A Design of Distributed Training Workflow for Hybrid Cloud

Tuan Anh Vuong
School of Electronic Engineering
Soongsil University
Seoul, Korea
anhvt@dcn.ssu.ac.kr

Thai Nguyen Duc Thong
School of Electronic Engineering
Soongsil University
Seoul, Korea
thainguyen1309@dcn.ssu.ac.kr

Younghan Kim
School of Electronic Engineering
Soongsil University
Seoul, Korea
younghak@ssu.ac.kr

***Abstract*—Distributed training is a key approach to accelerating the training of large deep learning models. However, deploying and managing these workloads in hybrid cloud systems, which combine on-premise and cloud resources, introduces significant challenges, such as resource heterogeneity and communication overhead. In this paper, we present a distributed training workflow that focuses on evaluating the feasibility of addressing these two challenges simultaneously. Our approach leverages proportional batch size allocation to balance workloads across GPUs with different capabilities and employs gradient compression to reduce the volume of data exchanged during synchronization. Preliminary experiments on a hybrid cloud testbed demonstrate that these techniques can improve training throughput and reduce time-to-accuracy, highlighting their potential to enhance efficiency in realistic multi-cloud scenarios.**

***Index Terms*—distributed training, cloud computing, hybrid cloud environment, deep learning, heterogeneous system**

I. Introduction

The rapid advancement of deep learning models has driven an increasing demand for distributed training systems that can scale across large GPU clusters. Although public clouds provide elasticity, many organizations adopt hybrid cloud environments, which encompass on-premise and cloud resources, to balance cost, performance, and security. Training in such environments introduces a variety of challenges [1]–[3], among which the most notable considerations include:

- **Communication overhead**: the cost of exchanging gradients or parameters among distributed workers, which can become a bottleneck when bandwidth is limited.
- **Resource heterogeneity**: the system consists of heterogeneous devices that differ in computing power, memory size, and communication bandwidth, etc. These variations can hinder workload distribution and training efficiency.
- **Security and privacy**: the need to protect data confidentiality and maintain model integrity when sensitive information is transmitted between different devices.
- **Fault tolerance**: distributed training is more prone to failures than single-node setups, requiring mechanisms to recover and resume training without disrupting progress.
- **Scalability**: the capacity of the system to maintain efficient performance and near-linear speedup as the number of computing resources increases.
- **Resource management and scheduling**: the process of dynamically allocating computational resources to optimize performance, cost, utilization, and other factors.
- **Debugging and troubleshooting**: identifying and resolving issues in large-scale distributed jobs, where scattered logs and metrics require effective monitoring and visualization to ensure training reliability.

Given the range of considerations, addressing all of them concurrently in a unified architecture can be non-trivial, although various solutions exist to mitigate these challenges. In this paper, we focus on evaluating the feasibility of addressing two key challenges, heterogeneous GPUs and communication overhead, which are particularly relevant for hybrid cloud environments. Specifically, we propose a distributed training workflow that leverages proportional batch size allocation to balance workloads across diverse GPUs and gradient compression to reduce the amount of data exchanged during synchronization. Afterwards, we conduct experiments to investigate the effectiveness of our approach, summarize key findings, and outline potential directions for future research and system improvements.

This work is implemented on Ray [4], a high-performance distributed computing framework, and deployed on Kubernetes [5] for cluster orchestration. This setup serves as the testbed for evaluating the proposed workflow. The rest of this article is organized as follows. Section II reviews background and motivation. Section III presents the design of our distributed training platform. Section IV discusses the experimental evaluation. Finally, Section V concludes and outlines directions for future work.

II. Background and Motivation

A. Distributed deep learning training

Training a deep learning model involves optimizing many parameters through iterative updates. The process consists of three stages: *forward propagation* to compute predictions and loss, *backward propagation* to calculate gradients, and *parameter update* to adjust weights using optimizers such as SGD [6] or Adam [7]. As models and datasets grow, training on a single machine becomes inefficient. Distributed training leverages multiple resources to speed up learning, and this

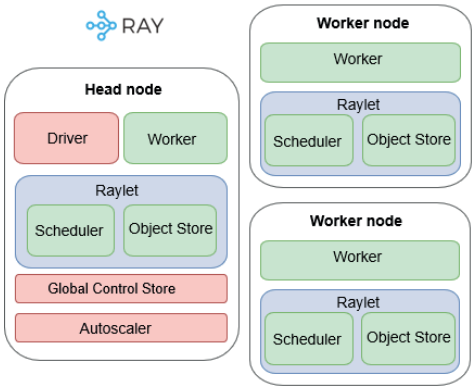


Fig. 1: Ray cluster architecture.

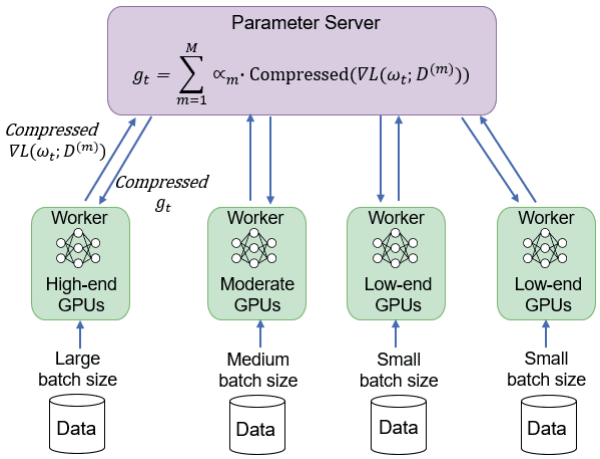


Fig. 2: Distributed training workflow overview.

paper focuses on *data parallelism*, the most widely adopted approach in such environments. In data-parallel distributed learning, each worker maintains a full model replica and processes a distinct mini-batch of data. During synchronous training, gradients computed by all workers are aggregated to form a global gradient, which is then used to update the model parameters with a chosen optimizer.

B. Communication architecture

Distributed training architectures can be broadly categorized into two families: Parameter Server [8] and All-Reduce [9]. In Parameter Server, workers compute gradients and send them to servers. Servers collect gradients from all workers, combine them to update the global parameters, and broadcast the updated model for the next training round. By contrast, All-Reduce follows a decentralized design, where gradients from every GPU are aggregated using collective operations such as Ring All-Reduce, after which each GPU updates its local parameters based on the aggregated result.

C. Ray on Kubernetes

Ray is an open-source framework for scalable AI and machine learning applications. Running Ray on a single node is simple, but scaling to multiple nodes requires a Ray cluster [10], which consists of several worker nodes connected to a common head node (Fig. 1). In addition, Kubernetes offers powerful resource orchestration, thereby providing an ideal platform for running Ray clusters at scale [11]. This setup allows users to efficiently leverage compute resources without requiring deep expertise in distributed systems and infrastructure management.

D. Motivation

Distributed training in hybrid cloud environments faces many obstacles, among which heterogeneous GPUs and communication bottlenecks are significant and practical to optimize. First, assigning equal batch sizes to heterogeneous GPUs leads to inefficient utilization, as faster GPUs must wait for slower ones, reducing overall system efficiency. A proportional batch size allocation aligns each device's workload with its capacity, mitigating this imbalance. Second, communication bottlenecks arise when synchronizing gradients and parameters across distributed nodes, especially with limited inter-node bandwidth. Gradient compression [12]–[16], [20]–[22] can reduce communication volume, thereby improving synchronization efficiency. This paper assesses the feasibility of integrating proportional batch size allocation and gradient compression into a unified workflow and evaluates how they jointly improve training efficiency under realistic hybrid conditions.

III. System Design

We design a hybrid cloud distributed training workflow (Fig. 2) based on the Parameter Server architecture. The centralized server aggregates gradients on CPUs, while parameter updates occur on GPUs, alleviating CPU bottlenecks and improving end-to-end training speed [17]. To further enhance efficiency, we adjust batch sizes according to GPU capabilities and apply gradient compression. These strategies help optimize hardware utilization while mitigating the communication overhead inherent in such diverse environments.

A. Proportional batch size allocation

A critical challenge in hybrid cloud environments is the presence of heterogeneous GPUs with varying computational capabilities. Assigning the same batch size to all workers leads to load imbalance, where slower GPUs delay global synchronization and reduce overall training throughput. To address this issue, our system allocates batch sizes proportionally based on the floating-point operations per second (FLOPs) of each GPU, while preserving a fixed global batch size, as it is considered an important training hyperparameter. Specifically, faster GPUs process larger batches, and lighter workloads are assigned to less powerful GPUs. The FLOPs specifications of different GPU models are publicly available in the official documentation provided by NVIDIA [18].

Since each worker processes different samples, directly averaging their gradients at the parameter server would bias the global update toward workers with smaller batches. Instead,

gradients are aggregated proportionally to the batch size of each worker, as defined in (1):

$$\alpha_m = \frac{n_m}{\sum_{i=1}^{M} n_i} \tag{1}$$

where α_m is the aggregation weight for worker m, n_m is the batch size of worker m, and $\sum_{i=1}^{M} n_i$ is the global batch size across all M workers. The global gradient at iteration t is then calculated as (2):

$$g_t = \sum_{m=1}^{M} \alpha_m \cdot \nabla L\left(w_t; D^{(m)}\right) \tag{2}$$

where $\nabla L(w_t; D^{(m)})$ is the gradient of the loss function L with respect to parameters w_t, computed on the local dataset $D^{(m)}$ of worker m. Finally, the new global model parameters are updated via SGD according to (3), in which η is the learning rate:

$$w_{t+1} = w_t - \eta \cdot g_t \tag{3}$$

This approach ensures that the global batch size is retained while preserving the convergence properties of the distributed optimization algorithm [19]. In our implementation, each gradient $\nabla L(w_t; D^{(m)})$ is compressed before aggregation, as described in Section III-B.

B. Gradient compression

In hybrid cloud environments, communication overhead can arise not only across WAN connections but also within local clusters when scaling to many GPUs. To mitigate this, our system employs Top-k sparsification, as it is simple and effective [15], [20]. Initially, each worker compresses its gradients locally and sends only the top $(100 - R_m^t)\%$ by magnitude to the server for global aggregation, where $R_m^t\,\%$ is the compression ratio for worker m in iteration t. To balance communication and mitigate stragglers, R_m^t is adaptively adjusted based on each worker's previous transmission time relative to the average of all workers. Specifically, workers with shorter times use lighter compression, while those with longer times apply more aggressive compression. To recover information lost during compression, error compensation [14] is employed, where discarded gradient values are accumulated in a residual buffer and added back to the gradients in the following iterations.

Although each worker transmits sparsified local gradients to the parameter server to reduce communication cost, the aggregated gradient g_t often becomes dense. To further save bandwidth, this aggregated gradient must be compressed before being sent back to the workers [21], [22]. In our design, Top-k sparsification is applied, where k is chosen to be approximately one thirty-second of the total number of model parameters. Moreover, it is essential to apply error compensation directly at the parameter server, since only the server has access to the history of compression errors and can properly correct them during aggregation.

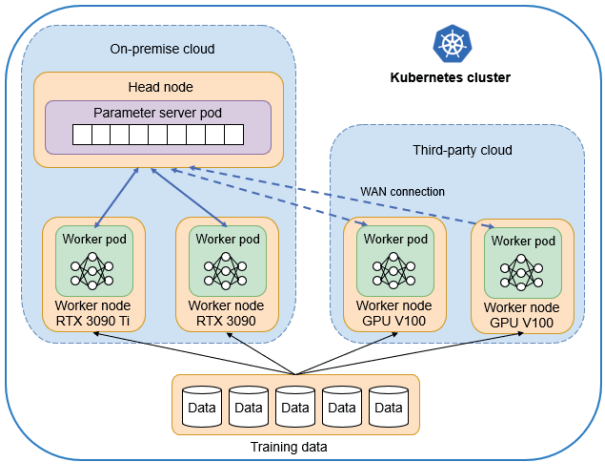


Fig. 3: Experimental system architecture.

IV. Experimental Evaluation

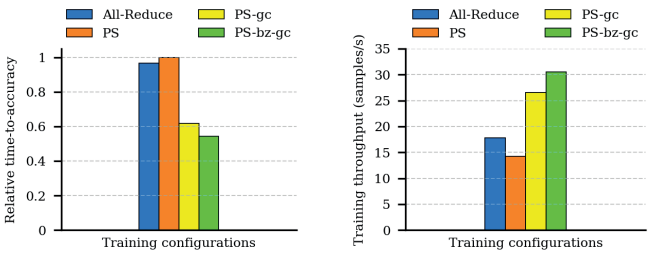
A. Cluster setup

We conduct experiments on a Kubernetes cluster of four nodes locally, assuming a hybrid cloud setup with two on-premise nodes (one with an RTX 3090 Ti and one with an RTX 3090) and two public cloud nodes (each with a V100 GPU), as shown in Fig. 3. To emulate a hybrid cloud environment, the network bandwidth is restricted. While intra-environment communication (within on-premise or within public cloud) is limited to 1 Gbps, communication between on-premise and public cloud nodes is limited to 100 Mbps to mimic WAN connections. The centralized server in the Parameter Server architecture is deployed on one of the on-premise nodes and is limited to 3 CPU cores.

We benchmark four distributed training configurations on the ResNet-152 model [23] with the ImageNet-100 dataset [24]. These include All-Reduce, Parameter Server (PS), PS with gradient compression (PS-gc), and our proposed PS with both gradient compression and proportional batch size allocation (PS-bz-gc). All experiments run for up to 100 epochs with an initial per-worker batch size of 64 and use SGD as the optimizer. The evaluation metrics consist of time-to-accuracy (TTA), training throughput, and loss trends.

B. Experimental results

Fig. 4a shows that, compared with standard PS, PS-bz-gc achieves the best performance with a 46% lower time-to-accuracy, PS-gc achieves a 38% reduction, and All-Reduce shows only a minor gain. In terms of training throughput, Fig. 4b demonstrates that PS-bz-gc reaches about 30.5 samples per second, more than double that of PS and 1.7× higher than that of All-Reduce. Fig. 5 indicates that although gradient compression slightly slows early loss reduction, this technique maintains similar convergence behaviour compared with other baselines in our benchmark. These results highlight the effectiveness of combining proportional batch sizing with gradient compression to mitigate communication overhead and improve resource utilization in hybrid cloud environments.



(a) Time-to-accuracy. (b) Training throughput.

Fig. 4: TTA and Training throughput.

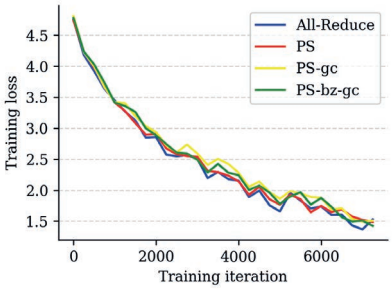


Fig. 5: Training loss over iterations.

V. Conclusion and Future Work

In this work, we present a distributed training workflow for hybrid cloud environments that addresses resource heterogeneity and communication overhead. By combining proportional batch sizing and gradient compression, the system balances workloads across diverse GPUs while reducing synchronization cost. Experiments on a hybrid cloud testbed show that our approach achieves more than 2× higher training throughput and up to 46% shorter time-to-accuracy than the standard PS setup. Future work could explore scaling to larger GPU clusters, integrating adaptive scheduling for runtime efficiency, and incorporating elastic and fault-tolerant mechanisms to enhance the robustness of production systems.

Acknowledgment

This work was partly supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) - ITRC (Information Technology Research Center) grant funded by the Korea government (MSIT) (IITP-2025-RS-2023-00258649, 50) and IITP grant funded by the Korea government (MSIT) (RS-2024-00398379, 50).

References

[1] Khan, Siffat Ullah, Habib Ullah Khan, Naeem Ullah, and Rafiq Ahmad Khan. "Challenges and their practices in adoption of hybrid cloud computing: An analytical hierarchy approach." Security and Communication Networks 2021, no. 1 (2021): 1024139.

[2] Verbraeken, Joost, Matthijs Wolting, Jonathan Katzy, Jeroen Kloppenburg, Tim Verbelen, and Jan S. Rellermeyer. "A survey on distributed machine learning." Acm computing surveys (csur) 53, no. 2 (2020): 1-33.

[3] Threadgill, Madison, and Andreas Gerstlauer. "A Survey of Distributed Learning in Cloud, Mobile, and Edge Settings." arXiv preprint arXiv:2405.15079 (2024).

[4] Moritz, Philipp, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol et al. "Ray: A distributed framework for emerging AI applications." In 13th USENIX symposium on operating systems design and implementation (OSDI 18), pp. 561-577. 2018.

[5] Kubernetes: an open source system for automating deployment, scaling, and management of containerized applications. URL: https://kubernetes.io/. Last accessed: October 2025.

[6] Zinkevich, Martin, Markus Weimer, Lihong Li, and Alex Smola. "Parallelized stochastic gradient descent." Advances in neural information processing systems 23 (2010).

[7] Kingma, Diederik P., and Jimmy Lei Ba. "Adam: A method for stochastic gradient descent." In ICLR: international conference on learning representations, pp. 1-15. 2015.

[8] Andersen, DG LiMu, and I. W. Park. "Scaling Distributed Machine Learning with the Parameter Server." In Proceedings of the Operating Systems Design and Implementation (OSDI). 2014.

[9] Patarasuk, Pitch, and Xin Yuan. "Bandwidth optimal all-reduce algorithms for clusters of workstations." Journal of Parallel and Distributed Computing 69, no. 2 (2009): 117-124.

[10] Ray Clusters Overview, URL: https://docs.ray.io/en/latest/cluster/getting-started.html. Last accessed: October 2025.

[11] Kanso, Ali, Edi Palencia, Kinshuman Patra, Jiaxin Shan, Mengyuan Chao, Xu Wei, Tengwei Cai, Kang Chen, and Shuai Qiao. "Designing a kubernetes operator for machine learning applications." In Proceedings of the Seventh International Workshop on Container Technologies and Container Clouds, pp. 7-12. 2021.

[12] Aji, Alham Fikri, and Kenneth Heafield. "Sparse communication for distributed gradient descent." arXiv preprint arXiv:1704.05021 (2017).

[13] Horvóth, Samuel, Chen-Yu Ho, Ludovit Horvath, Atal Narayan Sahu, Marco Canini, and Peter Richtárik. "Natural compression for distributed deep learning." In Mathematical and Scientific Machine Learning, pp. 129-141. PMLR, 2022.

[14] Seide, Frank, Hao Fu, Jasha Droppo, Gang Li, and Dong Yu. "1-bit stochastic gradient descent and its application to data-parallel distributed training of speech DNNs." In Interspeech, vol. 2014, pp. 1058-1062. 2014.

[15] Stich, Sebastian U., Jean-Baptiste Cordonnier, and Martin Jaggi. "Sparsified SGD with memory." Advances in neural information processing systems 31 (2018).

[16] Xu, Hang, Chen-Yu Ho, Ahmed M. Abdelmoniem, Aritra Dutta, El Houcine Bergou, Konstantinos Karatsenidis, Marco Canini, and Panos Kalnis. "Compressed communication for distributed deep learning: Survey and quantitative evaluation." (2020).

[17] Jiang, Yimin, Yibo Zhu, Chang Lan, Bairen Yi, Yong Cui, and Chuanxiong Guo. "A unified architecture for accelerating distributed DNN training in heterogeneous GPU/CPU clusters." In 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20), pp. 463-479. 2020.

[18] NVIDIA Deep Learning Performance Documentation, URL: https://docs.nvidia.com/deeplearning/performance/, 2023.

[19] Ferdinand, Nuwan, Haider Al-Lawati, Stark C. Draper, and Matthew Nokleby. "Anytime minibatch: Exploiting stragglers in online distributed optimization." arXiv preprint arXiv:2006.05752 (2020).

[20] Lin, Yujun, Song Han, Huizi Mao, Yu Wang, and William J. Dally. "Deep gradient compression: Reducing the communication bandwidth for distributed training." arXiv preprint arXiv:1712.01887 (2017).

[21] Wangni, Jianqiao, Jialei Wang, Ji Liu, and Tong Zhang. "Gradient sparsification for communication-efficient distributed optimization." Advances in Neural Information Processing Systems 31 (2018).

[22] Tang, Hanlin, Chen Yu, Xiangru Lian, Tong Zhang, and Ji Liu. "Doublesqueeze: Parallel stochastic gradient descent with double-pass error-compensated compression." In International Conference on Machine Learning, pp. 6155-6165. PMLR, 2019.

[23] He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition." In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770-778. 2016.

[24] Tian, Yonglong, Dilip Krishnan, and Phillip Isola. "Contrastive multiview coding." In European conference on computer vision, pp. 776-794. Cham: Springer International Publishing, 2020.