PHD-PSGD: Probabilistic Heterogeneity-aware Decentralized Training

Gyeongchan Yun

Abstract—Recent distributed training approach on deep learning (DL) adopts AllReduce for data parallel training. As the many decentralized algorithms can outperform the centralized ones, it becomes interested in the field of research. In this paper, within the trend of decentralized algorithm, we propose *PHD-PSGD* , a Probabilistic Heterogeneity-aware Decentralized training approach on heterogeneous GPU cluster. PHD-PSGD generates the groups of workers and dynamic communication graph with the heterogeneity-aware probability. Also, PHD-PSGD first proposes a hybrid decentralized algorithm that synchronous AllReduce within the group and asynchronous weight averaging among group. In our experiments, we evaluate ResNet-50 on ImageNet dataset. PHD-PSGD achieves 49% speedup of throughput and the target accuracy of 73.9% is reached 10% faster than that of AllReduce. However, PHD-PSGD cannot achieve the final accuracy of AllReduce. We discuss the future system design to solve the convergence problem.

Index Terms—Cloud computing, heterogeneous GPU cluster, distributed deep learning, data parallel training, decentralized algorithm

1 Introduction

Recently, the application of deep learning technology has been successful and popular in many areas. In the line with this trend, the computational complexity of the DNN model and the volume of datasets have increased for the high accuracy. Therefore, it is necessary to train DNN on multi GPUs to address a problem that requires a long time as training needs intensive computations and large datasets.

There have been a variety of parallel techniques to accelerate training on multiple GPUs: *Data parallel training* has a replica of the entire model and divides dataset across each GPU [1], [2]. Model parallel training holds a partition of the model among multiple GPUs [3], [4], [5], hybrid parallel training combines these two techniques [4], [6], [7]. Furthermore, due to the low GPU utilization of model parallel training, techniques to apply pipelining has been proposed [8], [9].

The widely used approach of distributed training is data parallel training since it is supported by most ML frameworks such as PyTorch [10] and Tensorflow [11]. In Figure 1, there are communication architectures in data parallel training. A parameter server (PS) [2] has been commonly used in distributed DL training. However, as shown in Figure 1 (a), this centralized architecture has a communication bottleneck with the high communication cost on the central nodes. AllReduce [12] in Figure 1 (b) takes a peer-to-peer communication so that improve network latency.

Moreover, the decentralized training in Figure 1 (c) has recently become popular in the field of research. After D-PSGD [13] shows theoretically proves decentralized algorithms can outperform centralized ones. Unlike PS and AllReduce, which use the specific communication graph, the decentralized training scheme can use an arbitrary connected communication graph to specify point-to-point communication between workers with stochastic weight averaging.

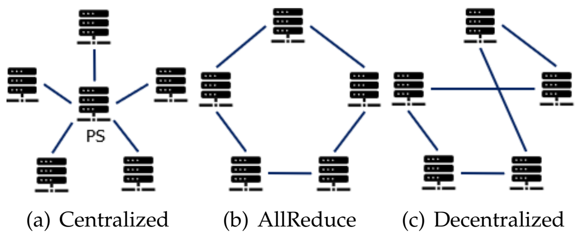


Fig. 1. Data parallel training architecture

With the popular of DL techniques and rapid development of GPU architecture and its expensive cost, heterogeneous GPU clusters are commonplace [14]. Various studies have been devoted to developing the centralized algorithms in heterogeneous environment [14], [15], [16], little attention has been paid to the decentralized algorithms to mitigate heterogeneous problem. Hop [17] points out this problem and succeed in applying technologies that have been applied to centralized settings such as backup worker, staleness bound, and skipping iteration to decentralized settings. In addition, there has been many studies on the decentralized algorithm in heterogeneous cluster [18], [19].

In this paper, within the trend of decentralized algorithm in heterogeneous environment, we propose *PHD-PSGD*, a Probabilistic Heterogeneity-aware Decentralized SGD training approach on heterogeneous GPU cluster. PHD-PSGD generates the groups of workers and dynamic communication graph with the weighted probability by heterogeneity-aware manner. Moreover, we first propose a hybrid decentralized parameter synchronization algorithm that synchronous AllReduce within the group and asynchronous weight averaging among group. We evaluate ResNet-50 [20]

• *G. Yun is with the School of Computer Science and Engineering, UNIST, Ulsan, Republic of Korea.*
E-mail: rugyoon@unist.ac.kr

which is one of the famous CNN models on ImageNet dataset for image classification. PHD-PSGD achieves 49% speedup of throughput and the target accuracy of 73.9% is reached 10% faster than that of AllReduce.

Our contributions are as follows:

- We first propose a hybrid decentralized parameter synchronization algorithm for heterogeneous GPU cluster.
- We first propose a dynamic communication graph connected by a heterogeneity-aware probability.
- We implement *PHD-PSGD* on PyTorch, one of the popular ML frameworks, and evaluate it with PyTorch AllReduce with NCCL which is a state-of-the-art data parallel training method.
- We evaluate ResNet-50 on ImageNet dataset. Compared with other datasets for image classification: MNIST [21] and CIFAR-10 [22], ImageNet dataset is time-consuming and difficult to achieve high accuracy.
- We provide a detailed analysis of performance improvement of *PHD-PSGD*
- In Section 6, we propose a future sophisticated *PHD-PSGD* design to improve its convergence efficiency and performance.

2 Background and Motivation

2.1 Data Parallel Training Architecture

As shown in Figure 1 (a), data parallel training learn the same replicated model with multiple GPU workers and mainly use two architectures to share trained parameters between workers - Parameter Server and Allreduce.

Parameter server is a centralized architecture consisting of the server group and several worker group. The server group maintains global shared parameters and communicates with worker group to update or broadcast the shared parameters. However, this centralized architecture has a communication bottleneck with the high communication cost on the central nodes. Also, DL models with dense parameters are not suitable for the parameter server [23]

AllReduce is the method presented in [12]. It has received attention because of its good performance by utilizing network efficiently. AllReduce consists of peer-to-peer communication and each worker delivers a portion of the gradients to nearby workers. By doing so, AllReduce reduces the amount of data that needs to be transferred and mitigates the centralized network bottlenecks [24]. Nevertheless, AllReduce still suffers from communication overhead when synchronizing gradients, because AllReduce communication is only possible when all workers complete their application simultaneously [9].

In *decentralized* training algorithms [13], [25], there is no central node. Similar with AllReduce, every worker maintains its own version of parameters. Workers use an arbitrary connected communication graph to specify point-to-point communication. In each iteration, a worker computes gradients, sends its parameters to its out-going neighbors, and updates its parameters by averaging them with its incoming neighbors. It has been recently theoretically shown for the first time that decentralized algorithms can outperform centralized ones.

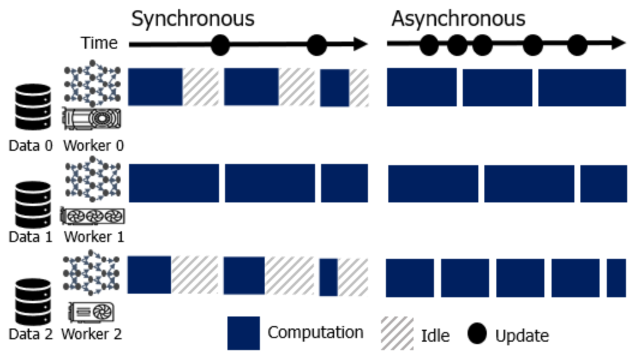


Fig. 2. Synchronization techniques in data parallel training

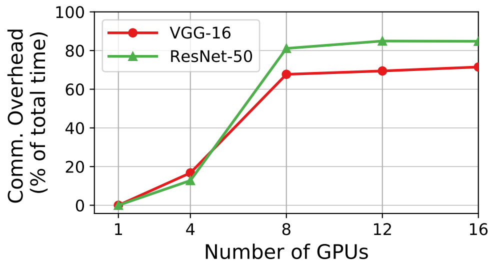


Fig. 3. Inter-node communication overhead on 16 GPU cluster

2.2 Synchronization in Data Parallel Training

In distributed data parallel training, parameter synchronization among multiple workers is necessary. *Bulk Synchronous Parallel (BSP)* is common technique to synchronize parameters after all other workers finish the current mini-batch. As shown in Figure 2, updates(gradients) of current mini-batch from all workers must be reflected to the parameters before starting next mini-batch. It shows high communication overhead and long system idle state on heterogeneous cluster. To reduce synchronization cost in BSP, *Asynchronous Parallel (ASP)* has been proposed that each worker does not wait for other workers. In Figure 2, if there is a worker finished current mini-batch, it send its updates to the parameter server and proceed to next mini-batch without waiting. It shows the improvement of training performance, but it is known that it does not ensure convergence. For *Stale Synchronous Parallel (SSP)*, it is allowed to use a staled version of the parameters for training. Each worker keeps their parameters and do not synchronize until staleness of the parameter exceeds predefined threshold. Each parameter may not reflect recent updates, but its convergence is proved. Also, it shows good performance with heterogeneous cluster.

2.3 Inter-node Communication Overhead

Data parallel training should perform parameter synchronization among multiple workers for each step. In multi-GPU server clusters, the workers on the same node come up with intra-node communication and do inter-node communication with the worker on the other node. Figure 3 shows the fraction of time spent by communication stalls in data parallel training on GPU clusters which consists of 4 GPUs in each node. Data parallel training with 4 GPUs only takes place intra-node communication. However, as

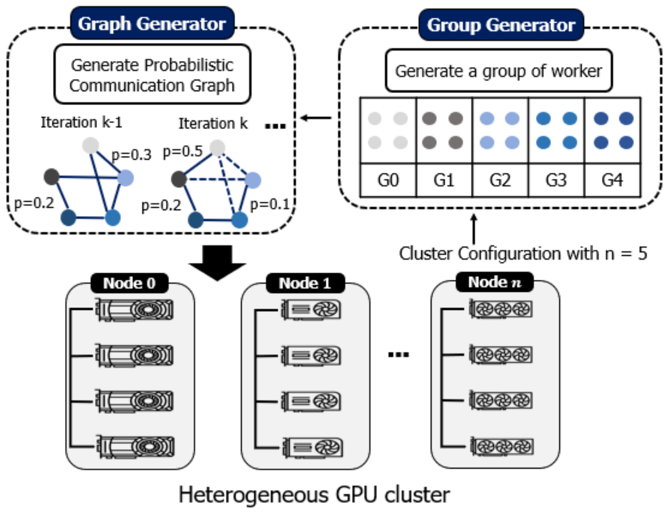

Graph Generator
Generate Probabilistic Communication Graph
Iteration k-1
Iteration k
p=0.3
p=0.5
p=0.2
p=0.2
p=0.1
Group Generator
Generate a group of worker
G0
G1
G2
G3
G4
Cluster Configuration with n = 5
Node 0
Node 1
Node n
Heterogeneous GPU cluster

Fig. 4. System architecture of PHD-PSGD

shown in Figure 3, the communication overhead drastically increases at data parallel training using 8 GPUs where it also happens inter-node communication. We can observe that the performance of data parallel training significantly suffers from inter-node communication. Furthermore, it is inavoidable as the scale of data parallel training increases.

3 System Design

In this section, we explain the system design of PHD-PSGD. PHD-PSGD focuses on training the DNN model by the decentralized approach on heterogeneous GPU cluster. As we mentioned in background and motivation, we try to tackle the performance degradation problem also shown decentralized approach in heterogeneous environments.

Figure 4 shows the overall system architecutre of PHD-PSGD. There are two main components: *group generator* and *graph generator*. PHD-PSGD generates the groups of workers in group generator and dynamic communication graph with the weighted probability by heterogeneity-aware manner in graph generator. In addition, PHD-PSGD adapts a hybrid parameter synchronization strategy that synchronous AllReduce within the group and asynchronous weight averaging among group.

3.1 Profiler

Profiler gathers two information of the input DNN model. First, Profiler measures the size of weight(weight, bias), which is called model parameters, for all layers in the DNN model in bytes. Second, Profiler measures the average computation time of the DNN model on each GPU by running 1000 iterations.

With the information of the input DNN model, Profiler estimates the communication time by dividing the size of model parameters by the theoretical intra- and inter-network bandwidth taken as the input of Profiler. Finally, Profiler records three configurations 1) computation capability of heterogeneous GPU 2) intra-node communication 3) inter-node communication for the heterogeneous GPU cluster configuration. The cluster configuration is taken as input to the group generator and the graph generator.

3.2 Group Generator

Group generator takes two inputs: 1) Cluster configuration 2) The number of GPUs per group N. Then, N_G is calculated by dividing the total number of machines in the cluster by N. Given N_G, the group generator makes groups to minimize the objective function among all candidate of groups. The equation for the objective function is as follows:

$$\sum Objective = \sum Comp(G_i) + \sum Comm(G_i)$$

$$where\ Comm = \frac{ParamSize(DNN\ model)}{NetworkBW}$$

For example, in Figure 4, given N_G of 5 and the cluster configuration, group generator generates 5 groups consisting of 4 worker with respect to the objective function.

3.3 Graph Generator

After groups being generated, the graph generator takes the group information. The graph generator calculates the weighted probability which is assigned by higher probability when communication between the groups with similar execution time. For instance, in Figure 4, the dynamic communication graph is configured with the probability of 0.5 between G0 and G1. It indicates that two groups G0 and G1 has the similar execution time.

3.4 Synchronization

To reduce synchronization cost in heterogeneous environment, PHD-PSGD adapts the hybrid parameter synchronization technique to leverage the decentralized settings configured by the group and graph generator.

Within the group, all workers execute AllReduce *synchronously*. In Figure 4, workers in G0, G1, ..,G4 respectively synchronize parameters of the DNN model by AllReduce. Among the groups, each group communicates with other groups *asynchronously* by weight averaging adapted in AD-PSGD. When workers within the group Gn is processing the mini-batch i, group Gn+1 which wants to do weight averaging should be blocked until the mini-batch i in the group Gn is finished. This is the reason that the communication graph among groups is configured to connect between groups of similar execution time with high probability in Section 3.3

4 Implementation

We implement PHD-PSGD by modifying the PyTorch distributed training code on ImageNet [26].

Group Generator: Group Generator is implemented by the PyTorch existing APIs: `init_process_group()` and `new_group()`. The function can be used to create new groups, with arbitrary subsets of all processes. The objective function in Section 3.2 is implemented by Python `math()` library. The objective function returns list of group consisting of worker id (from perspective of PHD-PSGD, process id). Then, the list is passed to `new_group()` to generate group.

Graph Generator: Graph Generator is implemented by the Pytorch point-to-point communication APIs: `send(tensor, group)` and `recv(tensor, group)`. The weighted probability in Section 3.3 is implemented by Python `random()` library. After group-pair is determined

	Architecture	CUDA Core	Memory Size(GB)
TITAN RTX	Turing	4608	24
TITAN V	Volta	5120	12
GeForce RTX 2060	Turing	1920	6
Quadro P4000	Pascal	1792	8

TABLE 1
Heterogeneous GPUs

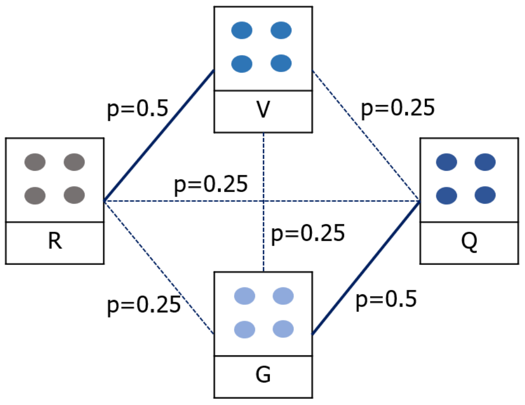


Fig. 5. Methodology of PHD-PSGD on the heterogenous GPU cluster

by the weighted probability, group-pair is passed as group to `send(tensor, group)` and `recv(tensor, group)`. Note that in PHD-PSGD, we only assume data parallel training, `tensor` is all of the model's parameter.

Synchronization: The PHD-PSGD's synchronization strategy is implemented by the Pytorch multi-GPU collective function APIs: `all_reduce_multigpu(tensor, group)` and `barrier()`. Within the group, the list of process id which is dealt with group in the implementation is passed to `all_reduce_multigpu(tensor, group)`. Among groups, weight averaging is processed with parameters of workers from `send()` and `recv()` in the graph generator while blocking by `barrier()`.

5 Evaluation

5.1 Experiment Setup

GPU cluster configuration: We have heterogeneous GPU cluster consisting of 4 nodes connected via 56Gbps Infiniband. Each node has two Intel Xeon Octa-core E5-2620 v4 @ 2.10 GHz processors and 64 GB memory. Each node is equipped with a different type of 4 GPUs and PCIe-3×16 (15.75 GB/s): TITAN RTX [27], TITAN V [28], GeForce RTX 2060 [29], and Quadro P4000 [30]. The total number of GPUs is 16 and the detailed hardware spec is shown in Table 1. Ubuntu 16.04 LTS with Linux kernel version 4.4, CUDA 10.2, cuDNN 7, NCCL v2.4 is used.

Frameworks: We select PyTorch v1.1.0 [10] to implement PHD-PSGD. Also, we adopt baseline as PyTorch AllReduce with DistributedDataParallel [31] on communication backend NCCL.

DNN model and Dataset: We evaluate ResNet-50 [20] which is one of the famous CNN models for image classification. The model is trained with the ImageNet (ILSVRC 2012) [32] dataset that has 1.28M training images and 50K validation images in 1000 categories.

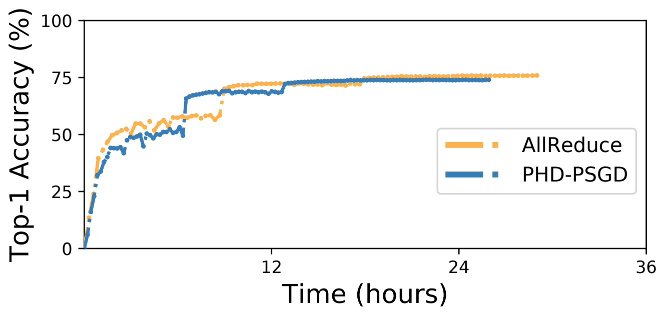


Fig. 6. Top-1 validation accuracy (%) of ResNet-50 using 16 GPUs

5.2 Methodology

Hyper-parameter configuration: With the constraint of memory size of heterogenous GPUs, we set up a batch size per GPU to 32. Thus, a global batch size in AllReduce is 512 since we have total 16 GPUs. The following hyper-parameters are equally configured in [20]. We use SGD with a momentum of 0.9. The learning rate starts from 0.1 and is divided by 10 at 30, 60, 80, 90 epoch. We use a weight decay of 0.0001 and adopt batch normalization.

PHD-PSGD: Given our heterogeneous GPU cluster, PHD-PSGD constructs their group and probabilistic communication graph shown in Figure 5.

Profiler runs to get the profiling data which includes the computation time of ResNet-50 on each GPU, the parameter size of ResNet-50 (98MB) and estimates the communication time by taking input as theoretical intra- and inter-network bandwidth of 15 GB/s, 56Gbps respectively.

Group generator takes two input as the number of GPU per group N = 4 and the GPU cluster configuration. Eventually, *Group generator* makes a decision of generating the group as the form of intra-node workers. It is determined that the computation is executed on the intra-node and asynchronous communication happens with the inter-node groups to aggregate model convergence data produce the minimal total training execution time.

Graph generator gives more weights to group-to-group communications with similar computing power. Since group-to-group communication behaves asynchronously, this communication graph helps reduce each group waiting time for finishing mini-batch processing within other groups. In our heterogeneous GPU cluster, with perspective of computational capability, $V > R > G > Q$ where V, R, G and Q represents TITAN V, TITAN RTX, GeForce RTX 2060 and Quadro P4000 respectively. Among them, V and R has the similar power, G and Q has the similar one. As shown in Figure 5, the weighted probability of communication between groups of V and R, groups of G and Q is 0.5 and other probability is configured to 0.25.

5.3 Convergence

In this section, we analyze the convergence performance of PHD-PSGD. The performance metric is the time to reach a target top-1 validation accuracy (%) which indicates how fast the training approach can make the DNN model achieve the target accuracy. Figure 6 shows the top-1 validation accuracy of ResNet-50 of PHD-PSGD and AllReduce over

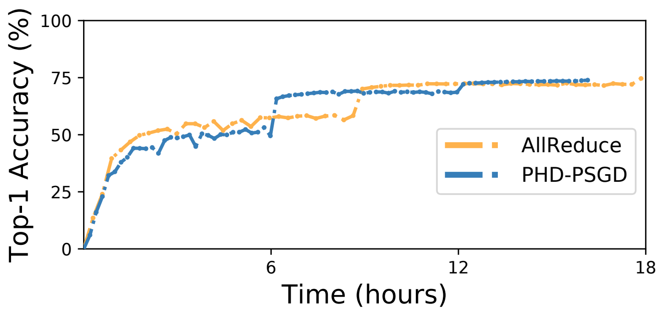


Fig. 7. Time to the target accuracy (%) of ResNet-50 using 16 GPUs

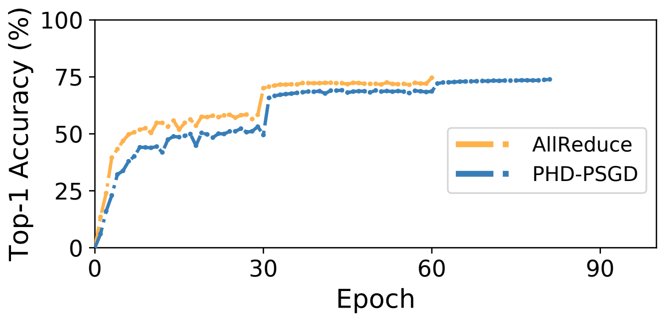


Fig. 8. Epoch to the target accuracy (%) of ResNet-50 using 16 GPUs

time using 16 GPUs. We have an observation that as ResNet-50 arrives at 81 epoch, the top-1 validation accuracy is reached to 73.9% and can not increase anymore. Thus, we stop PHD-PSGD over at 120 epoch, eventually, the result was 2% short of the 75.9% achieved by AllReduce as our baseline. We analyze that asynchronous weight averaging among the groups impact more negatively to the model's convergence than we expect. The convergence problem of PHD-PSGD is discussed in Section 6.

With the accuracy of PHD-PSGD not reaching that of AllReduce, we set the target accuracy to 73.9% and proceed with the evaluation. As shwon in Figure 7, the time to reach the target accuracy of PHD-PSGD is 10% faster than that of AllReduce. Figure 8 shows convergence rate of PHD-PSGD and AllReduce. We represent epoch to reach the target accuracy of ResNet-50 using 16 GPUs which indicates how many epochs are proceeded to achieve the target accuracy. The proceeding epochs by AllReduce is 60 and the epoch of PHD-PSGD is 81. We observe that it results in the decrease of the statistical efficiency that our hybrid parameter synchronization strategy where synchronous AllReduce within the group and asynchronous weight averaging among the group. Nevertheless, the reason to achieve the accuracy of PHD-PSGD 10% faster can be seen as a significant increase in training performance over the decreased statistical efficiency. The analysis of training performance proceeds in Section 5.4.

5.4 Performance

Figure 9 shows the training throughput of PHD-PSGD and AllReduce on ResNet-50 using 16 GPUs. The performance metric is images per seconds (Images/sec). For image classification task, a image is considered as a mini-batch, thus the metric indicates how many mini-batches are processed per seconds. As shown in Figure 9, PHD-PSGD achieves 49.2% speedup of the performance compared to AllReduce.

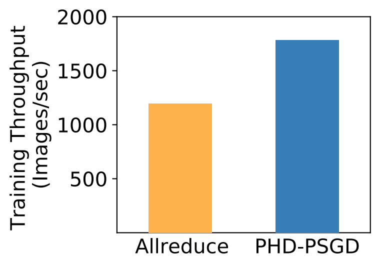


Fig. 9. Training throughput (Images/sec)

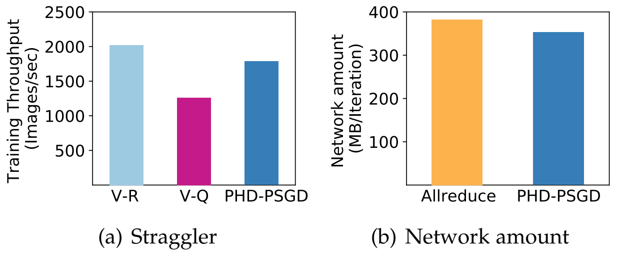
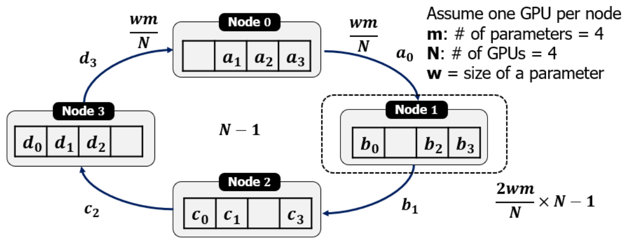


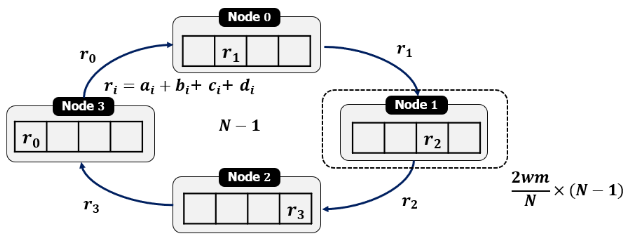
Fig. 10. Throughput analysis

To show how PHD-PSGD mitigates the straggler problem which is the common problem of data parallel training on heterogeneous GPU cluster, we measure the training performance two cases when V, the most powerful computing capability in our GPU cluster, is paired with R which has the similar capability and is paired with a straggler, Q. In Figure 10 (a), V-R denotes the pair of V and R, V-Q denotes that of V and Q. Since the communication graph in PHD-PSGD is probabilistic, these two cases are not always happened, but it can be shown that PHD-PSGD mediates the straggler problem.

Figure 10 (b) shows a network amount per iteration (MB/Iteration) on a server. We only consider the inter-node communication in this section. As shown in Figure 10 (b), the network amount of PHD-PSGD reduces 8% compared to that of AllReduce. We analyze that the dynamic communication graph with the weighted probability in PHD-PSGD can reduce inter-node communication overhead since the configurable communication graph in the decentralized training doesn't need to communicate with all other workers in each iteration. Moreover, since communication behavior of PHD-PSGD is stochastic, we can not analyze the pattern, thus we analyze a communication pattern of AllReduce in Figure 11 to ensure the measurement of the network amount on a server. In Figure 11, each worker sends and receives wm/N bytes of a chunk of model parameters for $2(N-1)$ communication steps, where gradients are reduced for the first $N-1$ steps in (a) and the reduced values r_i are broadcast back to all workers for the next $N-1$ steps in (b). Therefore, the amount of data going into and out of a single worker via network transfer is $2wm/N$ bytes and the total communication steps are $2(N-1)$, thus the total amount is $4wm/N(N-1)$. In our settings, N = 16, wm = 98 MB which



(a)



(b)

Fig. 11. Communication behavior of AllReduce

is the parameter size of ResNet-50, so a theoretical amount is 367.5 MB/Iteration which is close to the estimated amount of 382 MB in Figure 10 (b).

6 Discussion

In this section, we discuss the convergence problem of *PHD-PSGD* mentioned in Section 6.3 and future system design.

Convergence problem: In system design, *PHD-PSGD* focuses on improving the overall system performance. However, since it is well known problem that asynchronous parameter aggregation takes a negative effect on convergence [15], [33], we attempt to mitigate the adverse effect by executing AllReduce synchronously within group and communication asynchronously with other groups.

However, this asynchronous method seems to have the more negative impact than we expected. The critical reason we analyze is that in the process of determining the dynamic communication graph, the graph generator connects groups of similar computing power with high probability to prevent the communication stall, not considering the parameter staleness among groups. In addition, since the research in ML field is not a focus in this paper, but it is necessary to study the configuration of hyper-parameter for our different hybrid parameter synchronization policies. Even though the hyper parameter is set the same as AllReduce for a fair comparison.

As a result, ResNet-50 arrives at the latter epoch and the top-1 validation accuracy is reached to 73.9 % and can not increase anymore. Finally, the result was 2 % short of the 75.9 % achieved by AllReduce as our baseline.

Future system design: It seems that the policy of determining the weighted probability needs to be more sophisticated to consider the convergence as well. One of the future designs we discuss is that a convergence monitor component can be added to manage the mini-batch processing step in all groups. They store it as metadata so that the probability can be determined with the factor of predefined the group's staleness bound such as the SSP technique [16]. One concern about this design is the monitoring overhead. Since the convergence monitor checks the mini-batch processing step among all groups, they need to communicate with all other groups periodically. Therefore, to reduce the overhead, the technique to determine the sampling window also need to be studied. We remain it as future work.

7 Related Work

There have been studies about decentralized communication topology on heterogeneous cluster. D-PSGD [13] assumes that a network is formed as a connected graph. Every node has its local parameter of the model. In each clock, all nodes average its parameter only with its neighbors. It still has idle time caused by waiting for the slowest node to finish its job. AD-PSGD [18] is another work that shows zero idle time. It randomly selects one neighbor and averages parameters. This averaging step is done in asynchronous manner on each node. However, it only considers heterogeneous network topology, not a computational resources such as GPU and CPU. Hop [17] and Prague [19], which are the closet related study, consider decentralized training with heterogeneity-aware manner on heterogeneous GPU cluster. Hop proposes a queue-based synchronization protocol that can leverages backup workers and bounded staleness and apply skipping iterations in the decentralized setting to mitigate the effect of slower workers. Prague designs a distributed training method that has both high performance in homogeneous and heterogeneous environments. They propose Partial AllReduce that enables fast synchronization among a group of workers and static group scheduling in homogeneous environment.

There is AllReduce study to solve the problem of cluster with heterogeneous network bandwidth [34]. They execute AllReduce in two hierarchical network levels. However, this technique does not consider heterogeneous GPUs and not decentralized setting.

8 Conclusion

In this paper, we propose *PHD-PSGD*, a Probabilistic Heterogeneity-aware Decentralized training approach on heterogeneous GPU cluster. PHD-PSGD generates the groups of workers and dynamic communication graph with the weighted probability by heterogeneity-aware manner. Also, PHD-PSGD adapts a hybrid data parallel strategy that synchronous AllReduce within the group and asynchronous weight averaging among group. In our experiments, we evaluate ResNet-50 on ImageNet dataset. PHD-PSGD achieves 49% speedup of throughput and the target accuracy of 73.9% is reached 10% faster than that of AllReduce. However, PHD-PSGD cannot achieve the final accuracy of AllReduce. We discuss the future system design to solve the convergence problem.

References

[1] L. Bottou, "Large-Scale Machine Learning with Stochastic Gradient Descent," in *Proceedings of COMPSTAT*, 2010.

[2] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, "Scaling Distributed Machine Learning with the Parameter Server," in *Proceedings of the Symposium on Operating Systems Design and Implementation (OSDI)*, 2014.

[3] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, and N. Andrew, "Large Scale Distributed Deep Networks," in *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*, 2012.

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

[5] S. Lee, J. K. Kim, X. Zheng, Q. Ho, G. A. Gibson, and E. P. Xing, "On Model Parallelization and Scheduling Strategies for Distributed Machine Learning," in *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*, 2014.

[6] Z. Jia, S. Lin, C. R. Qi, and A. Aiken, "Exploring hidden dimensions in parallelizing convolutional neural networks," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2019.

[7] Z. Jia, M. Zaharia, and A. Aiken, "Beyond data and model parallelism for deep neural networks," in *In Proceedings of the 2nd SysML Conference(SysML)*, 2018.

[8] Y. Huang, Y. Cheng, D. Chen, H. Lee, J. Ngiam, Q. V. Le, and Z. Chen, "GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism," *arXiv preprint arXiv:1811.06965*, 2018.

[9] D. Narayanan, A. Harlap, A. Phanishayee, V. Seshadri, N. R. Devanur, G. R. Ganger, P. B. Gibbons, and M. Zaharia, "Pipedream: Generalized pipeline parallelism for dnn training," in *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, 2019.

[10] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[11] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, "Tensorflow: A system for large-scale machine learning," in *Proceedings of the Symposium on Operating Systems Design and Implementation (OSDI)*, 2016.

[12] M. Barnett, L. Shuler, R. van De Geijn, S. Gupta, D. G. Payne, and J. Watts, "Interprocessor collective communication library (intercom)," in *Proceedings of IEEE Scalable High Performance Computing Conference*, 1994.

[13] X. Lian, C. Zhang, H. Zhang, C.-J. Hsieh, W. Zhang, and J. Liu, "Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent," in *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*, 2017.

[14] J. Jiang, B. Cui, C. Zhang, and L. Yu, "Heterogeneity-aware distributed parameter servers," in *Proceedings of the International Conference on Management of Data (SIGMOD)*, 2017.

[15] J. Chen, X. Pan, R. Monga, S. Bengio, and R. Jozefowicz, "Revisiting distributed synchronous sgd," *arXiv preprint arXiv:1604.00981*, 2016.

[16] Q. Ho, J. Cipar, H. Cui, S. Lee, J. K. Kim, P. B. Gibbons, G. A. Gibson, G. Ganger, and E. P. Xing, "More Effective Distributed ML via a Stale Synchronous Parallel Parameter Server," in *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*, 2013.

[17] Q. Luo, J. Lin, Y. Zhuo, and X. Qian, "Hop: Heterogeneity-aware decentralized training," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 893–907, 2019.

[18] X. Lian, W. Zhang, C. Zhang, and J. Liu, "Asynchronous decentralized parallel stochastic gradient descent," *arXiv preprint arXiv:1710.06952*, 2017.

[19] Q. Luo, J. He, Y. Zhuo, and X. Qian, "Prague: High-performance heterogeneity-aware asynchronous decentralized training," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 401–416, 2020.

[20] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[21] Y. LeCun, "The mnist database of handwritten digits," *http://yann. lecun. com/exdb/mnist/*, 1998.

[22] A. Krizhevsky, G. Hinton, *et al.*, "Learning multiple layers of features from tiny images," 2009.

[23] S. Kim, G.-I. Yu, H. Park, S. Cho, E. Jeong, H. Ha, S. Lee, J. S. Jeong, and B.-G. Chun, "Parallax: Sparsity-aware data parallel training of deep neural networks," in *Proceedings of the European Conference on Computer Systems (EuroSys)*, 2019.

[24] A. Sergeev and M. Del Balso, "Horovod: fast and easy distributed deep learning in tensorflow," *arXiv preprint arXiv:1802.05799*, 2018.

[25] A. Kadav and E. Kruus, "Asap: asynchronous approximate data-parallel computation," *arXiv preprint arXiv:1612.08608*, 2016.

[26] PyTorch, "PyTorch distributed training on ImageNet." https://github.com/pytorch/examples/tree/master/imagenet.

[27] NVIDIA, "TITAN RTX." https://www.nvidia.com/en-us/titan/titan-rtx/.

[28] NVIDIA, "TITAN V." https://www.nvidia.com/en-us/titan/titan-v/.

[29] NVIDIA, "GeForce RTX 2060." https://www.nvidia.com/en-us/geforce/graphics-cards/rtx-2060/.

[30] NVIDIA, "Quadro P4000." https://www.nvidia.com/en-us/design-visualization/quadro-desktop-gpus/.

[31] PyTorch, "PyTorch DistirbutedDataParallel." https://pytorch.org/docs/stable/notes/ddp.html.

[32] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.

[33] S. Gupta, W. Zhang, and F. Wang, "Model accuracy and runtime tradeoff in distributed deep learning: A systematic study," in *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pp. 171–180, IEEE, 2016.

[34] X. Jia, S. Song, W. He, Y. Wang, H. Rong, F. Zhou, L. Xie, Z. Guo, Y. Yang, L. Yu, T. Chen, G. Hu, S. Shaohuai, and C. Xiaowen, "Highly Scalable Deep Learning Training System with Mixed-Precision: Training ImageNet in Four Minutes," *arXiv preprint arXiv:1807.11205*, 2018.