246.8KB	1 x	–
	GSP+GDS	98.58%	10.96KB	56.3 x	75%
ResNet-18	GSP	92.54%	44.68MB	1 x	–
	GSP+GDS	92.75%	0.80MB	55.48 x	8%
VGG-19	GSP	91.53%	155.84MB	1 x	–
	GSP+GDS	92.27%	2.74MB	56.81 x	14%
Seq2Seq	GSP	42.0995	82.07MB	1 x	–
	GSP+GDS	36.8331	1.444MB	56.82 x	1

VI. RELATED WORK

1) Heterogeneous memory. In order to solve the data allocation problem of multi-core systems, Guo *et al.* [19] proposed a polynomial time algorithm, which effectively reduces the memory access time overhead and energy consumption of the data allocation problem with exclusive data copies. Zhang *et al.* [20] decoupled the two issues of variable partitioning and task scheduling of the VS-SPM architecture, and proposed High Access Frequency First variable partitioning and Global View Prediction variable partitioning, and proposed a loop pipeline scheduling algorithm to effectively improve the overall throughput and average performance. In order to minimize the total cost, including energy and latency of memory access, Qiu *et al.* [21] proposed a hybrid SPM architecture that generates high access performance with low power consumption and Multidimensional Dynamic Programming Data Allocation that reduces memory access latency and power consumption strategies. And they then designed the Adaptive Genetic Algorithm for data allocation to minimize the accuracy lost.

2) Distributed communication backend. In the process of data packet processing with the traditional TCP/IP technology, data is copied and moved back and forth between the system memory, processor cache, and network controller cache, causing a heavy burden on the server's CPU and memory. RDMA is a host-offload, host-bypass technology. DMA is used when two or more computers are communicating, and the memory of one host is directly accessed from the memory of another host, thereby quickly moving data from one system to the remote system's memory. RNIC with an RDMA engine is responsible for managing the reliable connection between the source and the target. The advantages of RDMA include zero-copy, kernel bypass, no CPU involvement, message-based transactions and scatter/gather entries support. Gloo is a collective communication library proposed by Facebook, with an implementation that can be used for system memory buffers, and another for NVIDIA GPU memory buffers. Gloo is a collective algorithm that can be executed in parallel on two or more processes or machines. Multiple machines first need to find each other to be able to execute across them. Once they find each other, these machines can establish connections with each other in a complete network (each machine has a two-way communication channel to others) or in a subset. The required connection between the machines depends on the type of algorithm used. NCCL is a multi-GPU collective communication library implemented by Nvidia. NCCL is optimized and compatible with MPI, and can perceive GPU topology, promote multi-node with multi-GPU acceleration, maximize bandwidth utilization within the GPU, and make full use of all available GPUs within multiple nodes or between cross-border points. NCCL uses a communication ring to move data and reduce data among all GPUs, and achieve higher communication speeds on PCIe, Nvlink, and InfiniBand.

VII. Conclusion

In this paper, GSP improves the speed of existing distributed communication strategies by 45%$\sim$ 120% for deep learning tasks. To achieve acceleration without reducing the test accuracy, GSP uses a density-based clustering method to group nodes with similar training speed and sets group parameter servers and a global parameter server for aggregating the gradients. We further propose GDS to bridge the iteration gap among groups and reduce communication overhead. Our method improves the computing utilization of edge devices, saves network bandwidth, and improves the acceleration and scalability of distributed training in heterogeneous clusters.

References

[1] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 815–823, 2015.

[2] O. M. Parkhi, A. Vedaldi, and A. Zisserman, "Deep face recognition," in *BMVC*, 2015.

[3] J. Steuer, "Defining virtual reality: dimensions determining telepresence," 1992.

[4] R. Yung and C. Khoo-Lattimore, "New realities: a systematic literature review on virtual reality and augmented reality in tourism research," *Current Issues in Tourism*, vol. 22, pp. 2056 – 2081, 2019.

[5] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *ICML*, 2016.

[6] X. Lian, Y. Huang, Y. Li, and J. Liu, "Asynchronous parallel stochastic gradient for nonconvex optimization," in *NIPS*, 2015.

[7] M. Li, D. Andersen, A. Smola, and K. Yu, "Communication efficient distributed machine learning with the parameter server," in *NIPS*, 2014.

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

[9] H. Cui, J. Cipar, Q. Ho, J. Kim, S. Lee, A. Kumar, J. Wei, W. Dai, G. Ganger, P. Gibbons, G. Gibson, and E. Xing, "Exploiting bounded staleness to speed up big data analytics," in *USENIX Annual Technical Conference*, 2014.

[10] M. Zinkevich, A. Smola, and J. Langford, "Slow learners are fast," in *NIPS*, 2009.

[11] O. Dekel, R. Gilad-Bachrach, O. Shamir, and L. Xiao, "Optimal distributed online prediction using mini-batches," *J. Mach. Learn. Res.*, vol. 13, pp. 165–202, 2012.

[12] E. Xing, Q. Ho, W. Dai, J. Kim, J. Wei, S. Lee, X. Zheng, P. Xie, A. Kumar, and Y. Yu, "Petuum: A new platform for distributed machine learning on big data," *IEEE Transactions on Big Data*, vol. 1, pp. 49–67, 2015.

[13] S. Zheng, Q. Meng, T. Wang, W. Chen, N. Yu, Z. Ma, and T. Liu, "Asynchronous stochastic gradient descent with delay compensation," in *ICML*, 2017.

[14] A. F. Aji and K. Heafield, "Sparse communication for distributed gradient descent," in *EMNLP*, 2017.

[15] Y. Lin, S. Han, H. Mao, Y. Wang, and B. Dally, "Deep gradient compression: Reducing the communication bandwidth for distributed training," in *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, 2018.

[16] W. Wen, C. Xu, F. Yan, C. Wu, Y. Wang, Y. Chen, and H. Li, "Terngrad: Ternary gradients to reduce communication in distributed deep learning," in *NIPS*, 2017.

[17] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic, "Qsgd: Communication-efficient sgd via gradient quantization and encoding," in *NIPS*, 2017.

[18] P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang, M. Elibol, Z. Yang, W. Paul, M. I. Jordan, and I. Stoica, "Ray: A distributed framework for emerging AI applications," in *13th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2018, Carlsbad, CA, USA, October 8-10, 2018*, 2018, pp. 561–577.

[19] Y. Guo, Q. Zhuge, J. Hu, M. Qiu, and E. Sha, "Optimal data allocation for scratch-pad memory on embedded multi-core systems," *2011 International Conference on Parallel Processing*, pp. 464–471, 2011.

[20] L. Zhang, M. Qiu, W.-C. Tseng, and E. Sha, "Variable partitioning and scheduling for mpsoc with virtually shared scratch pad memory," *Journal of Signal Processing Systems*, vol. 58, pp. 247–265, 2010.

[21] M. Qiu, Z. Chen, and M. Liu, "Low-power low-latency data allocation for hybrid scratch-pad memory," *IEEE Embedded Systems Letters*, vol. 6, pp. 69–72, 2014.