Journal of Intelligent Computing and Networking
https://www.ffspub.com/index.php/jicn/index
ISSN 3079-9228 (print)
E-mail: jicn.office@ffspub.com

Article

Optimizing Distributed Training in Computing Power Networks: A Heterogeneity-aware Approach

Zhe Deng[1], Renchao Xie[1,2,3,†], Qinqin Tang[1,3] Li Feng[1], Zeru Fang[4], Hongliang Lu[5], Tao Huang[1,2]
[1] *State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China*
[2] *Purple Mountain Laboratories, Nanjing 2111111, China*
[3] *Guangdong Laboratory of Artificial Intelligence and Digital Economy (SZ), Shenzhen 518060, China*
[4] *Department of Electrical and Electronic Engineering, Imperial College London, London SW7 2AZ, UK*
[5] *Unit 61905 of PLA, Shenyang 110005, China*
[†] *E-mail: Renchao_xie@bupt.edu.cn*

Received: October 5, 2025 / Revised: December 2, 2025 / Accepted: December 5, 2025 / Published online: December 16, 2025

Abstract: Distributed training over computing power network (CPN) suffers from high round-trip times (RTTs), volatile bandwidth, heterogeneous accelerators, and non-independent identically distributed (non-IID) data, which collectively lead to imbalanced collaboration and slow convergence. To address these issues, we propose a heterogeneity-aware distributed training framework tailored for wide area network (WAN) settings. It adopts a hybrid hierarchical synchronization mechanism that redistributes communication load temporally and spatially, thus improving end-to-end training efficiency. In addition, we design a heuristic algorithm for optimizing the synchronization topology, guided by the computational capacities of training nodes and real-time network telemetry. We implement a distributed training system and validate the proposed framework through extensive simulations, demonstrating consistent performance gains across diverse heterogeneous settings.

Keywords: Distributed training; synchronous communication mechanism; resource heterogeneity; computing power network; combinatorial optimization

1 Introduction

With the rapid advancement of artificial intelligence, the scale and complexity of machine learning models continue to grow, driving an ever-increasing demand for computing resources [1]. Against this backdrop, distributed training has become essential for large-scale model development. By parallelizing training across multiple training nodes, it leverages geographically dispersed computational and data resources, significantly reducing training time and improving overall efficiency.

As a new networking paradigm, the computing power network (CPN) unifies and orchestrates dispersed computing, storage, and networking resources across wide-area domains [2]. While CPN offers elastic capacity and data proximity, real-world deployments exhibit strong heterogeneity across multiple dimensions—including compute resources, communication capabilities, and data distributions—which poses severe efficiency challenges for distributed collaborative training across nodes [3].

In general, distributed training in the CPN faces three core challenges:
1) Transmission latency: In an end-edge-cloud framework, high round-trip times (RTTs) and unstable bandwidth often dominate end-to-end training cost. Heterogeneous access and link technologies (e.g., Wi-Fi [4] vs. RDMA [5]) introduce capacity disparities and volatility, leading to communication heterogeneity and imbalanced collaboration. Moreover, due to firewalls, Network Address Translation (NAT), and protocol constraints, cross-domain reachability and stable routing cannot be guaranteed in advance, further increasing the complexity of routing and session management. Even in an ideal environment equipped with NVIDIA A800 GPU and 10 Gbps high-speed bandwidth for full parameter fine-tuning of the GPT-2 model, a single round of weight transmission can still

[†] Corresponding author: Renchao Xie
[*] Academic Editor: Chunxiao Jiang

take up to 45.9 seconds, accounting for 44.97% of the total training time. Furthermore, with an RTT of 20ms, the communication overhead can increase by up to 28.4× [6].

2) Resource heterogeneity and dynamics: In each training round, the model strictly follows the sequence of forward pass, backward computation, and parameter update; the next round can only proceed after the communication in the current round is completed. This strong dependency results in a largely sequential execution of computation and communication. However, in synchronous data-parallel training, resource heterogeneity and runtime variability across nodes intensify straggler effects. Because of substantial hardware and network disparities, effective training throughput can differ by orders of magnitude across common accelerator and interconnect configurations, leading to prolonged synchronization stalls and reduced cluster-wide utilization.

3) Significant data heterogeneity: In distributed training scenarios across multiple data centers, each training node exhibits distinct data personalization, with significant differences in the number of samples, data quality, and sample distribution, resulting in a pronounced non-independent and identically distributed (non-IID) data issue. Consequently, this leads to conflicting update directions of the local models and degrades the convergence speed and accuracy of the global model. A typical example is the cross-region COVID-19 diagnosis: the severity of the epidemic and case types in different regions are recorded differently, resulting in extreme label skew in the distribution of the node data [7]. This skew makes it difficult for the global model to form an effective recognition capability for rare case types after aggregation, which ultimately seriously impairs the global generalization performance of the model [8].

Existing optimization techniques do not fully address the core challenge posed by wide-area distributed training, where communication latency and computational efficiency are nonlinearly coupled in a highly heterogeneous, dynamic network. Based on the background and challenges outlined above, this paper proposes a heterogeneity-aware approach to optimize distributed training in CPNs, which adaptively adjusts the synchronous communication topology of each node according to the resource state, leverages edge-side computing capacity to disperse core-network hotspots and improves the overall resource utilization of the system. The main contributions of this paper can be summarized as follows:

- We design a heterogeneity-aware distributed training framework. For data transmission, this framework adopts a hierarchical progressive aggregation strategy and performs along-the-path partial aggregation on tree topologies. For model updating, the training node applies the data resampling strategy to counter non-IID skew, while the global parameter server applies a dynamic, fairness-aware step-size update to stabilize and accelerate convergence.
- We formulate a synchronization topology optimization problem that accounts for communication overhead and synchronization consistency requirements. And we design a heuristic synchronization topology optimization algorithm to minimize idle waiting and communication overhead.
- We deployed the proposed distributed training system across multiple machines and validated the feasibility and performance of its synchronization mechanism. Experimental results demonstrate stable and substantial training acceleration over mainstream synchronous communication mechanisms across diverse heterogeneous settings.

The rest of this paper is organized as follows. Section 2 reviews recent advances in distributed training optimization. Section 3 presents the proposed heterogeneity-aware training framework. Section 4 formalizes the optimization objective. Section 5 introduces a heuristic algorithm for generating global synchronization communication topologies. Section 6 presents the experiments and simulations, and the conclusion is summarized in Section 7.

2 Related Work

2.1 Mainstream Synchronous Communication Architectures for Distributed Training

Traditional distributed training often relies on the Parameter Server model, where worker nodes send gradients to a central server for aggregation. This server then distributes updated parameters back to the workers. This scheme is simple and easy to deploy, but it is prone to forming communication bottlenecks and also faces the risk of a single point of failure of the parameter server [9].

To alleviate the centralized bandwidth bottleneck, a decentralized ring-allreduce topology has gradually emerged. The core idea of the ring topology is to arrange all the nodes involved in the computation into a ring in a logical order, and the data is passed and aggregated gradually among neighboring nodes. Han et al. proposed a fair distributed training framework based on the ring architecture, called RingFFL [10]. RingFFL achieves efficient parameter aggregation through the ring topology while guaranteeing fairness among the participants. The framework strikes a good balance between communication cost and aggregation efficiency, making it suitable for large-scale distributed training. However, this ring topology has a strong serial dependency and is particularly sensitive to latency and bandwidth differences in inter-node communication. Bandwidth or latency degradation at any node can slow down the entire ring. In distributed training, this architecture reduces the burden on the central node but requires high network conditions, especially in a WAN environment across data centers.

In response to the WAN bandwidth constraints and high latency problems exposed by star and ring topologies when training across data centers, academia and industry have favored a hierarchical decentralized communication architecture [11, 12]. This architecture utilizes a tree topology to accomplish in-place aggregation in close proximity to nodes, thereby reducing cross-domain traffic. It has been shown that spanning tree is one of the potentially optimal structures for efficient parameter synchronization. For example, Yuan et al. designed a LAN-based hierarchical distributed training platform to solve the communication bottleneck problem [13]. The platform avoids the use of intermediate edge servers by creating local area network (LAN) domain groups in P2P mode, and exploits the high bandwidth and low communication cost of LAN for model aggregation. The FedPAQ method proposed by Reisizadeh et al. improves the communication efficiency through periodic averaging and quantization [14]. The method allows edge nodes to participate in synchronization, which reduces the communication load and further reduces the communication overhead by quantizing update messages. Hosseinalipour et al. developed a multi-stage hybrid federated learning based on fog learning, which combines a tree topology with a mesh topology by collaborating between different network layers to form a local consensus, and a multi-stage tree hierarchy with multi-stage Parameter Transfer [15]. Most existing research fails to account for differences in network resource availability and nodes' arithmetic performance, and typically assumes that the underlying topology is symmetric and homogeneous. This leads to the use of rigidly structured balanced trees for parameter synchronization. Some studies have explored irregular tree topologies, but these efforts ignore the blocking latency of intermediate nodes in the optimization objective [16].

2.2 Resource Scheduling Optimization in Distributed Training

In order to reduce the high communication and system overhead of distributed training, various communication optimization techniques have been proposed in the industry [17], such as gradient compression and sparsification on the node side [18, 19], the adoption of efficient communication topologies (e.g., ring-allreduce [20], three-dimensional hybrid parallel architectures [21]) to improve the data synchronization efficiency, and confidence-based importance routing and priority scheduling [22, 23].

However, many communication optimizations for distributed training are tailored to intra–data-center networks, which are characterized by high bandwidth and low latency, and tend to underperform in WAN settings. Under high packet loss and jitter, sparsifying gradient compression can exacerbate stale or missing gradient chunks due to retransmissions [24]. Latency-bound synchronous collectives (e.g., ring all-reduce) lengthen time-to-consistency and convergence [25], while non-IID cross-datacenter data amplifies bias and undermines importance-based scheduling [26].

AI training traffic is typically characterized by a small number of synchronous bursts of large streams, resulting in low entropy. This leads to the failure of traditional Equal-cost Multi-path (ECMP) hash routing mechanisms. Traffic conflicts are severe, network links are unevenly busy or idle, and effective throughput can be as low as 20% to 50%. To address these challenges in bandwidth-constrained scenarios, gradient compression is commonly used to reduce the actual amount of communication data. Wang proposes an adaptive gradient compression algorithm that adjusts the gradient communication compression rate for distributed nodes based on multi-dimensional evaluation features in the training of distributed deep neural networks, thereby reducing training time under different network conditions [27]. A novel adaptive Top-K stochastic gradient descent (SGD) framework is proposed to achieve adaptive sparsity for each gradient descent step to maximize the convergence performance by balancing the trade-off between the communication cost of distributed training and the model convergence error [28]. The AdaSFL algorithm is proposed to theoretically analyze model convergence speed, and adaptively control the local update frequency and different batch sizes per node based on the upper bound of convergence associated with the local update frequency for a given data batch size to improve the training efficiency [29]. Ling proposes a group hierarchical architecture that fuses data parallelism (DP) and model parallelism (MP), considering the heterogeneity of communication and computational resources as well as the time dynamics triggered by the competition for resource sharing The architecture, by implementing two levels of aggregation at the group level and the system level, effectively reduces synchronization scales and relieves the traffic pressure on the bottleneck links, thereby improving the overall training efficiency [30].

In summary, unlike distributed computing with homogeneous nodes inside a single data center, hardware and resource heterogeneity can turn weak data centers or bandwidth-constrained links into bottlenecks for synchronous communication across WANs. This reveals the difference between parameter synchronous traffic optimization and traditional routing optimization problems [31]. Therefore, more efficient traffic scheduling algorithms need to be designed for large-scale distributed training scenarios.

3 System Description

In this section, we present the system architecture of a distributed training framework across WANs and explain the training paradigm and implementation details.

3.1 System Model

For wide-area heterogeneous arithmetic networks, we design a distributed training framework to support efficient cross-domain synchronous communication. The overall architecture is shown in Figure 1.

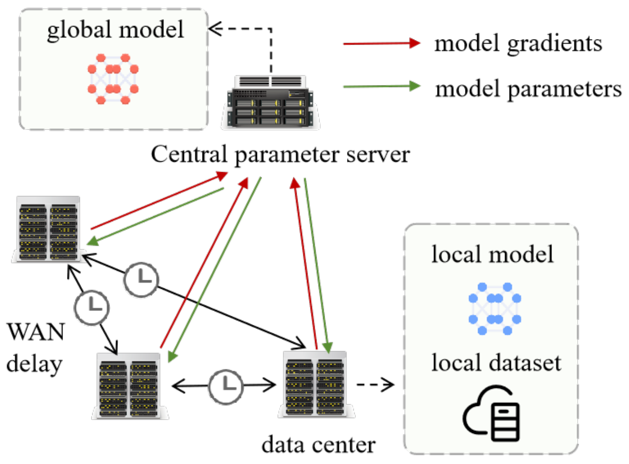


Figure 1: Distributed training paradigm in the CPN.

Figure 1 illustrates a distributed training system, comprising a set of geographically dispersed, heterogeneous data centers, denoted as $\mathbb{V} = \{1, 2, \dots, |\mathbb{V}|\}$. The sample set of data for each node is $\mathbb{S} = \{s_1, s_2, \dots, s_{|\mathbb{V}|}\}$. Data centers perform on-device training on their local datasets and periodically synchronize their model data. The global parameter server holds the master copy of the global model, serving as the central hub for all parameter updates within each synchronization cycle.

In the standard paradigm of synchronous data-parallel training, the global loss function is defined as:

$$F(\theta) = \frac{1}{|\mathbb{V}|} \sum_{v \in \mathbb{V}} p_v f_v(\theta; s_v), \tag{1}$$

where $p_v = \frac{|s_v|}{\sum_{u \in \mathbb{V}} |s_u|}$ denotes the data-size weight of node v, and $f_v(\cdot)$ is the loss function at node v.

In the τ-th global synchronization cycle, node v receives the latest global parameters θ_τ from the server as the initialization of its local model. Then, node v trains on its local dataset s_v and computes the current-round gradient $\nabla f_v(\theta_\tau; s_v)$. After completing local training, node v transmits its gradient to the server, which will subsequently aggregate these gradients to update the global model parameters. The updated global parameters are then synchronized back to all nodes as initial parameters for the next round of local training.

This periodic synchronous mechanism allows the data center to collaboratively train a global model without the need to collect raw data. However, the severe data heterogeneity and resource heterogeneity across nodes in the CPN pose a substantial challenge to model convergence efficiency.

3.2 Data β-resampling Mechanism

To address the model performance degradation caused by skewed or disjoint label spaces, we adopt a data resampling strategy based on the inverse effective number of samples [32]. This method aligns the class sampling probabilities across all nodes, thereby mitigating the adverse effects of non-IID data and promoting stable model convergence.

Let node v hold dataset s_v, and let the set of class labels be $\mathscr{Y} = \{1, 2, \ldots, Y\}$ with Y being the total number of classes. Suppose the label of the i-th sample in s_v is $Y(v,i)$. We define the sampling weight of this sample as

$$\gamma_{v,i} = \frac{1-\beta}{1-\beta^{N^v_{Y(v,i)}}}, \tag{2}$$

where $N^v_{Y(v,i)}$ denotes the number of samples with label $Y(v,i)$ in s_v, and $\beta \in [0,1)$ is a hyperparameter of resampling strength.

Then the probability that node v selects its i-th sample for training is

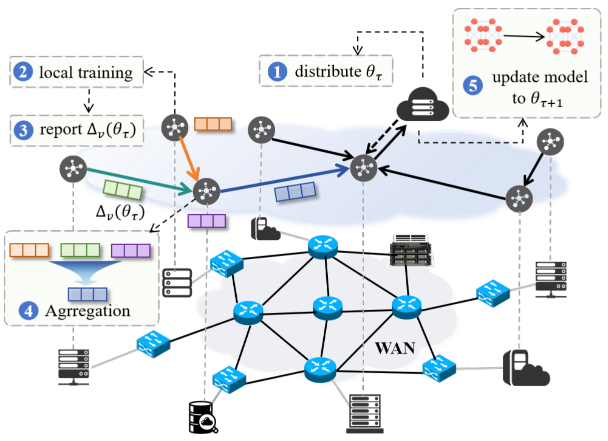
$$p_{v,i} = \frac{\gamma_{v,i}}{\sum_{l \in s_v} \gamma_{v,l}}. \tag{3}$$

This resampling mechanism enables each node to balance label distribution locally without accessing other nodes' data, thereby adhering to data privacy requirements.

3.3 Hierarchical Aggregation Transmission Mechanism

A hierarchical progressive aggregation strategy is adopted in the proposed distributed training framework to align the upper-layer synchronous communication logical topology with the underlying physical network.

Figure 2 illustrates the dataflow logic within a synchronization cycle, which consists of the following five key steps. First, the global parameter server distributes the latest model parameters to the root node. These parameters are then propagated hierarchically to all nodes in the tree, serving as initial parameters for their current training round. Each node performs local training in parallel to compute its own gradients. During the gradient synchronization phase, gradients are propagated upward from the leaf nodes and aggregated along the way. Each internal node receives gradients from its child nodes, performs local aggregation, and forwards the aggregated result to its parent node. Finally, the root node sends the domain-wide aggregated gradient to the global parameter server to update the global model. Upon receiving the uploaded gradients, the parameter server updates global model parameters and stores the latest snapshot. This hierarchical forwarding mechanism fully exploits the available bandwidth of idle links without increasing the total communication volume, effectively avoiding traffic hotspots and port contention. The resulting tree-like logical topology is referred to as the aggregation-tree.



1 distribute θ_τ
2 local training
3 report $\Delta_v(\theta_\tau)$
5 update model to $\theta_{\tau+1}$
$\Delta_v(\theta_\tau)$
4 Aggregation
WAN

Figure 2: Workflow of hierarchical aggregation transmission mechanism.

Given an aggregation-tree topology k_r with root node r, each internal node u performs a weighted aggregation of its own gradient and those from its children $\mathrm{Ch}(u)$. Let $k^{\mathrm{sub}}_{r\to u}$ denote the subtree of k_r rooted at u and $\mathbb{V}(k^{\mathrm{sub}}_{r\to u})$ denote the set of nodes in $k^{\mathrm{sub}}_{r\to u}$.

We define the aggregation weight $\omega_{u,n}$ as

$$\omega_{u,n} = \begin{cases} \dfrac{|s_n|}{\sum_{i \in \mathbb{V}(k^{\mathrm{sub}}_{r\to u})} |s_i|}, & \text{if } n = u \\[2ex] \dfrac{\sum_{i \in \mathbb{V}(k^{\mathrm{sub}}_{r\to n})} |s_i|}{\sum_{i \in \mathbb{V}(k^{\mathrm{sub}}_{r\to u})} |s_i|}, & \forall n \in \mathrm{Ch}(u) \end{cases}, \tag{4}$$

where $|s_i|$ is the sample size of node i.

The weighted aggregated gradient computed at node u, denoted by $\nabla^{\mathrm{agg}}_u(\theta_\tau)$ is given by

$$\begin{aligned} \nabla^{\mathrm{agg}}_u(\theta_\tau) &= \sum_{n \in \mathrm{Ch}(u)} \omega_{u,n} \nabla^{\mathrm{rep}}_n(\theta_\tau) + \omega_{u,u} \nabla_u(\theta_\tau) \\ &= \frac{\sum_{i \in \mathbb{V}(k^{\mathrm{sub}}_{r\to u})} \left(|s_i| \nabla_i(\theta_\tau)\right)}{\sum_{i \in \mathbb{V}(k^{\mathrm{sub}}_{r\to u})} |s_i|}, \end{aligned} \tag{5}$$

where $\nabla_n^{\text{rep}}(\theta_\tau)$ denotes the gradient reported by child node n. When $\text{Ch}(n) = \emptyset$, $\nabla_n^{\text{rep}}(\theta_\tau) = \nabla_n(\theta_\tau)$; otherwise, $\nabla_n^{\text{rep}}(\theta_\tau) = \nabla_n^{\text{agg}}(\theta_\tau)$.

Similarly, the final aggregated gradient submitted by node r to the global parameter server is

$$\nabla_r^{\text{rep}}(\theta_\tau) = \frac{\sum_{i \in \mathbb{V}(k_r)} \left(|s_i| \, \nabla_i(\theta_\tau)\right)}{\sum_{i \in \mathbb{V}(k_r)} |s_i|}. \tag{6}$$

3.4 Hybrid Asynchronous Aggregation Forest

However, the hierarchical aggregation topology remains vulnerable to stragglers; we therefore partition nodes by compute capacity and build a separate aggregation tree for each subset, forming an aggregation forest in which every tree carries the full gradient tensor.

Because different aggregation-trees have different synchronization periods, global updates are performed in an asynchronous manner across trees. As illustrated in Figure 3, each aggregation-tree performs synchronous local aggregation along its own routing paths, while the parameter server incorporates the updates from different trees whenever they arrive. As a result, each round of global parameter evolution only depends on the subset of nodes that has just finished synchronization. Allowing aggregation-trees to synchronize on their own schedules leads to asynchronous global updates and makes the system more tolerant of nodes with limited compute or network capacity. We term this scheme—synchronous aggregation within each tree and asynchronous updates across trees—the hybrid asynchronous aggregation forest (HAAF) synchronization mechanism.

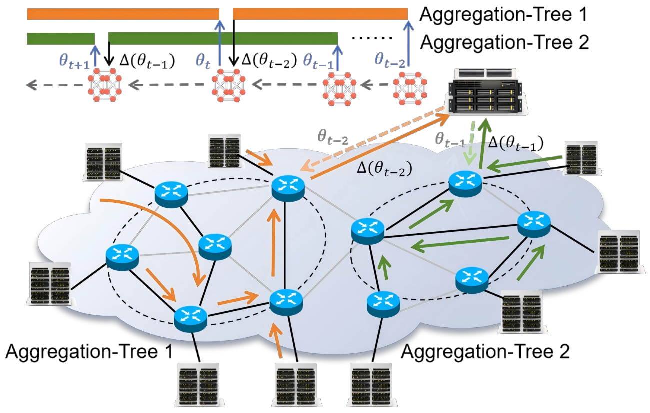


Figure 3: Framework of hybrid asynchronous aggregation forest.

To ensure an unbiased global update direction, we must account for each node's contribution to model improvement from multiple aspects, balancing efficiency and fairness.

1) Fairness weight function: Since aggregation-trees with higher-performance nodes update the model parameters more frequently, they can dominate training, causing the global model to drift toward their local data distributions and consequently hurting generalization. To counter this, we introduce a fairness weight function that increases the contribution of lower-performance, higher-loss nodes, thereby reducing accuracy variance.

In the early phase, high-frequency aggregation-trees, through rapid local iterations, pull the global parameters quickly toward the global optimum. Later, these trees stay near their local optima and produce small gradients, while low-frequency trees remain farther from their optima and yield larger gradients that help escape suboptimal local minima.

The global objective can be approximated as

$$\min_{\theta} \frac{1}{|\mathbb{K}|} \sum_{k \in \mathbb{K}} \frac{P_k}{q+1} \bar{f}_k^{\,q+1}(\theta), \tag{7}$$

where, analogous to data-size weights at the node level, $P_k = \frac{\sum_{i \in \mathbb{V}(k)} |s_i|}{\sum_{i \in \mathbb{V}} |s_i|}$ denotes the data-size weight of aggregation-tree k; $\mathbb{K}$ is the set of aggregation-trees, and $q \geq 0$ is the fairness factor hyperparameter. $\bar{f}_k(\theta) = \frac{\sum_{v \in \mathbb{V}(k)} f_v(\theta)}{|\mathbb{V}(k)|}$ is the average loss over all nodes in aggregation-tree k.

Theorem 1 *Assume the gradient of the nonnegative model loss function $f(\cdot)$ is L-smooth, i.e.,*

$$\forall x, y, \quad f(y) - f(x) \leq \langle \nabla f(x), y - x \rangle + \frac{L}{2} \|y - x\|^2.$$

It can be proven that for any $q \geq 0$ and any point θ, the function $\frac{1}{q+1} f^{q+1}(\cdot)$ has a local gradient Lipschitz constant at θ upper-bounded by

$$L_q(\theta) = L f(\theta)^q + q f(\theta)^{q-1} \|\nabla f(\theta)\|^2.$$

The detailed derivation is provided in the referenced literature [33].

Consequently, we introduce a fairness weight function ω_q^{fair} based on Theorem. 1

$$\begin{aligned} &\omega_q^{\text{fair}}\left(\nabla \bar{f}_{k_r}(\theta_\tau);\, \nabla_r^{\text{rep}}(\theta_\tau)\right) \\ &= \frac{L \bar{f}_{k_r}^{\,q}(\theta_\tau)}{L \bar{f}_{k_r}^{\,q}(\theta_\tau) + q \bar{f}_{k_r}^{\,q-1}(\theta_\tau) \|\nabla_r^{\text{rep}}(\theta_\tau)\|^2}, \end{aligned} \tag{8}$$

where L is a Lipschitz constant of the gradient of $f(\cdot)$. The fairness weight depends on the average loss and the aggregated gradient of aggregation-tree k_r, both of which evolve with the training dynamics and naturally induce an adaptive step.

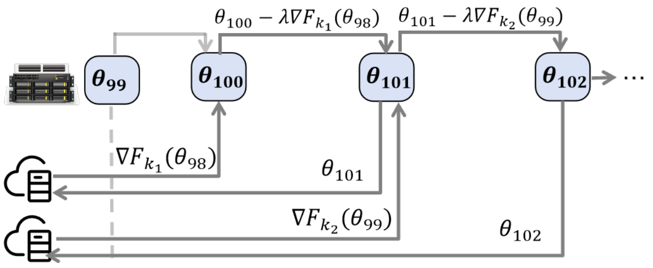

$\theta_{100} - \lambda \nabla F_{k_1}(\theta_{98})$
$\theta_{101} - \lambda \nabla F_{k_2}(\theta_{99})$
θ_{99}
θ_{100}
θ_{101}
θ_{102}
$\cdots$
$\nabla F_{k_1}(\theta_{98})$
θ_{101}
$\nabla F_{k_2}(\theta_{99})$
θ_{102}

Figure 4: Gradient staleness in asynchronous updates.

2) Staleness weight function: Due to the weak consistency among aggregation-trees in the asynchronous algorithm, gradient staleness is inevitable [34]. As shown in Figure 4, gradients arriving through slower aggregation paths are computed with model parameters from earlier rounds. This will induce oscillations in global training and degrade model accuracy. To mitigate such adverse effects, we introduce a polynomial staleness weight function ω_a^{stale} [35], defined as

$$\omega_a^{\text{stale}}(t;\tau) = (t-\tau+1)^{-a}, \tag{9}$$

where $a > 0$ is the staleness factor, controlling the decay rate; t denotes the current global synchronization round, and τ indicates that the aggregated gradient received by the parameter server at round t was computed using the global model parameters from round τ. When $t-\tau = 0$, there is no gradient staleness.

Thus, combining the fairness factor and staleness factor, the global parameter update can be expressed as

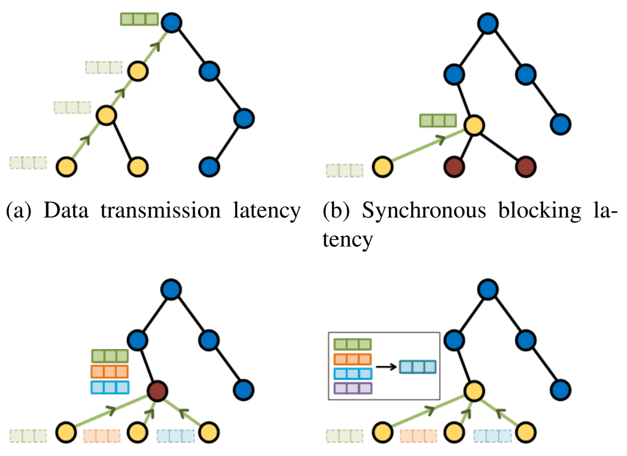
$$\begin{aligned} &\theta_{t+1} \\ &= \theta_t - P_{k_r} \cdot \omega_q^{\text{fair}}\left(\bar{f}_{k_r}(\theta_\tau); \nabla_r^{\text{rep}}(\theta_\tau)\right) \cdot \omega_a^{\text{stale}}(t;\tau) \cdot \nabla_r^{\text{rep}}(\theta_\tau) \\ &= \theta_t - \frac{\left(\sum_{i\in\mathbb{V}(k_r)} |s_i|\right) L\bar{f}_{k_r}^{q}(\theta_\tau)(t-\tau+1)^{-a}\nabla_r^{\text{rep}}(\theta_\tau)}{\sum_{i\in\mathbb{V}} |s_i| \left(L\bar{f}_{k_r}^{q}(\theta_\tau) + q\,\bar{f}_{k_r}^{q-1}(\theta_\tau)\,\|\nabla_r^{\text{rep}}(\theta_\tau)\|^2\right)} \end{aligned} \tag{10}$$

4 Problem Formulation

In this section, we analyze the routing optimization problem in the HAAF mechanism. First, we define synchronous delay into the following four types:

- Data transmission latency: The sum of link transmission latencies along the end-to-end communication path.
- Synchronization blocking latency: The stall time at non-leaf node while waiting for straggler children.
- Computation blocking latency: The stall time at non-leaf node due to unfinished local computation.
- Aggregation operation latency: The time consumption for the aggregation operation.

In summary, the synchronization latency of an aggregation-tree includes not only the transmission latency along the routing path, but also the latency caused by computing blocking due to straggler effect and aggregation operations. An illustration of the various latencies is given in Figure 5.


(a) Data transmission latency (b) Synchronous blocking latency

(c) Computation blocking latency (d) Aggregation operation latency

Figure 5: Schematic of synchronization latencies.

Since in real distributed training scenarios, the aggregation operation in distributed systems is done by highly optimized communication libraries (e.g., NCCL [36]), the processing latency is negligible compared to other types of latency. Therefore, for simplicity, node processing latency is not considered here.

The global parameter server updates the model parameters, and then pushes them back to the root node. We take the moment when the server broadcasts the updated parameters as the beginning of the synchronization cycle and denote the moment at which a node submits parameters to its parent as

$$T_{send}^{v} = \begin{cases} t_v + T_{start}^{v}, & \text{if } \mathrm{Ch}(v) = \emptyset \\ \max\left\{T_{\text{recv}}^{v},\ t_v + T_{start}^{v}\right\}, & \text{otherwise} \end{cases}, \tag{11}$$

where t_v is time consumption for local training and T_{start}^{v} is the moment when node v receive updated parameters and start training. $T_{\text{recv}}^{v} = \max_{n\in\mathrm{Ch}(v)}\left(T_{send}^{n} + d_{\text{trans}}(e_{n\to v})\right)$ denotes the moment when non-leaf node receives all children's data, and $d_{\text{trans}}(e_{n\to v})$ represents the transmission latency from node n to node v.

The synchronization period of aggregation-tree k_r can be expressed as

$$d_{k_r} = T_{send}^{r} + d_{\text{trans}}(r), \tag{12}$$

where $d_{\text{trans}}(r)$ is the transmission latency from root node r to the global parameter server.

Define the longest waiting time within a single cycle among all nodes as synchronization latency. The synchronization latency of aggregation-tree k_r is

$$l_{k_r} = d_{k_r} - \min_{v\in\mathbb{V}(k_r)} T_{send}^{v}. \tag{13}$$

To maximize system utilization while mitigating weak inter-node consistency under asynchronous updates, the mapping strategy for matching training nodes to the aggregation-tree must be specified meticulously. Let $\mathrm{X} = [\mathbf{x}_k]_{k\in\mathbb{K}}$ denotes a joint mapping strategy. Then the global optimization objective is defined as follows

$$\begin{aligned} \min_{\mathbf{X}} \quad & \lambda_1 \sum_{k\in\mathbb{K}} l_k(\mathbf{x}_k) \;+\; \lambda_2\,\mathrm{Std}\left(\left[d_k(\mathbf{x}_k)\right]_{k\in\mathbb{K}}\right), \\ \text{s.t.} \quad & \mathbf{x}_k = (x_{1,k}, x_{2,k}, \ldots, x_{|\mathbb{V}|,k})^{\top}, \quad \forall k\in\mathbb{K}, \\ & \sum_{k\in\mathbb{K}} x_{v,k} = 1, \qquad \forall v\in\mathbb{V}, \\ & x_{v,k} \in \{0,1\}, \qquad \forall v\in\mathbb{V},\ \forall k\in\mathbb{K}, \end{aligned} \tag{14}$$

where the first term of the global objective function is the total synchronization latency across all aggregation-trees, while the second term is the standard deviation of aggregation periods among the trees. $l_k(\mathbf{x}_k)$ and $d_k(\mathbf{x}_k)$ represent the synchronous latency and period of aggregation-tree k under its mapping strategy $\mathbf{x}_k$, respectively, which can be calculated based on Eq. (12) and (13). The binary variable $x_{v,k}$ indicates whether node v is assigned to aggregation-tree k. Specifically, $x_{v,k} = 1$ means node v belongs to aggregation-tree k. The coefficients λ_1 and λ_2 are weight factors used to balance dimensions.

5 Solution Algorithm

In this section, we tackle the optimization problem using a heuristic algorithm that constructs an optimal topology for synchronization communication logic.

5.1 Fastest Aggregation Topology Construction

Traditional shortest-path algorithms typically weight links by transmission latency to find routes with sufficient bandwidth to support fast end-to-end data flows while avoiding congestion on low-bandwidth links. However, the synchronous routing discussed here is a logical communication topology decoupled from the underlying physical bearer network. Traffic on an aggregation-tree is not a simple end-to-end stream; instead, it proceeds hop by hop and incurs synchronized blocking latencies at internal nodes.

Given a fixed node set for a tree, we design an algorithm to construct the synchronization communication logical topology solely from the perspective of data transmission latency, ignoring node-local computation time, as shown in Algorithm 1. Because synchronization latency is determined by the maximum leaf-to-root path latency, and the downstream (broadcast) and upstream (aggregation) phases are symmetric, the problem of finding the most efficient aggregation-tree topology reduces to a min–max objective: minimizing the latency of the slowest path.

Algorithm 1 iterates over all nodes to construct a set of aggregation-tree routing paths. For each ordered node pair (i, j) in graph $\mathscr{G} = (\mathbb{N}, \mathbb{E})$, we apply the shortest-path search algorithm (Dijkstra) to find the fastest aggregation path from node i to node j. The path is returned as a node sequence $p_{i\to j}$, with a time complexity of $O(N^2)$, N is the number of nodes in $\mathbb{N}$. If a node on the path does not belong to the node set of the corresponding tree, it is treated merely as a relay node and therefore is not recorded into the sequence $p_{i\to j}$. Collect these paths and record them in the set P. Each path in P represents the shortest aggregation path between a pair of nodes, ensuring that parameters can be synchronized along the route with minimal latency.

Algorithm 1 guarantees the shortest end-to-end path transfer latency from each node to the root node in the aggregation-tree topology. Moreover, when we extend the analysis to include node-local computational blocking, we can prove that the synchronization latency achieved by Algorithm 1 is no greater than that of any other topology with the same root, once per-node computational blocking is taken into account.

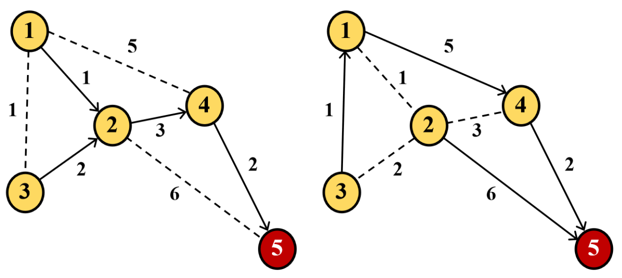
When node computational blocking is ignored, Figure 6(a), which is the output of Algorithm 1 for the given network, achieves lower synchronization latency than Figure 6(b). Furthermore, after incorporating computational blocking as in Figure 7, and fixing node 5 as the root while node 2 is the straggler, the difference in synchronization latency between the two aggregation-trees is determined solely by the time required for parameters to propagate from node 2 to the root; it is independent of the communication topology among node 2's predecessors. Therefore, the path from node 2 to the root must be the shortest path. In this case, multiple topologies may attain the same optimal value, but the aggregation topology produced by Algorithm 1 is guaranteed to belong to the optimal solution set.

Algorithm 1 Candidate Topologies Construction Algorithm

```
Require: Undirected graph $\mathscr{G} = (\mathbb{N}, \mathbb{E})$;
Link transmission latency matrix $[d(e)]_{e\in\mathbb{E}}$
Ensure: Candidate tree topology set $\mathscr{T} = [T_i]_{i\in\mathbb{N}}$
1: for $i \in \mathbb{N}$ do
2:     $P_i \leftarrow \emptyset$
3:     for $j \in \mathbb{N} \setminus \{i\}$ do
4:         Calculate node sequence $p_{i\to j}$ of shortest path
           from node $i$ to node $j$ using Dijkstra's algorithm
5:         $P_i \leftarrow P_i \cup \{p_{i\to j}\}$
6:     end for
7:     Merge duplicate sequences in $P_i$ to generate directed
       topology $T_i$
8: end for
9: return $\mathscr{T} = [T_i]_{i\in\mathbb{N}}$
```

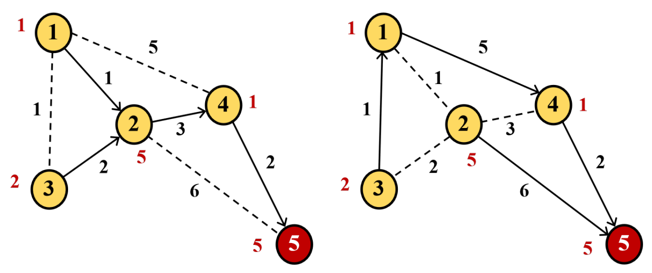
Algorithm 2 Optimal Aggregation Topology Selection Algorithm

```
Require: Candidate aggregation-tree set $\mathscr{T}$;
Node set $\mathbb{N}$ in $\mathscr{G}$;
Local computing time matrix $[t_n]_{n\in\mathbb{N}}$
Ensure: Optimal aggregation-tree topology $T_r$
1: for $n \in \mathbb{N}$ do
2:     Calculate synchronization latency $l_{T_n}$ for tree topol-
       ogy $T_n \in \mathscr{T}$ via Equation (13)
3:     Select topology with minimal latency, set root node:
       $r \leftarrow \arg\min_{n\in\mathbb{N}} l_{T_n}$
4: end for
5: return $T_r$
```



(a) Fastest topology (b) Random topology

Figure 6: Aggregation-tree topology optimized for transmission latency only.



(a) Fastest topology (b) Random topology

Figure 7: Aggregation-tree topology under computational blocking.

Iterate over each candidate topology in set $\mathscr{T}$ output by Algorithm 1 sequentially and compute the corresponding synchronization latency. The topology with the minimal latency is selected as the optimal aggregation-tree, as shown in Algorithm 2.

5.2 EDA-based Topology Optimization Algorithm

In this paper, we propose a combinatorial optimization algorithm for aggregation under complex and heterogeneous computing-network environments. Given that the global optimization objective is an implicit black box with respect to the decision variables, we employ a probability-model–driven metaheuristic—the Estimation of Distribution Algorithm (EDA)—to address the optimization problem formulated in Section 4. Furthermore, to avoid the limitation imposed by using a fixed number of aggregation-trees as a hard constraint on network strategy optimization, we design a lightweight two-stage coarse-to-fine search scheme to explore the suitable upper limit of the aggregation-tree number. The outer stage performs candidate selection and refinement over the upper-bound set $\mathscr{M}_{coarse} = \{1, \dots, m_{\max}\}$ to obtain a near-optimal upper bound m^*; the inner stage executes an EDA-based combinatorial strategy search with a fixed number limit of aggregation-trees. The detailed methodology is presented below.

1) Inner layer optimization: Unlike genetic algorithms, differential evolution applies difference-based mutation, crossover, and typically greedy survival selection at the individual level. EDA refines samples to estimate and update a probabilistic model, accumulating information across generations to guide the search. By tuning distributional parameters rather than solution parameters, EDA can more explicitly balance exploration and exploitation, thereby avoiding oscillation and premature convergence.

- Encoding: Given the number of all training nodes N and the upper bound on the number of aggregation-trees $K_{\max}$, the categorical matrix $\mathbf{O} = [o_{vk}] \in \mathbb{R}^{N \times K_{\max}}$ in which each gene $o_{v,k}$ represents the preference of node v to aggregation-tree k. We map $\mathbf{O}$ to a probability matrix $D = [\pi_v^{\mathrm{k}}] \in \mathbb{R}^{N \times K_{\max}}$ via the softmax transformation, where π_v^{k} denotes the discrete probability of matching node v with aggregation-tree k

$$\pi_v^{\mathrm{k}} = \frac{\exp(o_{vk})}{\sum_{k=1}^{K_{\max}} \exp(o_{vk})}. \tag{15}$$

Then, generate a one-hot selection vector $x_v = [x_{v,k}]_{k=1}^{K_{\max}}$ as a mapping strategy for each node by sampling under probability distribution $[\pi_v^{\mathrm{k}}]_{k \in K_{max}}$

$$x_{v,k} = \begin{cases} 1, & k = \arg\max_k \pi_v^{\mathrm{k}} \\ 0, & \text{otherwise} \end{cases}. \tag{16}$$

- Fitness evaluation: For a given strategy $\mathbf{X} = [x_v]_{v \in \mathbb{N}}$, partition nodes into subsets $\left[\mathbb{V}(k)\right]_{k \in K_{max}}$ and construct aggregation-trees according to Algorithm 1 and 2, and then compute the fitness $F(\mathbf{O})$, see Eq. (14).

- Population update: Let the current generation index be g. Rank population by fitness in descending order and select the top-τI individuals as the elite subset $S^{(g)}$, where I is the population size and $\tau \in (0,1]$ is the truncation ratio. The empirical mean and covariance of the elite set $S^{(g)}$ are estimated as

$$\mu^{(g)} = \frac{1}{\tau I} \sum_{x \in S^{(g)}} x, \tag{17}$$

$$\Sigma^{(g)} = \frac{1}{\tau I - 1} \sum_{x \in S^{(g)}} \left(x - \mu^{(g)}\right)\left(x - \mu^{(g)}\right)^{\top}. \tag{18}$$

To enhance numerical stability and maintain diversity, we add a diagonal jitter term $\eta \mathbf{I}$ ($\eta > 0$, $\mathbf{I}$ is the identity matrix), yielding

$$\tilde{\Sigma}^{(g)} = \Sigma^{(g)} + \eta \mathbf{I}. \tag{19}$$

Then, use the multivariate normal distribution $\mathscr{N}\left(\mu^{(g)}, \tilde{\Sigma}^{(g)}\right)$ as the generator for the next generation. Specifically, draw $\{x_i^{(g+1)}\}_{i=1}^{I}$ independently from $\mathscr{N}\left(\mu^{(g)}, \tilde{\Sigma}^{(g)}\right)$, evaluate their fitness, and replace the old population accordingly. To ensure elitism, the current best individual is directly copied to the next generation. Repeat this process until the strategy converges or reach the maximum number of generations.

2) Outer layer optimization: To adapt the scale of aggregation-trees to distributed training scenarios with different model sizes, we adopt a lightweight two-stage outer-layer model selection scheme to determine the number of global aggregation-trees.

In the coarse search stage, iterate over $K_{max} \in \mathscr{M}_{coarse}$ and independently run the inner-layer EDA with a small population size I_c and a small maximum number of generations G_c, recording the resulting objective $F(\mathbf{O} \mid K_{max})$ as the performance estimate $f^{coarse}(K_{max})$.

In the fine search stage, we select the top-t candidates by performance

$$\mathscr{M}_{fine} = \text{Top-}t\{[f^{coarse}(K_{max})]_{K_{max} \in \mathscr{M}_{coarse}}\}. \tag{20}$$

Conduct a deeper search on $\mathscr{M}_{fine}$ using a larger total number of generations I_f and a larger total generations G_f. We again record $F(\mathbf{X} \mid K'_{max})$ as the performance estimate $f^{fine}(K'_{max})$ for each K'_{max} in $\mathscr{M}_{fine}$.

The final upper bound on the number of aggregation-trees is set to

$$m^* = \arg\min_{K'_{max} \in \mathscr{M}_{\text{fine}}} f^{fine}(K'_{max}). \tag{21}$$

And the final mapping strategy is denoted as

$$\mathbf{X}^* = \arg\max_{\mathbf{X}} F(\mathbf{X} \mid m^*). \tag{22}$$

The overall algorithmic workflow of the EDA-based node mapping with coarse-to-fine search and the construction of aggregation topologies is shown in Algorithm 3.

Algorithm 3 EDA-based Topology Optimization Algorithm

Require: Upper-bound set $\mathscr{M}_{coarse}$; Truncation ratio τ; Hyperparameters (I_c, G_c) and (I_f, G_f); Top-t shortlist size t; The number of training nodes N

Ensure: Optimal node mapping strategy X^*

```
1:  for $K_{max} \in \mathscr{M}_{coarse}$ do
2:      Set $(I,G) \leftarrow (I_c, G_c)$
3:      Generate initial population $\mathscr{O}^{(0)} = \{O_i^{(0)}\}_{i=1}^{I}$
4:      for $g = 0$ to $G$ do
5:          for $i = 1$ to $I$ do
6:              Compute discrete probability matrix $D = [\pi_v^k] \in \mathbb{R}^{N \times K_{\max}}$
7:              $X^i \leftarrow [x_v]_{v \in \{1,\dots,N\}}$
8:              Partition nodes into subsets $[\mathbb{V}(k)]_{k \in K_{max}}$
9:              Construct aggregation-tree topology by Algorithm 2 for each $\mathbb{V}(k)$
10:             Evaluate fitness $F(O_i)$ based on Equation (14)
11:         end for
12:         Update population
13:     end for
14:     Record converged best fitness as performance estimate $f^{coarse}(K_{max})$
15: end for
16: Select $M_{fine}$
17: Set $(I,G) \leftarrow (I_f, G_f)$
18: for $K_{\max}^{'} \in \mathscr{M}_{fine}$ do
19:     Repeat the inner scanning (lines 4–15) with $(I,G)$ to obtain $f^{fine}(K_{\max}^{'})$
20: end for
21: $m^* \leftarrow \arg\min_{K_{\max}^{'} \in \mathscr{M}_{fine}} f^{fine}(K_{\max}^{'})$
22: Decode the best $X^*$ corresponding to $m^*$
23: return $X^*$
```

6 Performance Evaluation

To verify the effectiveness and scalability of the proposed heterogeneity-aware distributed training framework, we built a distributed platform that integrates the data resampling mechanism and the hybrid asynchronous aggregation forest. We then ran simulation experiments to evaluate the performance of the heuristic aggregation-topology optimization algorithm.

6.1 Experimental Setup

We deploy the distributed training platform on four physical hosts, whose hardware configurations are summarized in Table 1.

Table 1: Hardware configuration of experimental platform

Device Model	Component
LAPTOP-7GI3C08C	13th Gen Intel(R) Core(TM) i5-13420H
ROG Strix G512LV	Intel(R) Core(TM) i7-10875H NVIDIA TU106M (GPU)
Dell R740xd Server	Intel(R) Xeon(R) Gold 5218R NVIDIA L40 (GPU)
DESKTOP-RPI1KUU	13th Gen Intel(R) Core(TM) i7-13620H

On each host, we launch multiple training processes to emulate a total of 12 collaborative training nodes. An additional process plays the role of the global parameter server, which maintains the master copy of the model and performs global aggregation. Another control process acts as the master coordinator, issuing commands to training nodes, collecting runtime statistics, and orchestrating data exchange. To emulate heterogeneous WAN conditions, we inject link delays at the code level. The inter-node bandwidth is set between 1 Gbps and 10 Gbps, and the one-way link latency between nodes is randomized in the range of 10–50 ms.

We train a customized CNN-6 model on the CIFAR-10 dataset and adopt a Dirichlet-based partitioning strategy to simulate label-distribution skew across training nodes. For each class k in CIFAR-10 dataset, we sample a client allocation vector $p_k \sim Dirichlet(\alpha 1N)$, where $1N$ is an N-dimensional all-ones vector and α is the Dirichlet concentration parameter. Smaller α leads to more skewed class distributions, whereas larger α yields more balanced partitions.

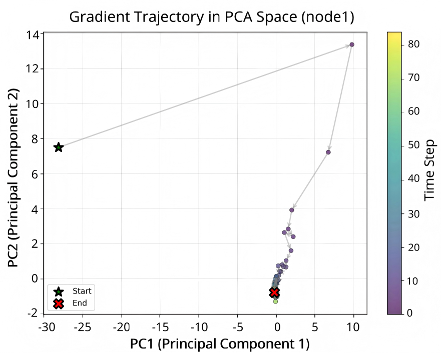
We compare the proposed hybrid asynchronous aggregation forest (HAAF) mechanism against three synchronization baselines: Parameter Server (PS): All workers send gradients to a central server, which aggregates them synchronously and broadcasts the updated model. AVL Tree (AVL): A balanced binary aggregation-tree in which each internal node has at most two children and the depths of leaf nodes differ by at most one. Fastest Aggregation Tree (FAT): A single aggregation-tree constructed via a shortest-path heuristic that minimizes transmission latency, but ignores heterogeneous local training times.

For all methods, the local mini-batch size at each node is set to 128 and SGD is used as the local optimizer with an initial learning rate of 0.01. The Lipschitz constant of the gradient is set to $L = 100$. By default, the Dirichlet concentration parameter is $\alpha = 1$, the resampling parameter is $\beta = 0.999$. In asynchronous updates, the staleness and fairness factors are $a = 0.8$ and $q = 1$, respectively. And the weight coefficients in the global optimization objective are set to $\lambda_1 = 0.5$ and $\lambda_2 = 0.1$.

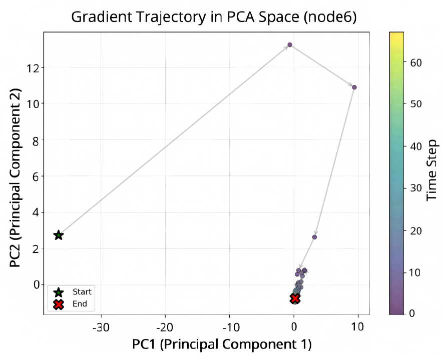
6.2 Simulation Results and Discussion

To intuitively illustrate the convergence behavior of each node under the HAAF architecture, we track the gradient trajectories of multiple nodes on two training tasks: (i) a CNN-6 model on CIFAR-10 and (ii) a LeNet-5 model on Fashion-MNIST. For each task, we apply PCA to project high-dimensional gradient onto two principal components and visualize the trajectories of three root nodes in the aggregation forest.

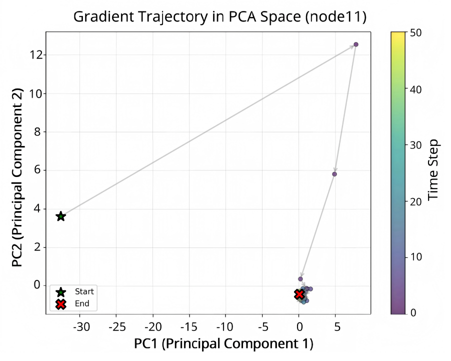
Under the given aggregation topology, the aggregation forest consists of three trees rooted at nodes 1, 6, and 11, whose synchronization periods increase in this order. As shown in Figure 8, the tree rooted at node 11 has the shortest aggregation period, so its local model quickly approaches a good local optimum in the early training stage. By contrast, the tree rooted at node 1 aggregates less frequently; its loss remains relatively high in later stages and occasionally



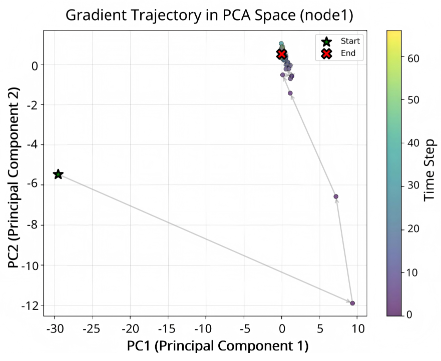
(a) Gradient trajectory of node 1 for CNN-6 on CIFAR-10



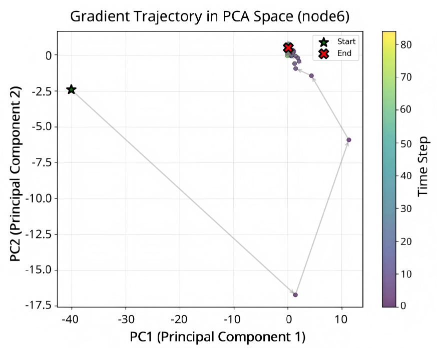
(b) Gradient trajectory of node 6 for CNN-6 on CIFAR-10



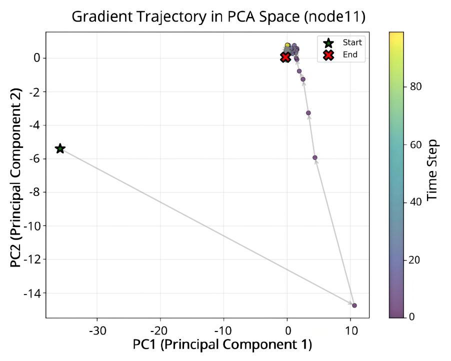
(c) Gradient trajectory of node 11 for CNN-6 on CIFAR-10



(d) Gradient trajectory of node 1 for LeNet-5 on Fashion-MNIST



(e) Gradient trajectory of node 6 for LeNet-5 on Fashion-MNIST



(f) Gradient trajectory of node 11 for LeNet-5 on Fashion-MNIST

Figure 8: Gradient trajectories of the three root nodes in PCA space for two representative model–dataset pairs. Panels (a)–(c) correspond to one pair and panels $(d)-(f)$ to another.

pulls the global model away from local minima, enabling further exploration of the loss landscape. Similar convergence patterns are observed across the two models, indicating that HAAF is robust to changes in model complexity and dataset characteristics.

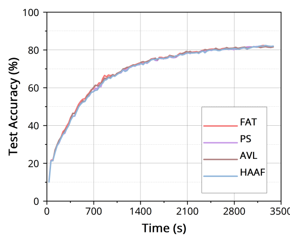
We next compare the convergence behaviors of four synchronization mechanisms under different levels of system heterogeneity. For each node, we measure the per-sample processing time and compute the Gini coefficient to quantify imbalance in computational capacity. A higher Gini coefficient signals a more severe imbalance, whereas a lower one reflects a more balanced and homogeneous system.

As shown in Figure 9, when Gini = 0.1 the system is nearly homogeneous and all mechanisms achieve similar convergence times. As heterogeneity increases, however, the straggler effect significantly slows down PS, AVL, and FAT, resulting in longer synchronization period and reduced overall throughput. In contrast, HAAF dynamically adapts the aggregation logic and effectively mitigates straggler-induced stalls. When Gini = 0.6, HAAF shortens convergence time by around 40% while maintaining comparable final accuracy; the slight accuracy loss is acceptable given the substantial improvement in convergence speed.

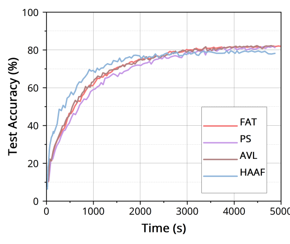
We then examine the sensitivity of the staleness factor a under two representative heterogeneity levels, Gini = 0.3 and 0.6, using fully synchronous PS and asynchronous PS as baselines. The results in Figure 10 show that fully synchronous PS achieves the highest final accuracy but converges the slowest due to strict synchronization. Asynchronous PS converges faster but suffers noticeable accuracy degradation, especially under strong heterogeneity. With the polynomial staleness weighting, HAAF substantially accelerates convergence while keeping the final accuracy close to that of fully synchronous PS. For $a \in (0, 1)$, the accuracy differences between different a values are small, and all HAAF configurations consistently outperform the asynchronous PS baseline.

To study the fairness–efficiency trade-off, we vary the fairness factor q and measure both the node-wise accuracy variance and the final test accuracy, as shown in Figure 11. When $q = 0$, the global model update does not account for performance fairness across aggregation-trees. In this case, the variance of per-node convergence accuracy increases sharply with the heterogeneity level, since high-performance trees dominate the global update direction. Introducing the fairness weight function ($q > 0$) significantly suppresses this variance and alleviates parameter drift. At the same time, the test accuracy of the global model is preserved or even slightly improved. Under the highly heterogeneous setting (Gini=0.6), increasing q from 0 to 5 reduces the accuracy variance from 19.2 to 11.81, while the final test accuracy improves from 69.5% to 77.07%. similar trends are observed for Gini=0.3, where the variance drops from 15.7 to 7.8 and the test accuracy can reach up to 79.5%. Moreover, the accuracies for

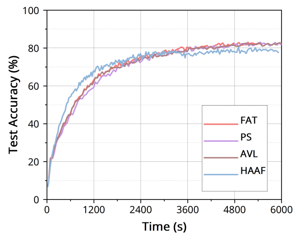
$q = 0.5$, $q = 1$, and $q = 5$ are very close in both settings, indicating that HAAF is insensitive to the exact choice of the fairness factor as long as it is set to a moderate value.



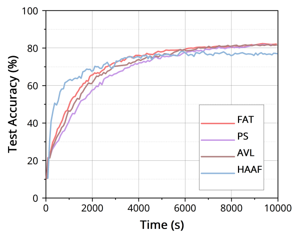
(a) Gini=0.1



(b) Gini=0.3

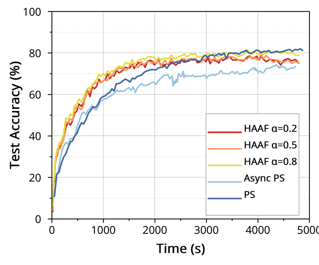


(c) Gini=0.5

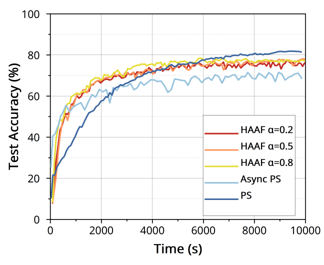


(d) Gini=0.6

Figure 9: Test accuracy over time under different synchronization mechanisms and system heterogeneity levels.

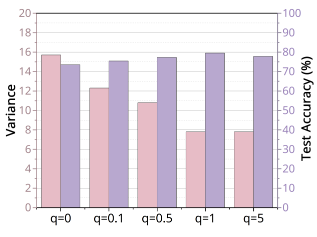


(a) Gini=0.3

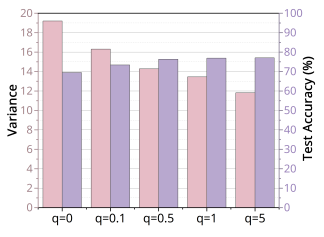


(b) Gini=0.6

Figure 10: Impact of staleness factor a on convergence under different heterogeneity levels.



(a) Gini=0.3



(b) Gini=0.6

Figure 11: Impact of the fairness factor q on the variance of local training progress and the final test accuracy under different system heterogeneity levels.

We further evaluate the proposed data β-resampling strategy under different degrees of label skew. By varying the Dirichlet concentration parameter α, we control the level of label-distribution heterogeneity, and by tuning β, we adjust the resampling strength and $\beta = 0$ corresponding to disabling resampling. The results in Figure 12 show that without resampling, the convergence accuracy degrades sharply as data non-IIDness increases. While β-resampling stabilizes convergence and significantly improves final accuracy by enforcing stronger gradient alignment across nodes. Compared with the no-resampling baseline, when $\alpha \leq 5$, the final test accuracy is improved by about 15%–39%. And Figure 13 visualizes the pairwise cosine similarity of initial gradients between nodes with and without resampling under the same Dirichlet setting, and confirms that gradients become much more aligned after applying the β-resampling mechanism.

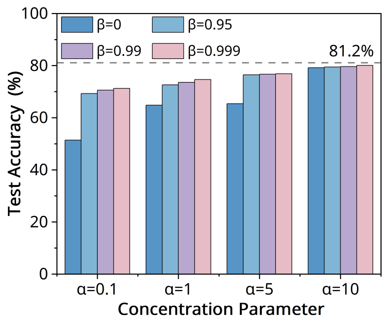
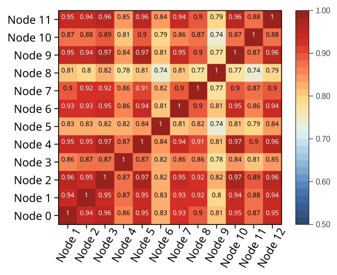
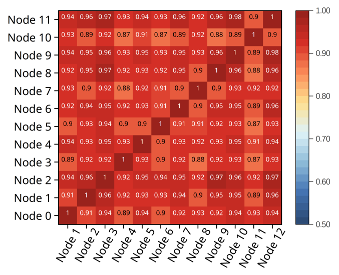
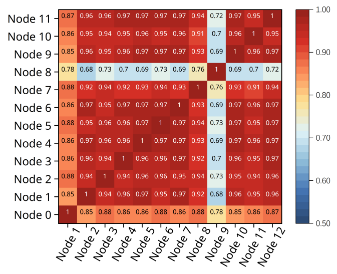


Figure 12: Test accuracy under varying β-resampling strengths and degrees of data heterogeneity, with the system heterogeneity fixed at a Gini coefficient of 0.1.


(a) $\alpha = 0.1, \beta = 0$


(b) $\alpha = 0.1, \beta = 0.999$


(c) $\alpha = 5, \beta = 0$

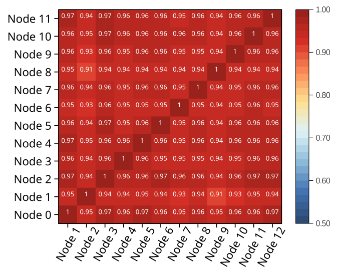

(d) $\alpha = 5, \beta = 0.999$

Figure 13: Similarity matrices of the learned aggregation topology under different hyperparameter settings of HAAF architecture.

To analyze how different synchronization mechanisms utilize heterogeneous resources, we measure the fraction of wall-clock time each node spends computation under each mechanisms. As illustrated in Figure 14, HAAF achieves both a lower mean and a lower variance of computation-time share across nodes. By shortening the idle waiting time of high-capacity nodes, HAAF improves the overall resource utilization of the system. In contrast, PS, AVL, and FAT exhibit pronounced straggler effects, which prolong synchronization cycles and reduce effective throughput.

Finally, we assess the scalability of the proposed framework by varying the number of training nodes. Figure 15 presents the convergence time of PS, AVL, FAT, and HAAF under different training scale. When the scale is small, nodes are not partitioned into multiple trees and all mechanisms perform similarly. As the system scales up, increasing heterogeneity and IO pressure substantially slow down the three baselines. By leveraging asynchronous-parallel updates across trees, HAAF achieves noticeably larger convergence-time reductions than FAT, AVL, and PS across all evaluated scales.

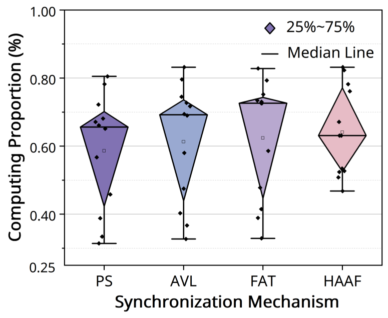


Figure 14: Distribution of per-node computation-time share under different synchronization mechanisms.

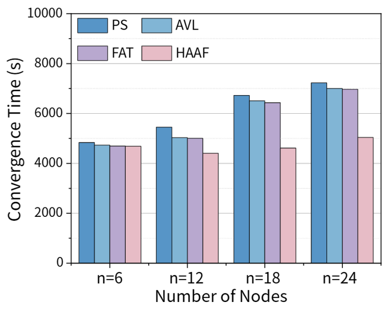


Figure 15: Convergence time versus system scale under different synchronization mechanisms.

7 Conclusion

In this paper, we propose a heterogeneity-aware framework for distributed training in CPNs. The framework employs a hybrid asynchronous aggregation forest mechanism, combined with a heuristic synchronization-topology optimization algorithm and a sample resampling strategy, to alleviate the adverse impact of system heterogeneity and data non-IIDness on model convergence. Experimental results show that, under various heterogeneous configurations, the proposed framework achieves substantial improvements in both training efficiency and convergence accuracy compared with mainstream synchronous communication mechanisms.

Funding

This work was supported by the National Natural Science Foundation of China under Grant 92367104. This research was financially supported by the Open Research Fund from Guangdong Laboratory of Artificial Intelligence and Digital Economy (SZ), under Grant No. GML-KF-24-25.

Author Contributions

Conceptualization, Zhe Deng and Renchao Xie; methodology, Zhe Deng; software, Zhe Deng and Qinqin Tang; validation, Zhe Deng and Li Feng; formal analysis, Zhe Deng; investigation, Zhe Deng; resources, Renchao Xie; data curation, Qinqin Tang; writing—original draft preparation, Zhe Deng; writing—review and editing, Qinqin Tang and Li Feng; visualization, Zeru Fang and Tao Huang; supervision, Renchao Xie; project administration, Hongliang Lu. All authors have read and agreed to the published version of the manuscript.

Conflict of Interest

All the authors declare that they have no conflict of interest.

References

[1] Wang, S., Zheng, H., Wen, X., Shang, F.: Distributed high-performance computing methods for accelerating deep learning training. Journal of Knowledge Learning and Science Technology, **3**(3), 108–126 (2024). https://doi.org/10.60087/jklst.v3.n3.p108-126

[2] Sun, Y., Liu, J., Huang, H.-Y., Zhang, X., Lei, B., Peng, J., Wang, W.: Computing power network: A survey. China Communications, **21**(9), 109–145 (2024). https://doi.org/10.23919/JCC.ja.2021-0776

[3] Pei, J., Liu, W., Li, J., Wang, L., Liu, C.: A review of federated learning methods in heterogeneous scenarios. IEEE Transactions on Consumer Electronics, **70**(3), 5983–5999 (2024). https://doi.org/10.1109/TCE.2024.3385440

[4] Oughton, E.J., Lehr, W., Katsaros, K., Selinis, I., Bubley, D., Kusuma, J.: Revisiting wireless internet connectivity: 5G vs Wi-Fi 6. Telecommunications Policy, **45**(5), 102127 (2021). https://doi.org/10.1016/j.telpol.2021.102127

[5] Gangidi, A., Miao, R., Zheng, S., Bondu, S.J., Goes, G., Morsy, H., Puri, R., Riftadi, M., Shetty, A.J., Yang, J., Shuqiang Zhang, S., et al.: RDMA over Ethernet for distributed training at Meta scale. In Proceedings of the ACM SIGCOMM 2024 Conference, pp. 57–70 (2024). https://doi.org/10.1145/3651890.3672233

[6] Zhang, Z., Cai, D., Zhang, Y., Xu, M., Wang, S., Zhou, A.: FedRDMA: Communication-efficient cross-silo federated LLM via chunked RDMA transmission. In Proceedings of the 4th Workshop on Machine Learning and Systems, pp. 126–133 (2024). https://doi.org/10.1145/3642970.3655834

[7] Alhafiz, F., Basuhail, A.: The data heterogeneity issue regarding COVID-19 lung imaging in federated learning: an experimental study. Big Data and Cognitive Computing, **9**(1), 11 (2025). https://doi.org/10.3390/bdcc9010011

[8] Haddadpour, F., Kamani, M.M., Mahdavi, M., Cadambe, V.R.: Local SGD with periodic averaging:

Tighter analysis and adaptive synchronization. In Proceedings of the 33rd International Conference on Neural Information Processing Systems, pp. 11082–11094 (2019). https://doi.org/10.1145/3472456.3472508

[9] Tu, J., Yang, L., Cao, J.: Distributed Machine Learning in Edge Computing: Challenges, Solutions and Future Directions. ACM Computing Surveys, **57**(5), 1–37 (2025). https://doi.org/10.1145/3708495

[10] Han, L., Huang, X., Li, D., Zhang, Y.: RingFFL: A ring-architecture-based fair federated learning framework. Future Internet, **15**(2), 68 (2023). https://doi.org/10.3390/fi15020068

[11] Cho, M., Finkler, U., Kung, D.: BlueConnect: Novel hierarchical all-reduce on multi-tiered network for deep learning. In 32nd Conference on Neural Information Processing Systems (NeurIPS 2018), PP. 1–8 (2019).

[12] Chai, Z., Ali, A., Zawad, S., Truex, S., Anwar, A., Baracaldo, N., Zhou, Y., Ludwig, H., Yan, F., Cheng, Y.: TiFL: A tier-based federated learning system. In Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing (HPDC), pp. 125–136 (2020). https://doi.org/10.1145/3369583.3392686

[13] Yuan, J., Xu, M., Ma, X., Zhou, A., Liu, X., Wang, S.: Hierarchical federated learning through LAN-WAN orchestration. arXiv preprint arXiv:2010.11612 (2020). https://doi.org/10.48550/arXiv.2010.11612

[14] Reisizadeh, A., Mokhtari, A., Hassani, H., Jadbabaie, A., Pedarsani, R.: FedPAQ: A communication-efficient federated learning method with periodic averaging and quantization. In International Conference on Artificial Intelligence and Statistics (AISTATS). Proceedings of Machine Learning Research (PMLR), pp. 2021–2031 (2020).

[15] Hosseinalipour, S., Azam, S.S., Brinton, C.G., Michelusi, N., Aggarwal, V., Love, D.J.: Multi-Stage Hybrid Federated Learning Over Large-Scale D2D-Enabled Fog Networks. IEEE/ACM Transactions on Networking, **30**(4), 1569–1584 (2022). https://doi.org/10.1109/TNET.2022.3143495

[16] Mayer, R., Jacobsen, H.A.: Scalable deep learning on distributed infrastructures: Challenges, techniques, and tools. ACM Computing Surveys (CSUR), **53**(1), 1–37 (2020). https://doi.org/10.1145/3363554

[17] Ouyang, S., Dong, D., Xu, Y., Xiao, L.: Communication optimization strategies for distributed deep neural network training: A survey. Journal of Parallel and Distributed Computing, **149**, 52–65 (2021). https://doi.org/10.1016/j.jpdc.2020.11.005

[18] Dutta, A., Bergou, E.H., Abdelmoniem, A.M., Ho, C.-Y., Sahu, A., Canini, M., Kalnis, P.: On the discrepancy between the theoretical analysis and practical implementations of compressed communication for distributed deep learning. In Proceedings of the AAAI Conference on Artificial Intelligence, pp. 3817–3824 (2020). https://doi.org/10.1609/aaai.v34i04.5793

[19] Duan, S., Wang, D., Ren, J., Lyn, F., Zhang, Y., Wu, H.: Distributed artificial intelligence empowered by end-edge-cloud computing: A survey. IEEE Communications Surveys & Tutorials, **25**(1), 591–624 (2022). https://doi.org/10.1109/COMST.2022.3218527

[20] Chen, Z., Liu, X., Li, M., Hu, Y., Mei, H., Xing, H.: RINA: Enhancing ring-AllReduce with in-network aggregation in distributed model training. In 2024 IEEE 32nd International Conference on Network Protocols (ICNP), pp. 1–12. IEEE (2024). https://doi.org/10.1109/ICNP61940.2024.10858570

[21] Shoeybi, M., Patwary, M., Puri, R., LeGresley, P., Casper, J., Catanzaro, B.: Megatron-LM: Training multi-billion parameter language models using model parallelism. arXiv preprint arXiv:1909.08053 (2019). https://doi.org/10.48550/arXiv.1909.08053

[22] Hashemi, S.H., Abdu Jyothi, S., Campbell, R.H.: TicTac: Accelerating distributed deep learning with communication scheduling. In Proceedings of the 2nd Machine Learning and Systems (MLSys) Conference, pp. 418–430 (2019).

[23] Jayarajan, A., Wei, J., Gibson, G., Fedorova, A., Pekhimenko, G.: Priority-based parameter propagation for distributed DNN training. In Proceedings of the 2nd Machine Learning and Systems (MLSys) Conference, pp. 132–145 (2019).

[24] Yan, G., Li, T., Huang, S.-L., Lan, T., Song, L.: AC-SGD: Adaptively Compressed SGD for Communication-Efficient Distributed Learning. IEEE Journal on Selected Areas in Communications, **40**(9), 2678–2693 (2022). https://doi.org/10.1109/JSAC.2022.3192050

[25] Wang, S., Li, D., Geng, J., Gu, Y., Cheng, Y.: Impact of network topology on the performance of DML: Theoretical analysis and practical factors. In IEEE INFOCOM 2019-IEEE Conference on Computer Communications, pp. 1729–1737 (2019). https://doi.org/10.1109/INFOCOM.2019.8737595

[26] Liu, D., Zhu, G., Zhang, J., Huang, K.: Data-importance aware user scheduling for communication-efficient edge machine learning. IEEE Transactions on Cognitive Communications and Networking, **7**(1), 265–278 (2020). https://doi.org/10.1109/TCCN.2020.2999606

[27] Wang, Z., Duan, Q., Xu, Y., Zhang, L.: An efficient bandwidth-adaptive gradient compression algorithm for

distributed training of deep neural networks. Journal of Systems Architecture, **150**, 103116 (2024). https://doi.org/10.1016/j.sysarc.2024.103116

[28] Ruan, M., Yan, G., Xiao, Y., Song, L., Xu, W.: Adaptive top-K in SGD for communication-efficient distributed learning. In GLOBECOM 2023—2023 IEEE Global Communications Conference, pp. 5280–5285 (2023). https://doi.org/10.1109/GLOBECOM54140.2023.10437811

[29] Liao, Y., Xu, Y., Xu, H., Yao, Z., Wang, L., Qiao, C.: Accelerating federated learning with data and model parallelism in edge computing. IEEE/ACM Transactions on Networking, **32**(1), 904–918 (2024). https://doi.org/10.1109/TNET.2023.3299851

[30] Ling, Z., Jiang, X., Tan, X., He, H., Zhu, S., Yang, J.: Joint dynamic data and model parallelism for distributed training of DNNs over heterogeneous infrastructure. IEEE Transactions on Parallel and Distributed Systems, **36**(2), 150–167 (2024). https://doi.org/10.1109/TPDS.2024.3506588

[31] Li, T., Sahu, A. K., Zaheer, M., Sanjabi, M., Talwalkar, A., Smith, V.: Federated optimization in heterogeneous networks. In Proceedings of the 3rd Machine Learning and Systems Conference, pp. 429–450 (2020).

[32] Tang, Z., Hu, Z., Shi, S., Cheung, Y.-M., Jin, Y., Ren, Z., Chu, X.: Data resampling for federated learning with non-IID labels. In Proceedings of the International Workshop on Federated and Transfer Learning for Data Sparsity and Confidentiality (in conjunction with IJCAI), (2021).

[33] Li, T., Sanjabi, M., Beirami, A., Smith, V.: Fair resource allocation in federated learning. arXiv preprint arXiv:1905.10497 (2019). https://doi.org/10.48550/arXiv.1905.10497

[34] Sun, S., Zhang, Z., Pan, Q., Liu, M., Wang, Y., He, T.: Staleness-controlled asynchronous federated learning: Accuracy and efficiency tradeoff. IEEE Transactions on Mobile Computing, **23**(12), 12621–12634 (2024). https://doi.org/10.1109/TMC.2024.3416216

[35] Xie, C., Koyejo, S., Gupta, I.: Asynchronous federated optimization. arXiv preprint arXiv:1903.03934 (2019). https://doi.org/10.48550/arXiv.1903.03934

[36] Awan, A.A., Hamidouche, K., Venkatesh, A., Panda, D.K.: Efficient large message broadcast using NCCL and CUDA-aware MPI for deep learning. In Proceedings of the 23rd European MPI Users' Group Meeting (EuroMPI '16), pp. 15–22 (2016). https://doi.org/10.1145/2966884.2966912