Self-Adaptive Task Allocation for Decentralized Deep Learning in Heterogeneous Environments

1st Yongyue Chao
Institute of Automation, Chinese Academy of Sciences
Beijing, China
chaoyongyue2020@ia.ac.cn

2nd Mingxue Liao
Institute of Automation, Chinese Academy of Sciences
Beijing, China
mingxue.liao@ia.ac.cn

3rd Jiaxin Gao
Institute of Automation, Chinese Academy of Sciences
Beijing, China
jiaxin.gao@ia.ac.cn

4th Guangyao Li
Institute of Automation, Chinese Academy of Sciences
Beijing, China
liguangyao2020@ia.ac.cn

***Abstract*—The demand for large-scale deep learning is increasing, and distributed training is the current mainstream solution. Decentralized algorithms are widely used in distributed training. However, in a heterogeneous environment, each worker computes the same amount of training data resulting in a lot of wasted time for waiting the straggler. In this paper, we proposed a self-adaptive task allocation algorithm (SATA) which allows that each worker acquires the amount of training data adaptively based on the performance of workers in the heterogeneous environment. In order to show the applicability of SATA in heterogeneous clusters better, we set up the heterogeneous cluster composed of two or three different types of GPUs. Besides, we conduct a series of experiments to show the performance of SATA. The experimental results illustrate several advantages of SATA. SATA can accelerate distributed training about 3.3X that of All-Reduce as well as about 1.9X-3.8X that of AD-PSGD algorithm. And the total training time of SATA is reduced from 20 to 40 percentage compared to All-Reduce.**

***Index Terms*—distributed training, task allocation, All-Reduce, heterogeneity**

I. Introduction

As the state-of-the-art artificial intelligent approach, Deep Neural Networks (DNNs) play an important role in various applications, such as natural language process and computer vision. The increasing number of large-scale DNN model parameters provide high accuracy for complicated problems. However, training large-scale DNNs takes a long time and requires a large amount of memory, which is hard to train by single machines. Therefore, distributed training is the most popular method to alleviate the problem.

In distributed training, the model is trained by multiple workers running on a GPU cluster. The most common idea is data parallelism [1] in distributed training. All training data are assigned to workers for computing gradients and then workers aggregate gradients among others. Centralized and decentralized training with stochastic gradient descent (SGD) are the main approaches of data parallelism. One of the centralized approaches, Parameter Server (PS) [2], uses several nodes as servers, which collect the gradients from workers as well as update the model. However, this leads to inevitable communication bottleneck problems. Decentralized approaches resolve these difficulties based on All-Reduce [3]. Each worker only communicates with its neighbors to obtain the average model.

Although these approaches achieve wonderful results in homogeneous environments, the straggler problem still appear in heterogeneous environments. During the process of model training, each worker holds the same amount of training data and needs to wait for others aggregating gradients or parameters, so the whole training speed depends on the slowest worker in heterogeneous environments which contain different types of GPUs and different network bandwidths in distributed training. These heterogeneous environments result in slowing down the training speed of clusters with the straggler. In order to eliminate the effect of the straggler, many studies update the model based on asynchronous SGD instead of synchronous SGD. However, asynchronous SGD has lower accuracy and lower usage in distributed training. Another way is assigning tasks to each worker reasonably. The most common method for assigning tasks is to adjust mini-batch size of each worker and reassign dataset at every epoch. Some papers study the algorithms in self-adaptive task allocation. However, most of them are contributed to centralized training rather than decentralization based on the rough empirical speculations.

In this paper, we proposed a self-adaptive task allocation algorithm (SATA) for decentralized training. We implement our algorithm based on Ring All-Reduce. On the basis of algorithm, the amount of mini-batch data is balanced adaptively only by training information. We establish a mathematical model to describe the ratio of mini-batch size to global batch size in the current epoch for task allocation. To show the algorithm, we implement SATA based on the Ring All-Reduce algorithm in experiments. We set a series of experiments to show the improvement of training speed. Besides, we train a simple convolutional neural network model on MNIST dataset as well as some complex models including ResNet and VGG on CIFAR10 dataset. Experimental results show that self-adaptive task allocation algorithm can reduce 20

DOI reference number: 10.18293/SEKE2022-071

to 40 pecentage of training time compared to All-Reduce. And SATA can accelerate distributed training about 3.3X that of All-Reduce. Besides, SATA make progress in speedup compared to other algorithms.

II. Related Work

The straggler problem is very common in heterogeneous environments. It occurs due to the performance difference among workers and the discrepancy of communication speed and bandwidth. Existing efforts on the straggler problem can be classified into two types: algorithms based on task allocation and algorithms based on model averaging.

For algorithms based on adaptive task allocation, Yang et al. [4] proposed a batch orchestration algorithm, which balances the amount of mini-batch data according to the speed of workers. It makes a linear regression on training time and mini-batch size so that allocates tasks for workers based on slope changing. The weakness is that it introduces redundant training information to balance tasks. FlexRR [5] addresses the straggler problem by integrating flexible consistency bounds with temporary peer-to-peer work reassignment. FlexRR increases the cost of training due to extra communication in worker computing. The current studies are mostly apply for PS and based on empirical speculations.

For algorithms based on model averaging, countermeasures for synchronization are utilized, including asynchronous execution, bounded staleness, backup workers, adjusting the learning rate of stale gradients and so on. AD-PSGD [6], Partial All-Reduce [7] and gossip SGP [8] improve global synchronization with partial random synchronization. Chen et al. [9] proposed to set backup workers in the cluster, which allow gradient aggregations without all workers participation. DYNSGD [10] can dynamically adjust the local learning rate of worker according to the delay of the worker. Zhang et al. [11]has a similar idea, it adjusts the learning rate by the state of gradient. However, These approaches use the same amount of training data on every worker. They still waste the waiting time and resources of workers while synchronizing due to unreasonable training data distribution. Therefore, self-adaptive task allocation algorithm is urgently needed.

III. Method

In order to accelerate, we proposed a self-adaptive task allocation algorithm (SATA) by balancing the number of tasks among workers automatically. In SATA, we reassign training data as n subsets to n workers based on training time and the amount of subsets from the previous epoch. First, we establish a mathematical model coordinating with information of the previous epoch and the current epoch to allocate tasks. Then we integrate the All-Reduce algorithm to build our adaptive distributed training process.

A. task allocation model

In this subsection, we construct a function describing the amount of tasks which is only related to how long each worker takes to process and how much training data each worker holds at the last epoch. The detailed induction procedures are described below.

1) preliminaries: In order to describe the model better, we make the following premises and notations based on the real situations.

First, due to synchronization operations before aggregating local gradients, it is approximately though that all workers start and end at the same time in the process of each gradient aggregation with All-Reduce. Therefore, the gradient aggregation time of all workers t_c^i is equal. We set:

$$t_c^1 = t_c^2 = \cdots = t_c^n \tag{1}$$

Second, in synchronous SGD algorithms, total training procedure includes three steps: computation, synchronization and update. All workers execute the same preprocessing steps as well as they are blocked at barrier finally. Therefore, it can be approximately thought that total training time of all workers T_i will be equal. We set:

$$T_1 = T_2 = \cdots = T_n \tag{2}$$

Third, we set the ratio of local mini-batch size to global batch size in the previous epoch w_i^{k-1} and in the current epoch w_i^k. To avoid modifying learning rate along with the global batch size, we assume that global batch size keeps as a constant. Therefore, the sum of the ratio w and the changed ratio u_i will be set as:

$$w_1^k + w_2^k + \cdots + w_n^k = 1 \tag{3}$$

$$u_1 + u_2 + \cdots + u_n = 0 \tag{4}$$

Finally, to ensure the convergence of DNN model, the data subset D_i on worker i should be the uniform distribution over the assigned data samples. We distribute training data by sequential assignment on a sample-by-sample basis.

Besides, there are some parameters of the task allocation model to illustrate: gradient computing time t_s^i, synchronization waiting time t_w^i and gradient computing speed v_i where $v_i = D_i/t_s^i$, $T_i = t_s^i + t_w^i + t_c^i$ and the gap between synchronization waiting time is $\Delta t_w^{ij} = |t_w^i - t_w^j|$.

2) mathematical model: First, our objective functions are described as following:

$$\min_D \sum_{i=1}^{n} \sum_{j \neq i}^{n} \Delta t_w^{ij} \tag{5}$$

$$\min_D \quad T_1, T_2, \cdots, T_n \tag{6}$$

(5) and (6) are to minimize the total synchronization waiting time Δt_w^{ij} and total training time T_i. According to the former preliminaries (1) - (4), Δt_w^{ij} can be optimized to $\Delta t_w^{ij} = t_w^i - t_w^j = t_s^j - t_s^i = \frac{D_j}{v_j} - \frac{D_i}{v_i} = 0$. Finally we get the following formula:

$$\frac{Dw_j}{v_j} - \frac{Dw_i}{v_i} = 0 \tag{7}$$

In appendix, we computed the updated w_i^{k+1} in each epoch:

$$w_i^{(k+1)} = u_i + w_i^{(k)} = \frac{w_i^{(k)}/t_s^i}{\sum_{i=1}^{n} w_i^{(k)}/t_s^i} \tag{8}$$

It is found that the ratio w_i^{k+1} of the next epoch only depends on the ratio w_i^k of the current epoch and total computing time t_s^i. To ensure the feasibility of the model, each worker need to broadcast the above two parameters. In Fig.1 it shows the process of task allocation model in epochs. After getting the ratio w_i^{k+1} by the model, we design a complete algorithm called SATA, which shows the whole process of distributed training with the mathematical model.

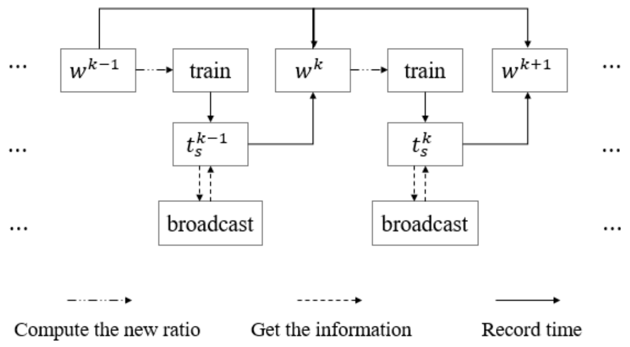


Fig. 1. The execution process of self-adaptive task allocation model, the ratio w_i^{k+1} depends on w_i^k and t_s

B. *self-adaptive task allocation algorithm*

In this subsection, we proposed a self-adaptive task allocation algorithm (SATA) to accelerate decentralized training. Before every epoch, SATA will call the model (8) to obtain the ratio w_i^k. Then SATA reassigns data subsets for all workers. Besides, local mini-batch size is revised by the product of global batch size N and the ratio w_i^k. As can be seen in Fig. 2, there are two cycles in the procedure of SATA. The outer cycle means reassigning data subsets for all workers based on the their own w_i^k at every epoch. The inner cycle means mini-batch data gradient aggregations in every epoch.

To accelerate, SATA makes use of self-adaptive task allocation to speed up. Fast workers take a long time to compute and stragglers take a short time, so the waiting time will be reduced naturally. Superior to other task allocation methods, SATA acquires the state of workers precisely based on reasonable derivation instead of empirical speculations. SATA can guarantee model parameters changing along with the gradient descent direction and converging to the stable loss. There is no changing in synchronous SGD back propagation. Therefore, the convergence point is equal to the original synchronous SGD. Besides, many papers([4,5,7,9,11]) also evaluate and derive the convergence of model when the amount of tasks is changed.

We describe the procedure of our SATA in Algorithm 1. There are three main parts in the SATA algorithm, determining the ratio, redistributing the dataset and training. Under relatively steady environment, w will be steady after several epochs. Therefore, the time of redistribution will be eliminated further in such an environment. By SATA algorithm, we can accelerate the whole procedure of distributed training.

Algorithm 1 Self-Adaptive Task Allocation (SATA) algorithm

```
Require:
  Randomly initialize the data subset ratio for each worker $i$
  at epoch 0: $w_i^0$
  Initialize gradient computing time: $t_s^i \leftarrow 0$
  for the $k$th epoch $\in 1, 2, \cdots, N$ do
      Broadcast $t_s^i$ and $w_i^{k-1}$
      Update $w^k$ by Adaptive task allocation model
      if $w^k \neq w^{k-1}$ then
          Redistribute dataset to workers with $w^k$
      end if
      while data are not completely consumed do
          mini-batch train model
          Record and update $(t_s^i)$
          All-Reduce on model
      end while
```

IV. Experiments

In this section, we developed a series of experiments on self-adaptive task allocation algorithm to observe the DNN model training acceleration. First, we set up several experiments to show that SATA can speed up and be heterogeneity-tolerant on synchronization. We set a real heterogeneous cluster configured with different types of GPUs. Then We compare the performance between a homogeneous cluster and the heterogeneous one. Finally, we compare the results of SATA with other algorithms contributing in straggler problems. The results of all of these experiments show that SATA is suitable for complex heterogeneous environments.

A. *acceleration by SATA algorithm in heterogeneous environments*

We do experiments on multiple machines with multiple GPUs to evaluate results of SATA algorithm. We train ResNet, VGG and ConvNet models on three nodes as well as one node with multiple GPUs with different initial values of w. We record the ratio w, gradient computing time t_s and total training time T in each epoch. In Fig. 3 and 4, the total training time is reduced along with the increasing of epoch (subgraph c,f in Fig. 3 and 4). The gap between gradient computing time of two workers becomes smaller too (subgraph a,d in Fig. 3 and 4). After several epochs, the ratio w becomes steady (subgraph b,e in Fig. 3 and 4).

Further we compared results using the same ratio with the self-adaptive ratio among three groups of GPU clusters. The results in Fig. 5 show that with our SATA algorithm the training time will be reduced in heterogeneous environments.

B. *compared to other algorithms for straggler problems*

In the final experiment, we focus on the straggler problem comparing with All-Reduce and AD-PSGD algorithm. We set a straggler with 2X, 5X and 10X slowdown and set the speedup ratio of PS as 1 as done in Prague [7]. As shown in Fig. 6, our SATA algorithm converges more quickly compared

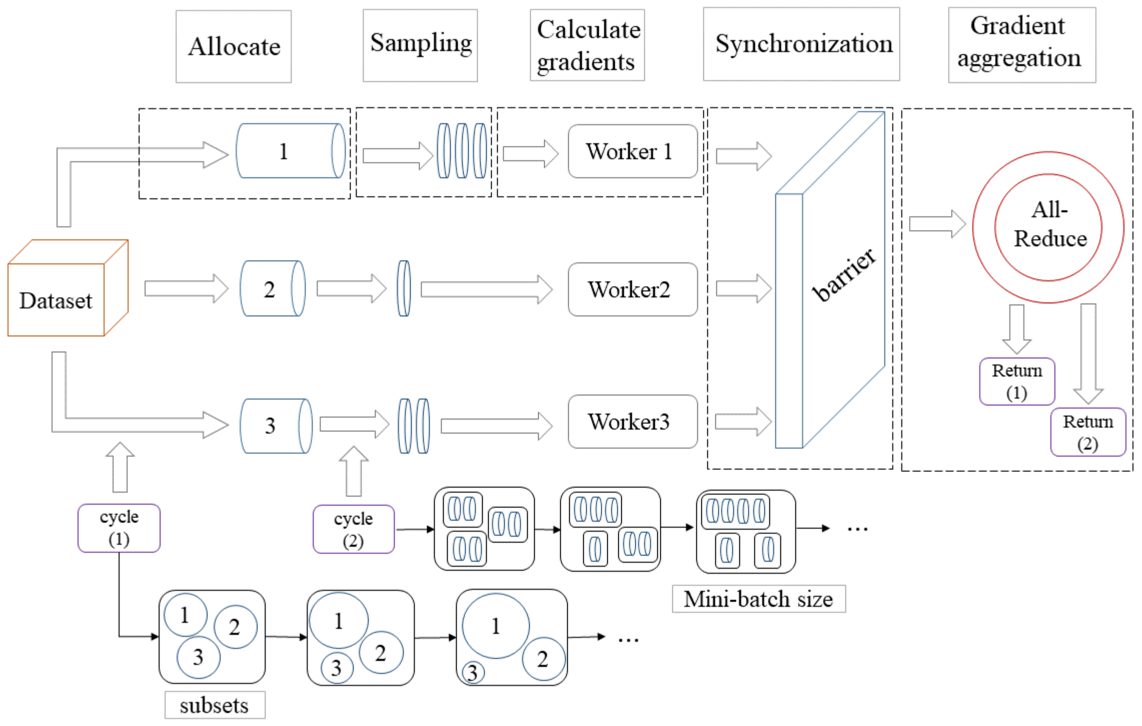


Fig. 2. The procedure of self-adaptive task allocation algorithm with three GPUs. Two cycles represent how the subsets and mini-batch sizes change as epoch increases. After one epoch, results return to cycle (1). After one gradient aggregation, results return to cycle (2)

to All-Reduce and AD-PSGD. In Fig. 7, SATA can reach about 3.3X that of All-Reduce given 2X/5X slowdown, 3.8X that of AD-PSGD under 2X slowdown and 1.9X under 5X slowdown.

V. Conclusion

To deal with stragglers in heterogeneous environments, we proposed a self-adaptive task allocation algorithm (SATA) in this paper. We firstly build a strict mathematical model for self-adaptive task allocation to determine the precise amount of mini-batch size iteratively updating for each worker. And then we give a detail process of our SATA algorithm for distributed training. By this algorithm, workers can reallocate local dataset adaptively according to current gradient computing time. We also set up a heterogeneous environment and designed a series of experiments to evaluate the performance of SATA for straggler problems. The experimental results show that SATA can accelerate distributed training about 3.3X that of All-Reduce and about 1.9X-3.8X that of AD-PSGD algorithm. It means that SATA has sound performance for straggler problems in heterogeneous environments.

In future, we will focus on distributed self-adaptive task allocation algorithms which will eliminate broadcast or centralized communication.

References

[1] T. Ben-Nun and T. Hoefler. “Demystifying Parallel andDistributed Deep Learning: An In-Depth Concurrency Analysis”. In: ACM Computing Surveys 52.4 (2018)

[2] Mu Li, Zhou Li , Alex Smola, Parameter server for distributed machine learning, In NIPS, 2013

[3] Priya Goyal, Piotr Dollár, Ross B. Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaim-ing He. Accurate, large minibatch SGD: training imagenet in 1 hour.CoRR, abs/1706.02677, 2013

[4] E. Y ang, D. K. Kang, and C. H. Y oun. “BOA: batch orchestration algorithm for straggler mitigation of distributed DL training in hetero-geneous GPU cluster”. In: The Journal of Supercomputing (2019).

[5] Aaron Harlap et al. “Addressing the straggler problem for iterative con-vergent parallel ML”. In: Proceedings of the Seventh ACM Symposium on Cloud Computing, Santa Clara, CA, USA, October 5-7, 2016.

[6] X. Lian et al. “Asynchronous Decentralized Parallel Stochastic Gradient Descent”. In Proceedings of the 35th International Conference on Machine Learning, PMLR 80:3043-3052, 2018(2017).

[7] Qinyi Luo, Jiaao He, Youwei Zhuo, and Xuehai Qian. 2020. Prague: High-Performance Heterogeneity-Aware Asynchronous Decentralized Training. Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems. Association for Computing Machinery, New York, NY, USA, 401–416.

[8] Yeo S , Bae M , Jeong M , et al. Crossover-SGD: A gossip-based communication in distributed deep learning for alleviating large mini-batch problem and enhancing scalability. 2020.

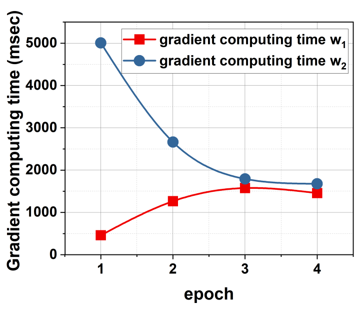
[9] Chen, Jianmin et al. “Revisiting Distributed Synchronous SGD.” ArXiv abs/1702.05800 (2016): n. pag.

[10] J. Jiang et al. “Heterogeneity-aware Distributed Parameter Servers”. In: Acm International Conference. 2017,pp. 463–478.

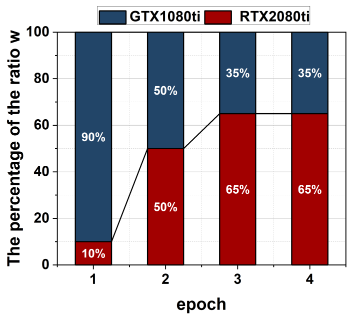
[11] Wei Zhang, Suyog Gupta, Xiangru Lian, and Ji Liu. 2016. Staleness-aware async-SGD for distributed deep learning. In Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI'16). AAAI Press, 2350–2356

[12] C. Szegedy et al. “Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning”. In: AAAI (2016)

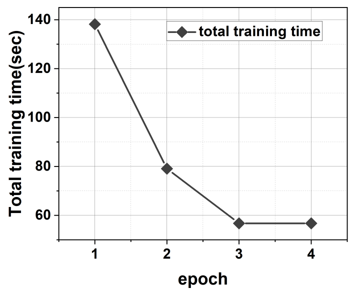
[13] K. Simonyan and A. Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition”.In: Computer Science (2014).



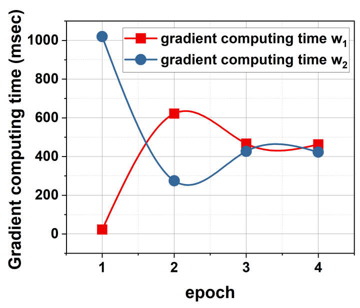
(a) ResNet50-t_s



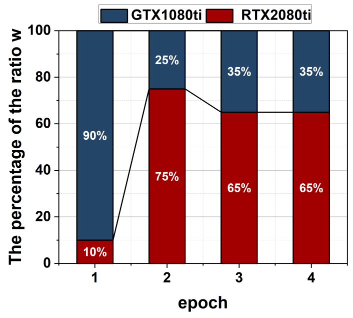
(b) ResNet50-W



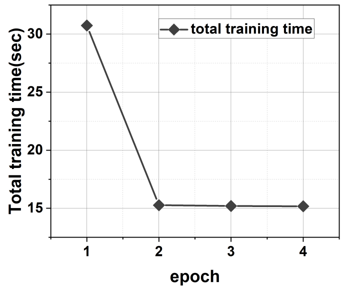
(c) ResNet50-T



(d) VGG16-t_s

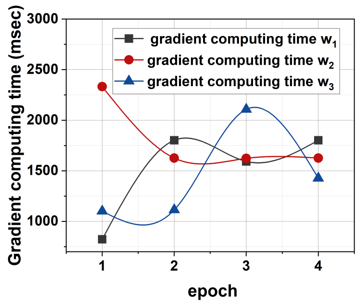


(e) VGG16-W

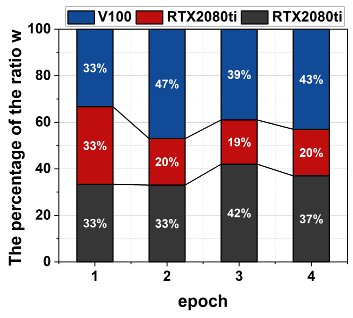


(f) VGG16-T

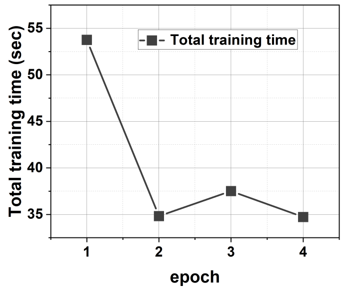
Fig. 3. the gradient computing time before one gradient aggregation, the ratio and training time from one machines with RTX2080ti and GTX1080ti.



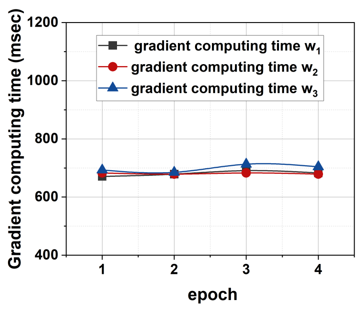
(a) ResNet50-t_s



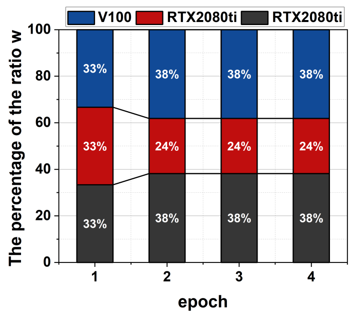
(b) ResNet50-W



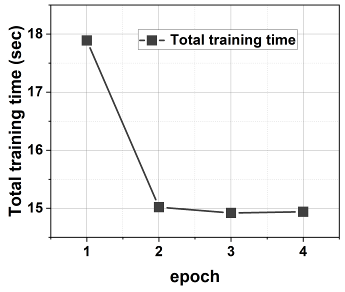
(c) ResNet50-T



(d) VGG19-t_s



(e) VGG19-W



(f) VGG19-T

Fig. 4. the gradient computing time before one gradient aggregation, the ratio and training time from three machines with 2*RTX2080ti and V100 respectively. (a) and (d) represent gradient computing time among workers. (b) and (e) represent the ratio w_i^k. (c) and (f) represent the total training time

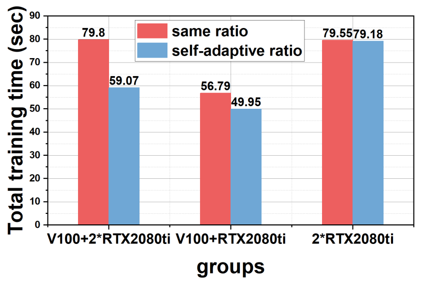
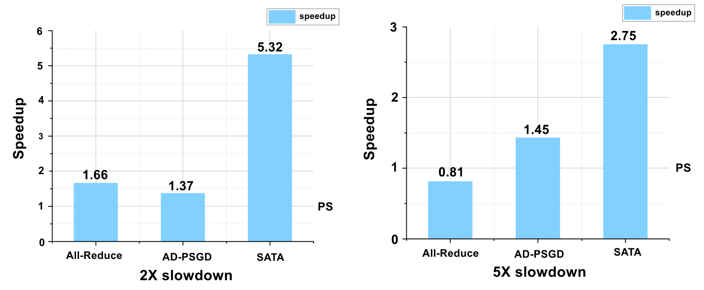
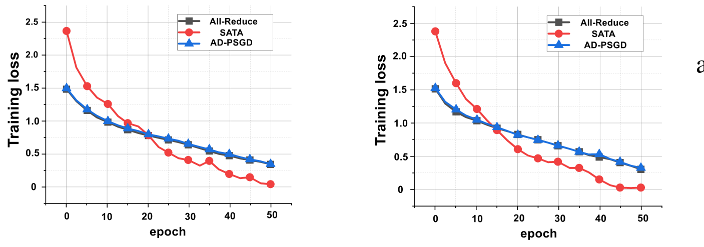


Fig. 5. total training time in one epoch compared with different types of GPUs



(a) speedup in 2X slowdown (b) speedup in 5X slowdown

Fig. 7. the training speedup of models



(a) convergence curve in 2X slowdown (b) convergence curve in 10X slowdown

Fig. 6. the training convergence curve of models for ResNet50.

APPENDIX

Assuming that the kth epoch task allocation of workers is $w_i^{(k)}$, the $(k+1)$th epoch is $w_i^{(k+1)}$, and the changed ratio is u_i as (9)

$$w_i^{(k+1)} = w_i^{(k)} + u_i, i \in [1..n] \tag{9}$$

Due to the waiting time of n workers expected to be 0, so the relationship between workers can be expressed as (10)

$$\frac{Dw_i^{(k+1)}}{v_i} - \frac{Dw_j^{(k+1)}}{v_j} = 0, i \neq j \tag{10}$$

From the view of the linear equation system, (10) is equivalent to (11).

$$\frac{Dw_i^{(k+1)}}{v_i} - \frac{Dw_j^{(k+1)}}{v_j} = 0, \forall i, j = (i+1) mod\ n \tag{11}$$

Simply to get:

$$\frac{w_i^{(k)} + u_i}{v_i} - \frac{w_j^{(k)} + u_j}{v_j} = 0, \forall i, j = (i+1) mod\ n \tag{12}$$

Extract the coefficient matrix to get

$$A' = \begin{bmatrix} \frac{1}{v_1} & \frac{-1}{v_2} & 0 & \cdots & \cdots & 0 & 0 \\ 0 & \frac{1}{v_2} & \frac{-1}{v_3} & \cdots & \cdots & 0 & 0 \\ 0 & 0 & \frac{1}{v_3} & \frac{-1}{v_4} & \cdots & \cdots & 0 \\ & & & \vdots & & & \\ 0 & 0 & \cdots & \cdots & 0 & \frac{1}{v_{n-1}} & \frac{-1}{v_n} \end{bmatrix} \tag{13}$$

The global batchsize remains unchanged, so we have (14) and then (15).

$$w_1 + w_2 + \cdots\cdots + w_n = 1 \tag{14}$$

$$u_1 + u_2 + \cdots\cdots + u_n = 0 \tag{15}$$

Combining (13) with (15) we have (16)

$$A = \begin{bmatrix} & & A' & & & \\ 1 & 1 & \cdots & \cdots & 1 & 1 \end{bmatrix} \tag{16}$$

The constant term is

$$b = \begin{bmatrix} \frac{w_2^{(k)}}{v_2} - \frac{w_1^{(k)}}{v_1} \\ \frac{w_3^{(k)}}{v_3} - \frac{w_2^{(k)}}{v_2} \\ \vdots \\ \frac{w_n^{(k)}}{v_n} - \frac{w_{n-1}^{(k)}}{v_{n-1}} \\ 0 \end{bmatrix} \tag{17}$$

Therefore we only need to solve (18).

$$\begin{gathered} \mathrm{A} \bullet \mathrm{u} = \mathrm{b} \\ u = [u_1, u_2, \cdots\cdots, u_n]^T \end{gathered} \tag{18}$$

Finally we get the solution in terms of u as (19).

$$u = \frac{v_i}{\sum_{j=1}^{n} v_j} - w_i^{(k)} \tag{19}$$