Hierarchical Heterogeneous Cluster Systems for Scalable Distributed Deep Learning

Yibo Wang
Electrical Engineering and Computer Science
University of California, Irvine
Irvine, USA
yibow6@uci.edu

Tongsheng Geng
Electrical Engineering and Computer Science
University of California, Irvine
Irvine, USA
tgeng@uci.edu

Ericson Silva
Electrical Engineering and Computer Science
Pontifical Catholic University of Minas Gerais
Minas Gerais, Brazil
ericson.marquiere@sga.pucminas.br

Jean-Luc Gaudiot
Electrical Engineering and Computer Science
University of California, Irvine)
Irvine, USA
gaudiot@uci.edu

***Abstract*—Distributed deep learning framework tools should aim at high efficiency of training and inference of distributed exascale deep learning algorithms. There are three major challenges in this endeavor: scalability, adaptivity and efficiency. Any future framework will need to be adaptively utilized for a variety of heterogeneous hardware and network environments and will thus be required to be capable of scaling from single compute node up to large clusters. Further, it should be efficiently integrated into popular frameworks such as TensorFlow, PyTorch, *etc.* This paper proposes a dynamically hybrid (hierarchy) distribution structure for distributed deep learning, taking advantage of flexible synchronization on both centralized and decentralized architectures, implementing multi-level fine-grain parallelism on distributed platforms. It is scalable as the number of compute nodes increases, and can also adapt to various compute abilities, memory structures and communication costs.**

***Index Terms*—Distributed, Allreduce, Framework, Deep Learning, Hierarchical, Heterogeneous, Cluster**

I. Introduction

Distributed deep learning frameworks have been developed to accelerate training speed by assigning computing jobs to distributed nodes in clusters [1]. The frameworks have been using a parameter-server model for centralized architectures and an all-reduce model for decentralized architectures. These two distinct models utilize either strict or flexible synchronization mechanisms [2]. It has been shown that these two models and architectures together with synchronization mechanisms have good performance when used in homogeneous cluster systems [3].

Many projects have focused on heterogeneous cluster systems [4], [5]. These works have tended to search for the homogeneous sections within the heterogeneous system to separate the whole system into different sections and use different schemes and synchronization approaches for the local section. This means that, first, these methods all assume that homogeneous sections exist in a heterogeneous system and are sufficiently large compared to the number of sections. Second, the network connections and communication cost in the cluster are ignored or ideally considered to be identical for every two or group of nodes, which significantly simplifies the problem with heterogeneous systems. In many scenarios, certain nodes or group within the clusters need to join or cooperate to complete tasks. Alternatively, disparate local clusters or data centers across large network may need to integrate. In most instances, there is a desire to optimize the utilization of diverse resources to achieve efficient neural network training. Thus, these conditions inevitably lead to a heterogeneous cluster environment that requires thorough research.

We are proposing to consider all these varieties of software, hardware, and network conditions in heterogeneous clusters. First, we introduce a dynamically hybrid and hierarchical logical architecture for cluster nodes, which can transform a classical centralized and decentralized architecture to make use of strict and flexible synchronization based on the target application and architecture. Second, we propose using multi-level fine grain parallelism to optimize the whole system so as to emphasize scalability, adaptivity and performance. We have developed an associated runtime system prototype to optimize the efficiency of deep neural network training and utilization of various hardware configurations in both GPU and CPU heterogeneous environments. We observe a close to optimal solution using our runtime system.

II. Related Work

Asynchronous distributed training has been proposed a decade ago to reduce the overhead of fully synchronizing when updating the gradient and parameter in neural network models during the neural network training process [2], [3].

This work was supported by the National Science Foundation (NSF) under award CCF-176379. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF.

979-8-3503-7128-4/24/$31.00 ©2024 IEEE

The topic is still of great interest because of the increasing size of training data and models requires increased computational power. The demand for computing power drives the evolution of more powerful GPUs and increasing large size of the cluster. Research has been extended to the optimization of computation, storage and network communication within the cluster during the distributed neural network training [6], [7].

Recent research on neural network training in heterogeneous clusters focuses on finding and grouping the homogeneous part in the cluster system, segmented by varying computation power. Jiang *et al.* [4] proposed Grouping-SGD which identify different groups by compute ability, and within each group using different synchronization strategies. The grouping strategy achieved better performance to accelerate the updates of the model parameters. However, a few factors are simplified and neglected. The grouping algorithm is simplified due to the compute nodes within each group are identical, leading to a simplified approach where heterogeneous factors such as computational power, memory capacity, network bandwidth, availability, reliability and etc, within each group are not fully considered. Moreover, scalability issues are not considered in the research which limits the use in large cluster environments. Kim *et al.* [5] explored the concept of grouping GPUs based on their compute power. They proposed a strategy that involves synchronization within each GPU group and coordination between mini-batch sets, contributing to a well-balanced and efficient training workflow. The parameter server method is used instead of the theoretically more efficient ring-allreduce, which may yield more performance improvement. Further, the efficiency of using heterogeneous GPUs is not addressed. The network factor of the two projects is not considered as a parameter into the system. The approach appears to be static, lacking consideration for dynamic changes during neural network training. This encompasses variations in computation power, accessibility of each node, and fluctuations in network conditions. The absence of these dynamic considerations may limit the adaptability and overall performance of the system. All the above methods improve the performance of neural network training by identifying and leveraging homogeneous parts within the cluster. However, they may not sufficiently address highly heterogeneous conditions present in the cluster. As a result, the applicability of these methods is limited in scope, particularly in scenarios where a significant level of heterogeneity exists among cluster components.

III. Proposed Framework

The overall structure of the framework we are proposing utilizes flexible synchronization [8] and a hybrid architecture which capable of adaptive and dynamic adjustments to the logical structure. These modifications are based on real-time assessments of heterogeneous hardware performance and the use of different algorithms. The targeted platform includes heterogeneous cores, heterogeneous nodes with different computation capability, and distributed nodes with diverse range of network environment. The overall hierarchical structure includes: First, a number of nodes form a unit group. The grouping criteria and how many nodes are in one group can be heuristic for the static structure. For example, we could decide for the identical nodes to be in one group and for the jobs to be evenly split among the nodes in the group. [9] However sometimes we could want to have less powerful nodes to work together with powerful nodes on some jobs which requires less computing power in a way to fully utilize the resources. [10]

Consider the nodes communication within each group, either a centralized or a decentralized way can be applied based on the characteristics of the nodes within the group. Each lower level group can also be considered a member of the high level group. The main issue we have discovered in the recent research paper [4], [5] is that the scalability issue is not well considered. Many of the algorithms and strategies are sensitive to the number of nodes in the group. So in order to organize a large amount of resources, a hierarchical structure has to be applied into the system.

The algorithms are linked to specific hardware components in designing both the initial and dynamic structures, aiming to use different algorithms based on the job profiles. Strategies are designed for compute-bound, memory-bound, and communication-bound applications, each with the corresponding design considerations described in the next section.

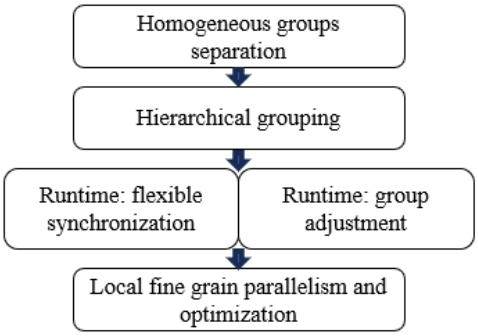

Fig. 1. Proposed Framework Layers

First our framework seeks to group the heterogeneous nodes and form homogeneous groups so as to be able to utilize ring allreduce and synchronization. The second level employs hierarchical grouping, implementing scalable features. It is crucial to limit the number of nodes within a group, preventing the group size from becoming excessively large. On the next level, the framework takes into account the computation and transfer rates to dynamically adjust the grouping.The goal is to prevent the grouping of nodes with significantly disparate transfer rates, avoiding potential inefficiencies. Finally, the framework employs fine-grain parallelism on each compute node to thoroughly explore compute efficiency and its correlation with the local memory structure. Each level of the framework will be discussed in the following subsections.

A. Homogeneous Groups Separation and Creation Involving Heterogeneous Nodes

In contrast to recent hybrid cluster systems discussed in the related work section, which primarily focus on utilizing

the ring allreduce with synchronization and discovering the homogeneous part of a cluster. [**?**], [4], [5] Instead, we are proposing the creation of homogeneous groups that consist of heterogeneous nodes. When we have more heterogeneous nodes and fewer homogeneous nodes in the system, our framework is able to group the heterogeneous nodes into a homogeneous group by considering their computer power and the transfer rate between them. In this way, the allreduce and fully synchronization can still be maintained in the system, and also all the compute nodes in the system can be utilized.

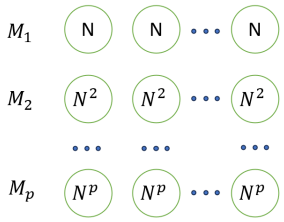

Fig. 2. Homogeneous group separation

As shown in Fig. 2, we use the $bigO$ notation to represent the unit cost (cost per byte) on a certain operation based on the workload and compute power. The node with $O(n)$ in the circle means it will take n units of time to complete a given operation with a given data set of size n. All the nodes in the system can be separated into p number of groups(vertically) from M_1 to M_p. We assigned heterogeneous nodes with the same order of magnitude into the a homogeneous group. Grouping algorithms can be used to separate the nodes first by considering transfer cost among the nodes and compute power. The algorithms are shown below:

Algorithm 1: Homogeneous Group Separation

```
W_i represent worker;
List of Group[ ] distinguished by communication cost;
List of group[ ] distinguished by compute power;
for i = 0 to n do
    for j = i − 1 downto 1 do
        if O(C_{i,i−1}) = O(C_{j,j−1}) then
            W_i.groupId = W_j.groupId;
            Group[W_i.groupId].add(W_i);
        end
    end
end
return output Group[], group[]
```

1) Mathematical analysis of compute and transfer ratio in system decisions: Once we have the grouping and characteristics are done, mathematical analysis can be applied on the static structure to give an early decision on how many nodes in the system should be used, and how they should synchronize with each other when a computation job is given. Using the setup in Fig. 2, we consider the transfer cost to be identical among each of the nodes suppose we have done the separation by transfer cost beforehand, using $O(T_r)$. We also define $O(r_b N)$ where $r_b = op_b/op_a$, which means the ratio of the cost for different operation opa and opb on the same compute node with same data size. Even op_a and op_b are within the same order of magnitude, but they are still different by a ratio of r_b.

Assume we have m jobs $(a_1, a_2, ..., a_m)$, each characterized by the same workload for simplicity. This scenario is analogous to the data parallelism strategy used in distributed neural network training where parallel compute nodes are used to process the partitioned training dataset using a replica of the deep learning model. If $m < M_1$ we have two options: either a serial computation on one of the nodes in M_1, or a parallel one within the group M_1. The cost for the serial option will be mN, and the parallel on M_1 will be nT_r for the transfer data and N for the compute. This means that the total cost is $nT_r + N$ when jobs are evenly assigned onto M_1 group. If the number of jobs is larger than the number of nodes in M_1, *i.e.,* $m > M_1$, we have the option to either send batch (m/M_1) jobs onto M_1, or utilize both M_1 and M_2 groups with M_1 jobs sent to M_1 and $(m - M1)$ jobs are sent to M_2, we can have the following two equations:

$$parallel_{M1} = (nT_r + N)(m/M_1 + 1) \tag{1}$$

$$parallel_{M_1M_2} = (nT_r + N)(m/(NM_1 + M_2)) \tag{2}$$

Apply the same logic on M_3 when all the M_1, M_2 and M_3 are used, so there are $(m - M_1 - nM_2/N)$ jobs on M_3, and total cost on this situation will be:

$$parallel_{M_1M_2M_3} = (nT_r + N)(m/(N^2M_1 + NM_2 + M_3)) \tag{3}$$

by deduction we can conclude that:

$$parallel_{M_1M_2M_3...M_p} = (nT_r + N)(m/(N^{p-1}M_1 \tag{4}$$

$$+N^{p-2}M_2 + N^{p-3}M_3 + ... + M_p)) \tag{5}$$

Overall, we have p equations to evaluate so we can compare and select the least costly option. This will be a static strategy before entering the runtime layer.

B. Hierarchical Logical Structure

It is generally accepted that the ring-allreduce used in decentralized structures is theoretically faster compared with the parameter server approach which is used in centralized architectures, provided all the compute nodes on the ring finish the same workload and communication cost is same between nodes on the ring. [11]

Scalability remains a significant challenge in cluster systems, and even with the implementation of ring allreduce and fully synchronized architectures, concerns regarding scalability persist. [**?**] As more nodes are included into the ring, the probability of incorporating a node with a slow transfer rate into the entire ring structure increased, thereby diminishing the overall performance. There is also a higher likelihood of having a faulty node in the whole structure. So we should improve scalability by using the hierarchical way of grouping to limit the size of the ring. This consequently

drives us to explore the hierarchical solutions if the nodes are heterogeneous and the number of nodes are increasing. Assume we have n number of compute nodes, where n is a large number. The intuitively straight way to form a hierarchy will be to take the square root of n, so the total number of the groups will be $\sqrt{n}$ and the size of each group is also $\sqrt{n}$. Subsequently, within each group, the hierarchical formation can continue with $\sqrt{n}$ subgroups. Fig. 3 shows two examples of hierarchical structures with logical connections to form different structures, which include the ring-allreduce configuration and the parameter server configuration.

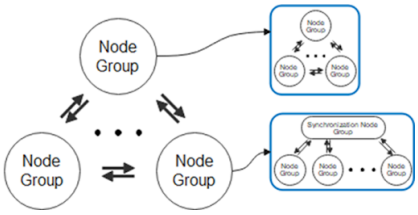

Fig. 3. Example of hierarchical logical structures

When selecting which nodes should be in one subgroup, random formation is one strategy if no particular feature can be discovered among the nodes. The central limit theorem [12] states that, as we take more samples, especially large ones, the graph of the sample means will look more like a normal distribution. This means that we can make sure that the overall compute power will be a normal distribution.

C. Runtime management module with compute and transfer variance(dynamic structure)

The logical organization of hierarchical nodes within our system provides several significant advantages. First, regardless of a node's computational capacity relative to others in the system, it can be effectively used. This means that even fewer powerful nodes make meaningful contributions to the overall system performance. Furthermore, our hierarchical node structure allows for the dynamic inclusion or removal of nodes from specific groups based on network availability and instantaneous data transfer rates. This adaptability at a local level enables fine-grained configuration adjustments in response to varying computational resources and network conditions. Given the inherent volatility of computational power and network conditions, we've developed a runtime management module. This module serves as a proactive supervisor during neural network training, capitalizing on flexible synchronization techniques. By adapting to real-time system characteristics, the module optimizes resource allocation, ensuring that the neural network operates efficiently, even in the face of fluctuating computational resources and network conditions.

Fig. 4 shows node 4 is excluded from the ring when the runtime management module detects an abnormal situation from node 4 either because of slowdown in the computation or the congestion happened in the the network connection from node 4 to runtime management module. Nodes 1, 2 and 3 can form a new logical ring and continue the ring allreduce process when updating the model parameters.

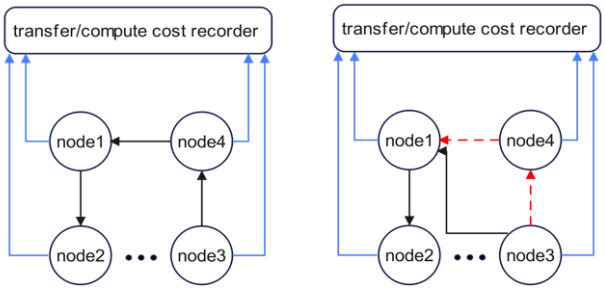

Fig. 4. Dynamic synchronization based on transfer and compute cost

We can use the abnormal detection which identifies issues like loss of dynamic compute power in nodes or network transfer congestion, as an API access. This enables integration of additional intelligence on the design of the algorithm into the whole system. For example, algorithms like running average or limited width sliding window algorithm are used as we develop the runtime system.

Algorithm 2: Runtime Distributed Ring-AllReduce

$node_i$ represents worker i;
$node_i.ring[]$ contains nodes active in ring-AllReduce;
while not converged **do**
  for $i = 0$ **to** n **do**
    if $abnormal\_detection(node_i)$ **then**
      $ring[].remove(node_i)$;
    end if
  end for
  for $i = 0$ **to** n **do**
    $update(node_i.ring[])$;
  end for
end while

Algorithm 3: Runtime Abnormal Detection

$node_i$ represents worker i;
$threthold$ represents limit value to define abnormal;
while not converged **do**
  for $i = 0$ **to** n **do**
    if $abs(node_i.cycle\_time - node_i.running\_average) > threthold$ **then**
      $node_i.abnormal = true$;
    end if
    $update(node_i.running\_average)$
  end for
end while

D. integrate fine grain parallelism into the framework (local optimization)

When it comes to distributing computational tasks across individual nodes, it is crucial to locally optimizing at the level of

each computation node. Initial research findings, as illustrated in Fig. 5, highlight a direct correlation between memory and cache size and the performance of the ring-allreduce algorithm. In the specific context of distributed neural network training, especially when applying model parallelism across the entire system, it becomes imperative to carefully allocate the size of each segmented model part to align with the cache and memory capacities of the local compute node. The preliminary test shown on Fig. 5 shows a AMD Ryzen 5 1600 6-Core CPU with 512K L2 cache, has 5-10% of performance increase when the data size fell within the bounds of the L2 cache. Further, the comparison between the allreduce algorithm and the parameter server model shows as much as twice the performance improvement. These findings underscore the importance of considering both hardware-specific optimizations, such as cache utilization, and algorithmic choices in the pursuit of efficient distributed computing for neural network training.

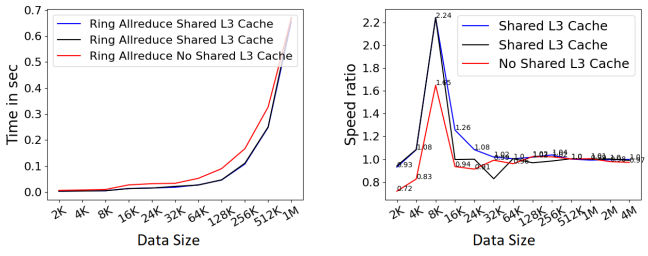

Fig. 5. Cache and Memory Effect on Ring Allreduce and Parameter Server

Zeng *et al.* [13] use the Codelet Model, which is a fine grain dataflow-based execution model, to group 10-15 neurons as a codelet which has achieved 60% higher speed up than a pervasively used coats-grain multithreaded parallel computing model, in the test of Lenet-5, a 7-layer convolutional neural network, on single-node computing system. Geng *et al.* [14] proposed to demonstrate the need for fine-grain synchronization in the presence of rather coarse-grained workload partitioning. They compared the coarse-grain parallelization of a 5-point stencil application implemented with OpenMP to several variants using a fine-grain event-driven execution model, and demonstrated that even with a uniform amount of work per thread, fine-grain synchronization still matters in a regular general-purpose systems.

Thus, by integrating the fine grain model with the distributed system, we expect a cumulative up speedup that comes both from the fine grain model and the distributed system.

IV. Experiment and Discussion

We applied our runtime system in three different environment: heterogeneous CPU and heterogeneous GPU environments. We compared ring allreduce with fully synchronization, no synchronization, fixed percentage synchronization, and flexible synchronization which our runtime module monitor the runtime behavior of each node, and the network condition. We also test the robustness of our runtime system by inserting the delays and variances into the network connections, and we observed the abnormal node such as loss of compute power or loss of connection, will be dropped of the synchronization process if it is detected, to make sure the overall performance of the whole system. All the tests are implemented and integrated with Pytorch framework.

A. Heterogeneous CPU environment

The CPU cluster we formed for evaluation comprises 13 nodes. Nodes with fast connections are placed into one group. Only one node in the group serves as communication node with the outside group. The nodes in group 1 and group 2 are within the same magnitude as we use big O notation to describe the compute power. Group 3 and 4 is one more magnitude slower comparing with group 1 and 2.

TABLE I
Hardware Configurations of Heterogeneous CPUs

Group	*Nodes*	*CPU*	*Network Adapter*
group1	node 0-3	Intel i5	1Gbps
group2	node 4-7	AMD Ryzen 5 1600	1Gbps
group3	node 8-10	ArmV8	1Gbps
group4	node 11-13	ArmV7	1Gbps

We select the LeNet networks and MNIST dataset in this experiment. We record the training time till the test accuracy reaches 95%. In the case of connection all group 1 to 4, the nodes of group 3 and 4 are dropped off during the runtime because the compute power is magnitudes lower and also with a slow connection to group 1 and 2.

Comparing the Fully Synchronization, Percentage Synchronization, and Flexible Synchronization. Percentage Synchronization has close to optimal performance, and flexible synchronization has the advantages on the runtime when network variance is introduced into the computation environment. The plots presented in Figure 6 illustrates a significant decrease in the performance of full synchronization when confronted with network or computation variances. While the percentage synchronization also experiences a slowdown, it maintains a relatively high performance by constraining the synchronization rate with slower connections or computation nodes. The runtime flexible synchronization demonstrates an improvement in performance compared to full synchronization, although it falls short of the performance achieved by percentage synchronization. Despite not matching the efficiency of percentage

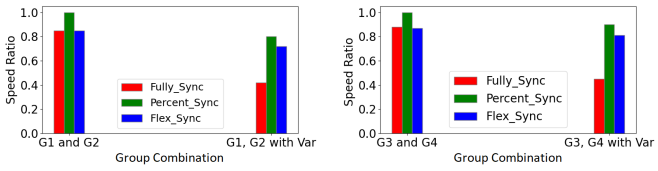

Fig. 6. Heterogeneous CPU Experiment Result

synchronization, the runtime management module remains effective in addressing the variances or turbulence within the distributed system. This suggests that the runtime flexible

synchronization can serve as a valuable tool for mitigating the impact of disruptions in a distributed computing environment.

B. Heterogeneous GPU environment

We have conducted tests on four distinct sets under varied group conditions. Across the board, the performance of full synchronization shows a notable decline when confronted with huge differences in computer power or substantial variances in the network. This decline was particularly evident in scenarios such as G1 and G2 with network variance or G1 and G3. In contrast, percentage synchronization showcased greater resilience by adeptly regulating the synchronization rate, especially in the presence of slower connections, whether in computation or transfer. Flexible synchronization, operating as a runtime method to govern the synchronization rate, proved to be a viable alternative. It demonstrated an ability to sustain performance levels when compared to the more resource-intensive full synchronization. This underscores the effectiveness of adopting a flexible strategy, particularly in environments characterized by computational disparities or network variations, where it can effectively mitigate unnecessary costs and optimize overall system performance.

TABLE II
HARDWARE CONFIGURATIONS OF HETEROGENEOUS GPUS

Group	***Nodes***	***GPU***	***Network Adapter***
group1	node 0-1	Nvidia P6000	1Gbps
group2	node 2	Nvidia P6000	1Gbps
group3	node 3	Nvidia 3060Ti	1Gbps

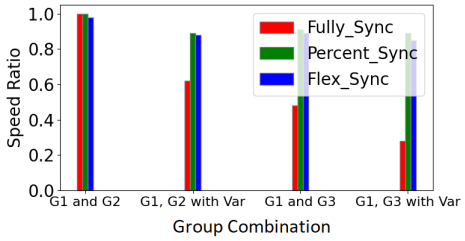

Fig. 7. Heterogeneous GPU Experiment Result

V. CONCLUSION AND FUTURE WORK

A dynamically hybrid and hierarchical logical architecture for cluster nodes has been introduced to address adaptivity and scalability issues in distributed neural network training. All the software, hardware, and network variables in the whole system were design with consideration to using the different layers of classification and dynamic runtime system. We have been able to utilize the hierarchical hardware resources more efficiently in an uncertain network environment.

Work is still needed in the future. First, the algorithm which is used in the runtime system to control the objects and frequency of the synchronization can still be improved. Even though we have improved the running time to convergence, there is still some room for improvement to reach the optimal value. More effort should be put to bear on exploring more efficient algorithms. Machine learning strategy can also be brought into consideration.

Second, the experiments conducted so far may not fully represent exascale scenarios, despite considering scale factors. Future research could benefit from the inclusion of additional hardware resources, such as multiple clusters or multiple data centers, to achieve a more comprehensive understanding of system behavior. Additionally, the current simplification of network conditions might not capture all the intricacies of the model. Therefore, future efforts should consider a more detailed examination of various network variables to enhance the accuracy and reliability of the entire system.

REFERENCES

[1] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, *et al.*, "Large scale distributed deep networks," *Advances in neural information processing systems*, vol. 25, 2012.

[2] R. Mayer and H.-A. Jacobsen, "Scalable deep learning on distributed infrastructures: Challenges, techniques, and tools," *ACM Computing Surveys (CSUR)*, vol. 53, no. 1, pp. 1–37, 2020.

[3] 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 *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, pp. 265–283, 2016.

[4] W. Jiang, G. Ye, L. T. Yang, J. Zhu, Y. Ma, X. Xie, and H. Jin, "A novel stochastic gradient descent algorithm based on grouping over heterogeneous cluster systems for distributed deep learning," in *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pp. 391–398, IEEE, 2019.

[5] Y. Kim, H. Choi, J. Lee, J.-S. Kim, H. Jei, and H. Roh, "Towards an optimized distributed deep learning framework for a heterogeneous multi-gpu cluster," *Cluster Computing*, vol. 23, pp. 2287–2300, 2020.

[6] S. Ouyang, D. Dong, Y. Xu, and L. Xiao, "Communication optimization strategies for distributed deep neural network training: A survey," *Journal of Parallel and Distributed Computing*, vol. 149, pp. 52–65, 2021.

[7] P. Sun, Y. Wen, R. Han, W. Feng, and S. Yan, "Gradientflow: Optimizing network performance for large-scale distributed dnn training," *IEEE Transactions on Big Data*, vol. 8, no. 2, pp. 495–507, 2019.

[8] J. Cipar, Q. Ho, J. K. Kim, S. Lee, G. R. Ganger, G. Gibson, K. Keeton, and E. Xing, "Solving the straggler problem with bounded staleness," in *14th Workshop on Hot Topics in Operating Systems (HotOS XIV)*, 2013.

[9] T. Castiglia, A. Das, and S. Patterson, "Multi-level local sgd: Distributed sgd for heterogeneous hierarchical networks," in *International Conference on Learning Representations*, 2020.

[10] T. Geng, M. Amaris, S. Zuckerman, A. Goldman, G. R. Gao, and J.-L. Gaudiot, "A profile-based ai-assisted dynamic scheduling approach for heterogeneous architectures," *International Journal of Parallel Programming*, vol. 50, no. 1, pp. 115–151, 2022.

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

[12] S. G. Kwak and J. H. Kim, "Central limit theorem: the cornerstone of modern statistics," *Korean journal of anesthesiology*, vol. 70, no. 2, pp. 144–156, 2017.

[13] S. Zeng, J. M. M. Diaz, and S. Raskar, "Toward a high-performance emulation platformfor brain-inspired intelligent systemsexploring dataflow-based execution model and beyond," in *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, vol. 2, pp. 628–633, IEEE, 2019.

[14] T. Geng, S. Zuckerman, J. Monsalve, A. Goldman, S. Habib, J.-L. Gaudiot, and G. R. Gao, "The importance of efficient fine-grain synchronization for many-core systems," in *International Workshop on Languages and Compilers for Parallel Computing*, pp. 203–217, Springer, 2016.