Accepted to appear in IEEE Transactions on Services Computing, 2024

Efficient Parameter Synchronization for Peer-to-Peer Distributed Learning With Selective Multicast

Shouxi Luo, Pingzhi Fan, Ke Li, Huanlai Xing, Long Luo, and Hongfang Yu

***Abstract*—Recent advances in distributed machine learning show theoretically and empirically that, for many models, provided that workers will eventually participate in the synchronizations, *i*) the training still converges, even if only *p* workers take part in each round of synchronization, and *ii*) a larger *p* generally leads to a faster rate of convergence. These findings shed light on eliminating the bottleneck effects of parameter synchronization in large-scale data-parallel distributed training and have motivated several optimization designs. In this paper, we focus on optimizing the parameter synchronization for *peer-to-peer* distributed learning, where workers broadcast or multicast their updated parameters to others for synchronization, and propose SELMCAST, a suite of expressive and efficient multicast receiver selection algorithms, to achieve the goal. Compared with the state-of-the-art (SOTA) design, which randomly selects exactly *p* receivers for each worker's multicast in a bandwidth-agnostic way, SELMCAST chooses receivers based on the global view of their available bandwidth and loads, yielding two advantages, i.e., accelerated parameter synchronization for higher utilization of computing resources and enlarged average *p* values for faster convergence. Comprehensive evaluations show that SELMCAST is efficient for both peer-to-peer Bulk Synchronous Parallel (BSP) and Stale Synchronous Parallel (SSP) distributed training, outperforming the SOTA solution significantly.**

***Index Terms*—Distributed learning, receiver selection, parameter synchronization**

I. INTRODUCTION

Over the past decade, machine learning techniques obtain tremendous success and have been widely employed for various applications like *email filtering*, *advertising recommendation*, *speech recognition*, *machine translation*, *computer vision*, etc [1]–[5]. With the increasing popularity of machine learning and the rapid development of new technologies, the realistic quantities of training data for a learning task have increased from GBs to TBs and PBs. Data-parallel distributed training has become the key to obtaining the resulting model over such massive amounts of data within reasonable times [2]–[4].

In datacenter-based data-parallel distributed training, the dataset is generally split and then distributed among a group of high-performance servers (i.e., workers), each of which holds a replica of the model and iteratively updates its model values with local training. To guarantee convergence, these workers will synchronize their new resulting models periodically, via various communication topologies such as the *star* (i.e., parameter server), *tree*, *ring*, and *peer-to-peer* [3], [6]. These schemes have various properties regarding *latency*, *traffic overheads*, and *reliability*, thus being employed by various learning systems and training algorithms.

With the continued growth of the training scale, the volume of traffic triggered by parameter synchronization is increasing greatly. Moreover, the wide employment of new hardware like GPU, FPGA, and TPU, has repeatedly accelerated the computation a lot, while the upgrade of network infrastructure is relatively complicated and slow [7]. As a result, the non-trivial time it takes for the underlying network to complete the parameter synchronization would dominate the time cost of the entire training, becoming the bottleneck. Optimizing the communication bottleneck involved in parameter synchronization is crucial for the implementation and deployment of large-scale distributed machine learning. Indeed, this is a hot research topic, where a large number of works are involved [2], [3], [6], [8]–[10]. Recent advances have shown theoretically and empirically that, for many models, provided workers would participate in the synchronization eventually, *i*) the training will still converge, even if there are only *p* workers taking part in each round of synchronization, and *ii*) a larger value of *p* on average generally leads to a smaller round of training to converge [4], [9], [11]–[15]. These findings shed light on the optimization of parameter synchronization, having motivated the improvements of several learning algorithms and systems [4], [11]–[15].

In this paper, we focus on accelerating the parameter synchronization for *peer-to-peer* distributed learning, following the works of Orpheus [4], Malt [15], and SFB [14], and propose SELMCAST, an expressive and efficient multicast receiver selection algorithm to achieve the goal. More specifically, in a naive *peer-to-peer* distributed training, to drive a round of synchronization, each worker would broadcast its new local model, the model update, or the sufficient factor of the model update to all other workers [4], [14]. As a result, the synchronization of *n* workers would yield $O(n^2)$

This work was supported in part by NSFC under Grant 62002300, Grant 62020106001, and Grant 62394324, in part by the Fundamental Research Funds for the Central Universities under Grant 2682024ZTPY050, and Grant 2682022ZTPY089, and in part by the Sichuan Science and Technology Program under Grant 2023ZHJY0009. The preliminary version of this work entitled "Fast Parameter Synchronization for Distributed Learning with Selective Multicast" was published in IEEE International Conference on Communications (ICC) 2022 [1] [DOI: 10.1109/ICC45855.2022.9838266]. *(Corresponding author: Shouxi Luo.)*

Shouxi Luo is with the School of Computing and Artificial Intelligence, Southwest Jiaotong University, Chengdu 611756, China (e-mail: sxluo@swjtu.edu.cn).

Pingzhi Fan is with the Key Laboratory of Information Coding and Transmission, CSNMT Int Coop. Res. Centre, Southwest Jiaotong University, Chengdu, 611756, China (e-mail: p.fan@ieee.org).

Ke Li and Huanlai Xing are with the School of Computing and Artificial Intelligence, Southwest Jiaotong University, Chengdu 611756, China (e-mail: keli@swjtu.edu.cn; hxx@swjtu.edu.cn).

Long Luo and Hongfang Yu are with the School of Information and Communication Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China (e-mail: llong@uestc.edu.cn; yuhf@uestc.edu.cn).

traffic volumes. For this issue, Orpheus [4], the state-of-the-art proposal, allows each worker only deliver model updates to p (out of $n-1$) randomly selected receivers in a bandwidth-agnostic way. Such a design reduces the total traffic volume and guarantees eventual convergence. However, it still suffers from two serious problems. Firstly, since the completion of synchronization is dominated by the slowest receiver, this bandwidth-agnostic random selection might not remove the bottleneck, thus bringing no improvement to the synchronization with a high probability. Secondly, a larger proportion of receivers involved in the synchronization generally leads to fewer rounds of training to converge [9], [11], [12]. Orpheus misses this opportunity of optimization since it only chooses p receivers even if more receivers have sufficient bandwidth.

As a comparison, SELMCAST picks receivers based on the global view of their available bandwidth and loads heuristically. Despite the fact that the original selection problem is hard to solve in theory, powered by insights stemming from a model-based analysis of the problem, SELMCAST is excellent in achieving efficient selections for both Bulk Synchronous Parallel (BSP) and Stale Synchronous Parallel (SSP) distributed training [4]. Extensive evaluations imply that SELMCAST makes near-optimal selections, outperforming Orpheus significantly, in terms of both the completion time of synchronization or the utilization of computation resources, and the average number of selected receivers.

Briefly, the contributions of this paper are four-fold:

1) A thorough analysis of the drawbacks of bandwidth-agnostic random multicast receiver selection employed by state-of-the-art solution.
2) A multiple-objective (mixed) integer linear programming that describes the optimal receiver selection problem and motivates our algorithm design.
3) SELMCAST, a suite of $O(n^3)$ bandwidth-aware receiver selection algorithms constructing efficient multicast topologies for the parameter synchronization of both BSP and SSP peer-to-peer distributed training.
4) Extensive evaluations, showing that SELMCAST is effective, efficient, and *near-optimal* to accelerate data-parallel peer-to-peer distributed learning built upon both BSP and SSP.

Design Space and Limitation of SELMCAST. Theoretically, a larger multicast scale (saying p for instance) generally brings benefits to the converging speed (in terms of the number of training rounds to converge) of peer-to-peer distributed training, at the possible cost of slower completion of a round of distributed training as more traffic volume would be introduced for parameter synchronization [1], [11]. Note that, the wall time of the training is the product of the number of involved training rounds and the average time cost of each round. Thus, for a given distributed training task, there is a sweet spot for the choice of p. However, in practice, the convergence behavior of distributed training can be affected by a lot of factors like the structure of the model, the quality of the training dataset, and the settings of various hyper-parameters including the batch size, and learning rate [16], [17]; as far as we know, there does not exist a clear model to describe the impacts of p on the converging speed thus the sweet spot of p is case-by-case and hard to track [11], [16]. Based on these facts, SELMCAST aims to provide a generic multicast optimization service for the parameter synchronization acceleration of various peer-to-peer distributed model training tasks. At the core, SELMCAST achieves this goal using best-effort optimization designs—It tries to simultaneously *i*) accelerate the completion of each synchronization thus improving the utilization of computing resources, and *ii*) enlarge the average scale of multicast.

In the rest of this paper, we first overview the background and motivation in §II, then formulate the problem, analyze it, and propose SELMCAST as a solution in §III. After that, performance evaluation and related work discussion follow in §IV and §V, respectively. Finally, §VI concludes the paper.

II. BACKGROUND AND MOTIVATION

In this section, we first briefly overview the background of distributed data-parallel training (§II-A), the synchronization schemes of BSP and SSP (§II-B), and analyze the drawbacks of state-of-the-art solutions for receiver selection (§II-C).

A. Distributed Data-Parallel Training

Nowadays, distributed data-parallel (DDP) training is widely employed by machine learning algorithms to train models over massive amounts of data within reasonable times. In DDP training, the dataset is split across a group of workers, who then train their local replicas of the model iteratively in parallel. To ensure convergence, workers synchronize their updated local models periodically. Regarding the implementation of parameter synchronization, there are four types of basic communication topology designs widely used today, namely, *stars (i.e., parameter server)*, *trees*, *rings*, and *peer-to-peer*, respectively [3].

These designs have different properties in terms of *latency*, *traffic overhead*, *reliability*, etc., targeting various application scenarios. Among them, the *peer-to-peer* architecture is fully decentralized, allowing workers to communicate with each other directly, thus eliminating the single point of failure and bottleneck. Currently, peer-to-peer parameter synchronization is supported by many distributed machine learning frameworks such as TensorFlow, PyTorch, MXNet, MALT, and Orpheus, and used by numerous training algorithms [4], [14], [15]. In this paper, we focus on optimizing model (parameter) synchronization for peer-to-peer distributed learning.

B. BSP and SSP

As for the consistency model of workers, current peer-to-peer training frameworks like Orpheus [4] generally support two types of convergence-guaranteed designs, namely BSP and SSP [4], [18], [19], respectively. When BSP is employed, all workers start the new round of training at the same time, then synchronize their locally updated model parameters once all workers complete the current round of training, and move to the next round after the synchronization at the same time again. Various recent studies have shown that the time it takes to complete a round of training for workers is generally

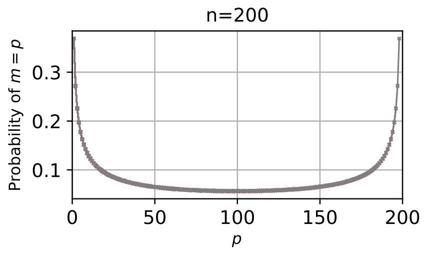


Fig. 1: The probability that a given worker happens to obtain the balanced load (i.e., $m = p$) is small; it decreases drastically then increases slowly, with the growth of p.

skewed due to various reasons [20]. As a result, in BSP, workers who complete early have to be idle to wait for these stragglers, leading to a waste of time and low utilization of computing resources. To address the impact of straggler workers, the design of SSP is proposed [18], [19], as a relaxed-yet-generalized version of BSP. Specifically, in SSP, a worker is allowed to conduct synchronization and move to the next round of training, once its training round is not larger than the slowest worker too much; i.e., the gap is within a given threshold like ssp. In practice, the value of ssp can be either pre-defined or dynamically tuned [21]. Following this, BSP can be treated as the specific case with $ssp = 0$.

C. Drawback of State-of-the-art Solutions

In peer-to-peer parameter synchronization, the communication overhead grows quadratically with the number of training workers, as each worker would broadcast its updated model values to all other nodes [3], [15]. Motivated by the fact that the training of many models is able to tolerate some levels of partial and staleness parameter synchronization, proposals like Orpheus [4], Malt [15], and partial SFB [14] let each worker send its new model to only a subset of the receivers, making the traffic overheads controllable. This partial broadcast (a.k.a., multicast) does reduce the amount of traffic; however, the completion time of synchronization might not change, because the selection of receivers does not take the available bandwidth of each worker into account. Take the design adopted by the state-of-the-art Orpheus [4] as an example. Suppose that there are n workers training a model with data parallelism. At each round of synchronization, every worker in Orpheus would deliver its model to other p ($1 \leq p \leq n-1$) randomly selected receivers, unaware of their available link capacities.

In theory, given a worker, the probability that it is selected as the receivers of m transfers can be calculated by Equation (1).

$$Pr(m) = C_{n-1}^{m} \left(\frac{p}{n-1}\right)^{m} \left(1 - \frac{p}{n-1}\right)^{n-1-m} \tag{1}$$

If the randomization of receivers is conducted perfectly, each worker will be selected as the receiver of p multicast transfers on average. Numerically, the probability that a given worker happens to be involved in other p transfers is small; and the value first decreases drastically and then increases slowly, with the growth of p. As the instance of $n = 200$ in Figure 1 shows, when $p = 1$, $Pr(m = p) = Pr(1) \approx 0.37$; and once p is in the range of [18, 181], the probability of achieving the balanced load for a given worker is less than 0.1. These results imply that even if all workers have the same available link capacity, random receiver selection would lead to a serious load imbalance. Even worse, in production, there might be other applications hosted in the same cluster thus the available link capacities that each worker could use are highly skewed. This mismatch between the selection of receivers and the available bandwidth makes the time costs of parameter synchronization unoptimized with a very high probability.

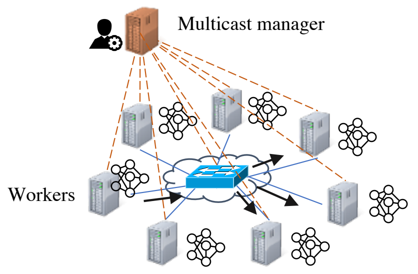


Fig. 2: An overview of SELMCAST-optimized systems.

Another drawback of random selection is that it selects only p receivers even if there are abundant workers with sufficient bandwidth. Recent studies [9], [11] show that the increase in the proportion of workers generally yields a faster convergence of the training. Thus, there is room for improvement.

III. SELECTIVE MULTICAST

As Figure 2 sketches, SELMCAST is a set of multicast receiver selection algorithms running at a centralized manager that help distributed training workers determine the communication topology for the multicast of their parameters. It can be integrated into existing communication libraries with engineering efforts and employed to enhance the performance of distributed training frameworks like Orpheus [4], Malt [15], and SFB [14], and other emerging peer-to-peer distributed training systems [22]. Despite the implementation of SELMCAST relying on a logically centralized manager, it can handle large-scale training clusters since the job of the SELMCAST manager is minimal, i.e., determining the receivers for multicast requests based on the observed network states. In practice, the involved multicast might be implemented at either Layer 3 (i.e., L3 for short) or Layer 7 (i.e., L7 for short) by using existing transport protocols [2], [23]. Accordingly, SELMCAST should be expressive to support both L3 and L7 multicast implementations at the same time.

During training, the SELMCAST manager collects the available bandwidth and remaining multicast task of each worker, based on which, it computes efficient multicast topologies incrementally, for each worker that becomes ready to synchronization, to control their multicast transfers. Once selected by SELMCAST, workers will launch either real multicast (L3) or unicast (L7) transfers to conduct the delivery, according to the underlying network [23], [24]. As being load-agnostic, the random multicast adopted by [4] is likely to run into load imbalance with high probability. Differently, the core idea of SELMCAST is to choose receivers for newly active parameter multicast transfers in a bandwidth-aware way, such that it would take less time for the entire synchronization to complete. We not only control the minimum number of receivers for each multicast, but also ensure that each worker always multicasts its parameters to all other workers at least one time in every $k + 1$ rounds of its synchronization, such that eventual (global) synchronization is ensured. Moreover, to accelerate the convergence, we try to let more workers participate in the multicast, in case their joins would not postpone the completion of the entire multicast too much. Given that *asynchronous parallel* (ASP) can be treated as the specific case of SSP (i.e., $ssp = +\infty$), SELMCAST can be extended to support ASP-based peer-to-peer distributed training and other advanced designs relying on mixed designs [25].

In the rest of this section, we first analyze the optimal receiver selection problem formally (§III-A), among which, the used notations are summarized in Table I; then based on the findings, we further design receiver algorithms for peer-to-peer BSP (§III-B) and SSP (§III-C) training in detail, respectively. As we will show, both algorithms could be implemented as $O(n^3)$, making SELMCAST very efficient.

A. Problem Analysis

Now, we formally analyze the selection problem for BSP training in theory, which provides insights on designing heuristic algorithms for both BSP (§III-B) and SSP (§III-C) training.

1) Math Formulation: Given the non-blocking design of modern data center networks [23], [26], [27], we abstract the entire data center network out as one big switch as Figure 2 shows, in which congestions only occur at ingresses and egresses. Without loss of generality, we assume that the training task involves n workers (denoted by N), and denote the remaining bandwidth of the abstract switch on ingress i and egress j that the distributed training could use as b_i^I and b_j^E, respectively. To perform parameter synchronization, the i-th sender would deliver its newest model to at least p_i other workers. Let the binary variable of $x_{i,j}$ denote whether the j-th worker is selected as the receiver of the multicast task rooted at worker i in this round of synchronization, or not; then, we have constraints (2) and (3). Note that there is no need to explicitly transmit the results of worker i to itself via the network. Thus, we define that $x_{i,i}$ is always 0 for $\forall i \in N$, i.e., (4). For ease of representation, we further let X_i be $\{x_{i,j} : j \in N\}$ and X be $\cup_{i \in N} X_i$, respectively.

$$x_{i,j} \in \{0, 1\}, \qquad \forall (i, j) \in N \times N \tag{2}$$

$$\sum_{j \in N} x_{i,j} \geq p_i, \qquad \forall i \in N \tag{3}$$

$$x_{i,i} = 0, \qquad \forall i \in N \tag{4}$$

Moreover, to guarantee convergence, each worker should synchronize with all other workers at least once every $k + 1$ round. Let D be the set of worker pairs out-of-model delivery in the last k rounds. For $(i, j) \in D$, the corresponding $x_{i,j}$ would be enforced to 1 as Eq.(5) shows.

$$x_{i,j} = 1, \quad (i, j) \in D \tag{5}$$

The delivery of each worker's model is a typical one-to-many transfer that can be carried out with techniques like L3 IP multicast [23]. If the underlying network does not support IP multicast, workers could alternatively launch a group of concurrent unicast transfers for all sender-receiver pairs to achieve the multicast at the application layer (i.e., L7). Note that, like the case of coflow [26], [28], a parameter synchronization is treated as done if and only if all the selected deliveries have finished. Let v_i be the original volume for the multicast task rooted at worker i, $v_{i,j}$ be the remaining volume that worker i has not delivered to worker j, and t be the time that all workers need to complete this amount of data. Then, in the case of BSP training, just enforcing all involved transfers to finish at the same time (i.e., t) would not harm the completion of the entire synchronization. To avoid congestion, we would have constraints (6) and (7) for involved $x_{i,j}$s and $v_{i,j}$s. Here, s_i is either $\max_{j \in N} v_{i,j} x_{i,j}$ or $\sum_{j \in N} v_{i,j} x_{i,j}$ ($\geq p_i \geq 1$), as (9) denotes, respecting whether L3 multicast is employed or not.

$$\frac{s_i}{t} \leq b_i^I, \qquad \forall i \in N \tag{6}$$

$$\sum_{i \in N} \frac{v_{i,j} x_{i,j}}{t} \leq b_j^E, \qquad \forall j \in N \tag{7}$$

$$t > 0 \tag{8}$$

$$s_i = f(X_i, V_i) = \begin{cases} \max_{j \in N} v_{i,j} x_{i,j} & \text{for L3 multicast} \\ \sum_{j \in N} v_{i,j} x_{i,j} & \text{for L7 multicast} \end{cases} \tag{9}$$

Obviously, inequalities (6) and (7) can be rewritten as the linear inequalities (10) and (11), respectively.

$$t \geq \frac{1}{b_i^I} s_i, \qquad \forall i \in N \tag{10}$$

$$t \geq \frac{1}{b_j^E} \sum_{i \in N} v_{i,j} x_{i,j}, \qquad \forall j \in N \tag{11}$$

Then, the optimization receiver selection problem for BSP training can be formulated as a multi-objective mixed-integer linear programming (MILP) as (12) shows.

$$Minimize \left\{ \left(t, -\sum_{x_{i,j} \in X} x_{i,j} \right) : (2)-(5), (8)-(11) \right\} \tag{12}$$

2) Hardness: As the reduction of the completion is more important than the enlargement of the multicast scale, the multi-objective mixed integer programming specified by (12) can be solved with *lexicographic optimization*: i.e., 1) first computing the optimal value of t, saying t^* for instance, by ignoring the second objective; and then 2) minimizing the second objective soloing by adding the requirement of $t \leq t^*$ as a new constraint. Unfortunately, these two sub-problems are hard to solve. For example, as Theorem 1 specifies, even selecting receivers for these multicast tasks to minimize their completion time is NP-hard in theory. Thus, to obtain an efficient and effective communication topology for the multicast of model parameters, we design heuristic algorithms based on the problem structure.

TABLE I: Notations used in both the problem analysis (§III-A) and algorithm designs (§III-B, §III-C)

Notation	Description
i, j, k, l	indexes of ingresses, egresses, workers, or multicast tasks
κ	the index of the worker newly ready to start to multicast
N	the set of all workers
G	the set of already started multicast tasks
p_i	the multicast rooted at worker i needs p_i receivers at least
v_i	the original volume of the multicast task rooted at worker i
$v_{i,j}$	the remaining volume of the transfer from worker i to j
$c_{i,j}$	the round that worker i has not select j as one of its receiver
D	a set of worker pairs that must be selected for delivery
b_i^I	the available bandwidth on ingress i
b_i^E	the available bandwidth on egress i
τ_i	the time that the SSP training worker i would wait if its multicast task is completed immediately
η	if $v_{i,j} < \eta v_i$, worker j would be selected as the receiver of i in this round of synchronization
$x_{i,j}$	1-0 variable, indicating whether worker i is determined to sends to j
$y_{i,j}$	1-0 variable, indicating whether worker i is temporarily selected to sends to j
s_i	either $\max_{j:j\neq i} x_{i,j}$ or $\sum_{j:j\neq i} x_{i,j}$; see its definition of (9)
t	variable, indicating the remaining time to complete the multicast
t_i	the estimated deadline for the multicast rooted at worker i to complete, under the basic selection

Theorem 1. *For a group of multicast tasks, the optimization problem of selecting no less than p receiver(s) for each of them to minimize their completion time is theoretically NP-hard.*

Proof. We prove the NP-hardness of the optimization problem specified in Theorem 1 by showing that it can be reduced from the corresponding decision problem of "determine whether there exists a receiver selection plan making their completion time no more than a given value t", which is NP-hard.

Obviously, for any instance of the above-mentioned decision problem, we can transform it to: *i*) first compute the optimal result of the optimization problem, saying t_o for example, *ii*) then check whether $t_o \leq t$ is satisfied or not. That is to say, the decision problem reduces to the optimization problem, and thus the former is no harder to solve than the latter [29].

Now, we prove the NP-hardness of the above decision problem, by performing a reduction from the *set-partition problem*, which is NP-hard since the well-known NP-complete *subset sum* problem reduces to it [29], [30]. Generally, the *set-partition* problem can be defined as "determine whether it is possible to partition a set of positive integer numbers, saying S for instance, into two sets A and $\bar{A} = S \setminus A$, such that $\sum_{x\in A} \sum x = \sum_{x\in \bar{A}} x$" [29], [30]. Given an instance in which S involves n elements, namely $v_1, \cdots, v_n$, respectively, we can transform it to an instance of the above decision problem as follows. By appending any two positive values v_{n+1} and v_{n+2}, we obtain a sequence of $n + 2$ numbers. Consider a cluster containing $n + 2$ workers (labeled $w_1, \cdots, w_{n+2}$), where the i-th worker must multicast its local data with the volume of v_i to any other n selected workers; all workers have sufficient uplink bandwidth; the first n workers also have sufficient downlink bandwidth, but the last two (i.e., w_{n+1} and w_{n+2}) only have the same download bandwidth of b. Note that the slowest transfer dominates the completion (i.e., makespan) time of a group of multicast tasks. To minimize the completion time, workers w_{n+1} and w_{n+2} can directly select elements in $S = \{w_1, \cdots, w_n\}$ as their receivers; for each of the first n workers, saying w_j for instance, besides selecting $S \setminus \{w_j\}$ as the receivers, it has to select either w_{n+1} or w_{n+2} as its receiver to meet the requirement of sending local data to n others. Let A and $\bar{A}$ be the sets of workers that select w_{n+1} and w_{n+2}, respectively; then, we have $S = A \cup \bar{A}$ and $A \cap \bar{A} = \emptyset$; and the final completion time is $\frac{\max(\sum_{x\in A} x, \sum_{x\in \bar{A}} x)}{b} \geq \frac{\sum_{x\in S} x}{2b}$. So far, we have transformed the problem of "partition S into two sets equally" into the problem of "determine whether it is possible to find a receiver selection plan yielding the completion time of $t = \frac{\sum_{x\in S} x}{2b}$". Thus, Theorem 1 is proved. □

3) Insight: For the above multi-objective optimization problem, a useful design trick is to first minimize the completion time t, and then fix the value of t to maximize the value of $\sum_{x_{i,j}\in X} x_{i,j}$, i.e., selecting as many receivers as possible. For the minimization of t, if the constraint of (10) is relaxed, the optimization objective can be rewritten as

$$\textit{Minimize} \quad \max_{j\in N} \frac{1}{b_j^E} \sum_{i\in N} v_{i,j} x_{i,j} \tag{13}$$

which gives us the guideline of selecting receivers for each multicast request in a weighted load-balanced way, yielding SELMCAST. In short, SELMCAST involves two passes: *i*) it first selects receivers to meet the minimum scale requirement of each multicast, respecting the goal of minimizing t (i.e., *Basic Selection*); then, *ii*) to make full use of available link capacities, it extends each request's receiver set, provided that including these receivers would not increase their completion times (i.e., *Pareto Improvement*).

B. Receiver Selections for BSP Training

In BSP, workers would move to the next round of training only when all the triggered multicast tasks have been completed. Thus, SELMCAST first selects as few workers as possible to meet the basic requirements (i.e., *Basic Selection*); then based on this basic selection, it estimates the completion time of the parameter synchronization, and tries to select as many receivers as possible, provided the extension of receivers would not hurt the completion time (i.e., *Pareto Improvement*).

1) Basic Selection: Lines 4-14 in Algorithm 1 show the design of how SELMCAST selects receivers to satisfy the basic requirements of receiver number for each parameter multicast request. Suppose that beyond selecting receivers to meet the requirement of D (Lines 4-6), multicast i needs q receivers

Algorithm 1 SELMCAST for BSP Training

```
Require: $N, D, \{b_i^I\}, \{b_i^E\}, \{p_i\}, \{v_i\}, \{c_{i,j}\}$
Ensure: selection plan $\{x_{i,j}\}$
    /*Initialization*/
 1: for each $(i,j) \in N \times N$ do
 2:     $x_{i,j} \leftarrow 0$; $v_{i,j} \leftarrow v_i$
 3: end for
    /*Basic Selection*/
 4: for each $(i,j) \in D$ do                          ▷ for consistency
 5:     $x_{i,j} \leftarrow 1$
 6: end for
 7: $L \leftarrow$ sort $\{i \in N\}$ in non-increasing order of $\sum_{j:j\neq i} x_{i,j}$
 8: for each $i \in L$ do
 9:     $q \leftarrow p_i - \sum_{j:j\neq i} x_{i,j}$               ▷ need $q$ receivers more
10:     $J \leftarrow$ Get the $q$-lightest loaded receivers,
                     where receiver $j$'s load is $\frac{v_i + \sum_{l:l\neq j} v_{l,j} x_{l,j}}{b_j^E}$
11:     for each $j \in J$ do
12:         $x_{i,j} \leftarrow 1$
13:     end for
14: end for
    /*Pareto Improvement*/
15: $t \leftarrow 0$                                ▷ estimated completion time
16: for each $i \in N$ do                  ▷ estimate the completion time
17:     $s_i \leftarrow f(\{x_{i,j} : j \in N\}, \{v_{i,j} : j \in N\})$            ▷ via (9)
18:     $r_i \leftarrow \sum_{l:l\neq i} v_{l,i} x_{l,i}$
19:     $t \leftarrow \max(t, \frac{1}{b_i^I} s_i, \frac{1}{b_i^E} r_i)$
20: end for
21: $C \leftarrow$sort $\{(i,j) : x_{i,j}=0, i \neq j\}$ in non-increasing of $c_{i,j}$
22: for each $(i,j) \in C$ do ▷ select more receivers if possible
23:     $s_i \leftarrow f(\{x_{i,j} : j{\in}N\}, \{v_{i,j} : j{\in}N\})$, provided $x_{i,j} \leftarrow 1$
24:     $r_j \leftarrow \sum_{l:l\neq j} v_{l,j} x_{l,j}$, provided $x_{i,j} \leftarrow 1$
25:     if $t \geq \frac{1}{b_i^I} s_i$ and $t \geq \frac{1}{b_j^E} r_j$ then
26:         $x_{i,j} \leftarrow 1$
27:     end if
28: end for
```

more (Line 9). Motivated by (13), it first selects the q-lightest loaded egresses as receivers provided the corresponding receiver was selected as the receiver (Line 10), and updates the corresponding $x_{i,j}$ (Line 12). In consideration of that the more determined receivers a multicast has, the less flexibility it would have in selecting its receivers, Algorithm 1 processes only these unstarted multicast requests, in non-increasing order of their determined receivers (Lines 7, 8).

2) Pareto Improvement: It is obvious that the selections made by the basic selection of Algorithm 1 do not guarantee *work-conservation*. In other words, there is still remaining bandwidth that could admit more receivers. To address this issue, we further design a Pareto improvement scheme as Lines 15-28 in Algorithm 1 shows. Basically, it first estimates the least completion time that transfers (either multicast or unicast depending on the underlying network) would obtain (Line 15-20). Then, to pursue the goal that the selection of extension receivers would not hurt the completion of all requests, it repeatedly admits a new receiver, provided there is enough remaining bandwidth on both the involved ingress and egress (Line 33). Given the fact that some worker pairs might not have been selected to synchronize for rounds of training, Algorithm 1 checks worker pairs in non-increasing order of the number of their un-selected rounds (Line 21).

Algorithm 2 SELMCAST for SSP Training

```
Require: $N, D, G, \kappa, \eta, \{b_i^I\}, \{b_i^E\}, \{p_i\}, \{v_i\}, \{\tau_i\},$
    $\{x_{i,j}\}, \{v_{i,j}\}, \{c_{i,j}\}$
Ensure: updated selection plan $\{x_{i,j}\}$; updated $G$
    /*Initialization*/
 1: for each $j \in N \setminus \{\kappa\}$ do
 2:     $x_{\kappa,j} \leftarrow 1$; $v_{\kappa,j} \leftarrow v_i$
 3: end for
 4: $G \leftarrow G \cup \{\kappa\}$
 5: for each $(i,j) \in N \times N$ do
 6:     $y_{i,j} \leftarrow 0$
 7: end for
    /*Basic Selection*/
 8: for each $(i,j) \in \{(i,j) : x_{i,j} = 1\}$ do    ▷ for consistency
 9:     if $(i,j) \in D$ or $v_{i,j} < \eta v_i$ then
10:         $y_{i,j} \leftarrow 1$
11:     end if
12: end for
13: $L \leftarrow$ sort tasks in $G$ in non-decreasing order of their $\{\tau_i\}$
14: for each $i \in L$ do
15:     $q \leftarrow p_i - \sum_{j:j\neq i} y_{i,j}$               ▷ need $q$ receivers more
16:     $J \leftarrow$ the $q$-lightest loaded receivers from $\{j : x_{i,j}=1\}$,
                     where receiver $j$'s load is $\frac{v_{i,j} + \sum_{l:l\neq j} v_{l,j} y_{l,j}}{b_j^E}$
17:     for each $j \in J$ do
18:         $y_{i,j} \leftarrow 1$
19:     end for
20: end for
    /*Pareto Improvement*/
21: for each $i \in L$ do               ▷ estimated completion times
22:     $s_i \leftarrow f(\{y_{i,j} : j \in N\}, \{v_{i,j} : j \in N\})$            ▷ via (9)
23:     $t_i \leftarrow \max(\tau_i, \frac{1}{b_i^I} s_i)$
24:     for each $j \in \{j : y_{i,j} = 1\}$ do
25:         $r_j \leftarrow \sum_{l:l\neq j} v_{l,j} y_{l,j}$
26:         $t_i \leftarrow \max(t_i, \frac{1}{b_j^E} r_j)$
27:     end for
28: end for
29: $C \leftarrow$ sort $\{(i,j) : x_{i,j}=1, y_{i,j}=0\}$ in non-increasing of $c_{i,j}$
30: for each $(i,j) \in C$ do ▷ select more receivers if possible
31:     $s_i \leftarrow f(\{y_{i,j} : j{\in}N\}, \{v_{i,j} : j{\in}N\})$, provided $y_{i,j} \leftarrow 1$
32:     $r_j \leftarrow \sum_{l:l\neq j} v_{l,j} x_{l,j}$, provided $y_{i,j} \leftarrow 1$
33:     if $t_i \geq \frac{1}{b_i^I} s_i$ and $t_i \geq \frac{1}{b_j^E} r_j$ then
34:         $y_{i,j} \leftarrow 1$
35:     end if
36: end for
37: for each $(i,j) \in N \times N$ do
38:     $x_{i,j} \leftarrow y_{i,j}$
39: end for
```

C. Receiver Selections for SSP Training

Different from BSP, each worker following SSP can send its local training results to other receivers immediately and then move to the next round of training if the gap between its local training round and the smallest one among all workers is less than the pre-defined threshold k. As a result, all workers would not trigger multicast at the same time. To deal with all the above facts and make efficient usage of the bandwidth, as Algorithm 2 specifies, the designs that SelMcast employs for both the *Basic Selection* and *Pareto Improvement* are a little bit different from those specified in Algorithm 1.

In short, during SSP training, workers would trigger Algorithm 2 to conduct preemptive worker selections in an online manner. Assume that a group of workers G is currently conducting the multicasting using the plan of $\{x_{i,j}\}$, and the remaining volume of the sub-task that worker i delivering to worker j is $v_{i,j}$. For $\forall i \in N \setminus G, \forall j \in N$, we have $x_{i,j} = 0$. Then, a new worker κ is ready for model delivery. To admit it and optimize the delivery, we might preemptively disable some of the receivers for tasks belonging to G.

In SSP, to simplify the management of data transmission, we assume that, for each multicast task, if a worker is excluded from the receiver set, either at the beginning or during the delivery for the optimization of other newly launched multicast tasks, it would not be re-selected to receive data from the same sender in this round of synchronization again. Following this design, the updated receivers for multicast task rooted at worker i are selected from workers whose $x_{i,j}$s are ones (e.g., Lines 8, 16, and 29). Thus, at the very beginning (i.e., Lines 1-3), for the newly ready worker κ, we set $x_{i,j} = 1$ for $\forall j \in N \setminus \{\kappa\}$. And as Lines 5-7 shows, during the procedure, we use $\{y_{i,j}\}$ to cache the new receiver selection plans, which are used to update the value of $\{x_{i,j}\}$ at the end (Lines 37-39).

1) Basic Selection: Compared to Algorithm 1, the basic selection of Algorithm 2 involves three main differences as Lines 8-20 show. Firstly, for the multicast task rooted at worker i, only workers whose $x_{i,j} = 1$ might be selected as the receiver. Secondly, to avoid the waste of bandwidth, if worker j has received a given proportion of the data from worker i, i.e., the amount of remaining volume is less than ηv_i, this worker would be always selected in this round of synchronization. Here, η is a tunable parameter, and $\eta = 0.75$ has shown a good performance in our evaluations (§II). Thirdly, regarding the order of processing multicast tasks, they are checked in non-increasing order of their estimated blocking times: $\{\tau_i\}$. Here, the value of τ_i indicates the time that worker i would still need to wait if its multicast task were completed immediately, under the current settings of SSP. It is generally 0, but would be the estimated completion time of the straggler, in the case that this worker is the bellwether and the gap between their training rounds is larger than the threshold of SSP.

2) Pareto Improvement: Lines 21-36 in Algorithm 2 sketch the procedure of Pareto improvement. Compared with the case of BSP, the main difference for SSP is that SelMcast employs a distinct t_i to estimate the "deadline" for the completion of each multicast task under the basic selection (Lines 21-28) (in BSP, all (sub)tasks share the same t). Then, based on this, SelMcast would try to select as many receivers as possible, provided this "deadline" is still met (Lines 30-36). Following this, the selection of receivers is optimized; during the delivery, strict prioritized yet work-conserving bandwidth allocations are needed for concurrent transfers.

D. Time Complexities of Proposed Algorithms

Now, we analyze the worst-case time complexity of both algorithms. Algorithm 1 is made up of three parts namely *initialization*, *basic selection*, and *Pareto improvement*, respectively. The initialization of $x_{i,j}$s can be conducted with the worst-time complexity of $O(n^2)$. And the core of *basic selection* is to sort workers according to their $\sum_{j:j\neq i}^{x_{i,j}}$s and determine their receivers in order. For each multicast request, the q-lightest loaded receivers would be selected within $O(n^2)$. Thus, the entire *basic selection* can be completed within $O(n^3)$. Regarding *Pareto improvement*, its worst-case time complexity is $O(n^4)$ since both the sort of $\{(i,j) : x_{i,j} = 0, i \neq j\}$ and the selection of receivers followed can be done within $O(n^4)$. Putting the three parts together, the worst-case time complexity of Algorithm 1 is $O(n^4)$. The workflow of Algorithm 2 is quite similar to that of Algorithm 1. Following a similar analysis, we would obtain that its worst-case time complexity does not exceed $O(n^4)$ as well.

IV. Performance Evaluation

Now, we evaluate SelMcast through simulations. As it is designed to provide generic acceleration services for the parameter synchronization of peer-to-peer distributed training, we mainly assess SelMcast from a perspective of system optimization. Extensive results indicate that SelMcast is near-optimal and scalable. Using optimized designs, SelMcast not only reduces the completion time of parameter synchronization for BSP training, or equivalently, improves the average NPU utilization for SSP training, but also increases the number of receivers very efficiently in both cases.

A. Methodology

1) Simulator and baselines: To study the performance of SelMcast in detail, like the recent works [10], [28], we develop a flow-level, discrete-event network simulator with Python 3, which precisely simulates the network behavior of peer-to-peer distributed training systems under various scheduling schemes. We mainly use Orpheus, i.e., selecting receivers randomly [4], and *Optimal*, i.e., selecting receivers following the results of model (12) solved using *lexicographic optimization*, as baselines. By default, consistent with today's network design, we assume that concurrent transfers share link capacities fairly following the principle of per-flow max-min fairness [10]. And when the multicast is implemented at the network layer (i.e., L3), its throughput is determined by the slowest receiver. In summary, we consider the following schemes and scenarios.

- **SelMcast (BSP, SSP)**: multicast receivers are determined using either Algorithms 1 or 2, respecting whether BSP or SSP training is employed;
- **Orpheus (BSP, SSP)**: for both BSP and SSP training, multicast receivers are randomly selected, following [4].

Note that, to guarantee the convergence of partial synchronization, the operator might want a worker to send to all other workers at least once every k rounds of local training. To support this, when determining receivers for the multicast task rooted at s, we modify Orpheus to: *i*) first selects all workers that have not been multicasted by s in the last k rounds; then *ii*) randomly selects more receivers if the number of selected workers has not reached the requirement of p.

- **Optimal (BSP)**: for each round of BSP synchronization, multicast receivers are determined following the results of model (12), solved with *Lexicographic optimization*.

2) Workloads and metrics: In tests, we consider that n workers, networked with an abstract non-blocking switch [23], [27], are training a model collectively. To synchronize the intermediate-trained results, each worker would multicast its result to other p colleagues (by default, in our tests, $p = 0.3$). Besides the training, there might be other services hosted on the same cluster, and the shared network bandwidth is managed proportionally. Accordingly, we assume that the available bandwidth of each egress and ingress on the switch that the training can use is $\bar{b}(1 + \lambda x)$ and $\bar{b}\mu(1 + \lambda x)$, respectively. Here, x is a random value following the uniform distribution of $U[-1; 1]$; λ $(0 \leq \lambda < 1)$ and μ $(\mu > 0)$ are two tunable parameters, controlling the bound of bandwidth variation, and the relative bandwidth of the ingress over that of the ingress, respectively. As specified in Section I, our proposal aims to provide a general communication optimization scheme for peer-to-peer distributed training. Thus, we use system-related metrics like the average completion time of a round of synchronization, the number of receivers involved in each round (i.e., scale), and the utilization of the computation resources for performance evaluations. Given that the convergence behavior of distributed training is jointly determined by a lot of factors and varies with tasks [9], [17], the above metrics are more qualified than metrics like the speed of convergence.

- Workers in BSP training would iterate to the next round of training at the same time. Thus, we mainly assess selection schemes SELMCAST, Optimal, and Orpheus by the quality of multicast topology they conduct in terms of two metrics. The primary is *completion time*, i.e., the completion time normalized by the ideal completion time that the involved synchronization would take to complete; and the secondary is *average multicast scale*, i.e., the number of selected receivers per multicast task, normalized by the total number of workers. Using these normalized metrics, the settings of both the model size and the averaged link bandwidth $\bar{b}$ are insignificant. Besides, we also test the computation time to study their scalabilities, in terms of the computation time each scheme would take. By default, we assume that each worker would multicast their results to other p workers, and let $\lambda = 0.5$, $\mu = 1$, $n = 200$, and $p = 0.3n$. When the multicast is implemented at L3, congestion would mainly occur at egresses. To study the algorithm performances under a situation where ingresses also become the bottlenecks, we also set $\mu = \frac{1}{p}$ for L3 multicast in some tests.

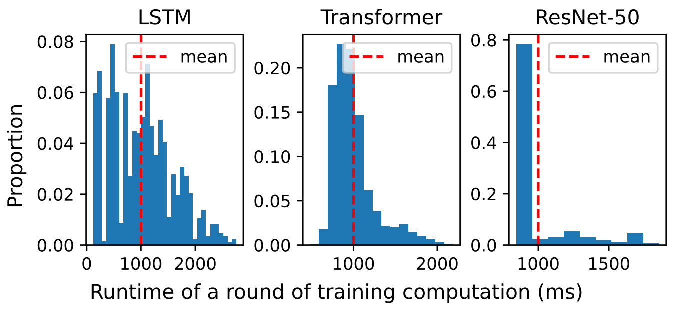


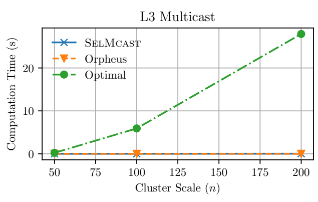
Fig. 3: The distributions of the runtime of a round of training computation for three types of workloads reported by [20]. Runtimes here are re-scaled to have the mean value of 1000ms.

- In the case of SSP, we consider three types of training workloads as Figure 3 shows, i.e., workers are training an LSTM model using the UCF101 dataset, a Transformer model using the WMT16 dataset, and a ResNet-50 (CNN) model using the ImageNet dataset, respectively [20]. Hereafter, they are referred to as LSTM, Transformer, and ResNet-50 for short. We directly use the distribution of the runtime that it would take for a worker to complete a round of training, reported by [20], and re-scale the values such that their average approximates 1000ms, to drive the simulation. Motivated by real-world training configurations, by default, we let $\bar{b}$ = 40Gbps, and assume that the size of data to multicast is 200MB. For different multicast management schemes, besides the *average multicast scale* of all multicast tasks, we mainly use the *average NPU utilization*, i.e., the average of workers' utilizations of computation resources, defined by the ratio of their total computation time to their total training time, as the metric. Basically, given the same or larger multicast scales, the higher computation utilization generally means that the training would converge faster, with both a smaller training round and time cost. By default, we consider that $\eta = 0.75$, both the threshold of SSP and the value of k are 4, and observe the iterative training in 10 minutes to compute the performance. We have observed consistent results under different cluster scales, and the results reported in this paper are based on the case of $n = 50$.

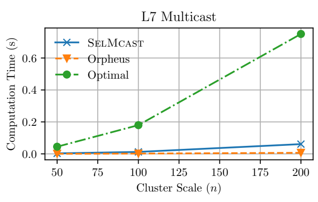
For each parameter setting, we conduct 10 trials to compute and report the *mean* values. When investigating the time costs of the algorithms, to filter out the possible noise introduced by the OS dynamics, we report the *median* values instead.

B. Performance for BSP Training

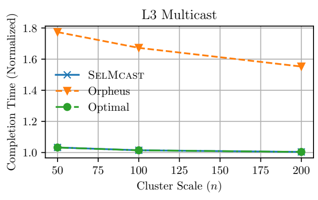
1) Efficiency: Figure 4 shows the computation time costs, normalized synchronization completion times, and average receiver numbers achieved by the schedules of SELMCAST, Orpheus, and *Optimal*, respectively, where the scale of the training cluster increases from 50 to 200 workers. These tests are conducted upon a Ubuntu PC equipped with one AMD Ryzen 5 (3500X 6-Core) CPU Processor and two 8G DDR4 RAM cards. Both SELMCAST and Orpheus only use a single core while the Gurobi-powered *Optimal* will take over



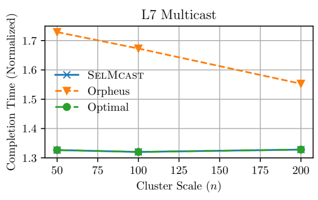
(a) [Median] Efficiency (L3)



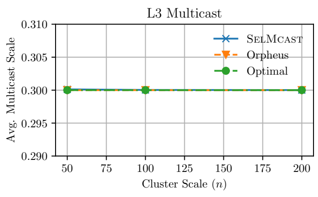
(b) [Median] Efficiency (L7)



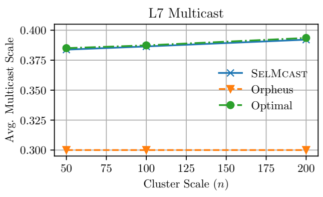
(c) Completion time (L3)



(d) Completion time (L7)

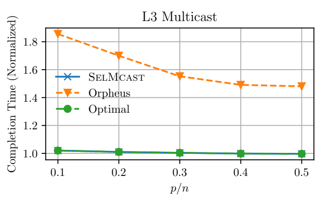


(e) Avg. Multicast scale (L3)

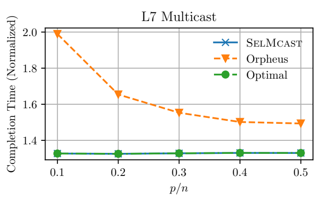


(f) Avg. Multicast scale (L7)

Fig. 4: [BSP] SelMcast is very efficient to achieve near-optimal performances, in terms of both the normalized completion times and average multicast scales. Note that the results of SelMcast generally overlap with those of *Optimal*.



(a) Completion time (L3)



(b) Completion time (L7)

Fig. 5: [BSP] SelMcast achieves consistent normalized completion times when p increases, while the results of Orpheus decrease to approach 1.5, for both L3 and L7 multicast.

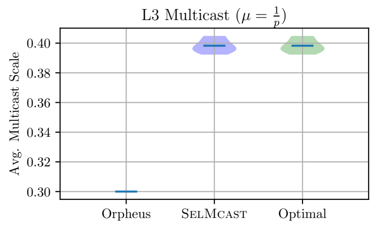


Fig. 6: [BSP] When $\mu = \frac{1}{p}$, SelMcast would select about 33% more receivers per worker than Orpheus for L3 multicast.

all available cores for parallel computation. To ensure that *Optimal* would finish within a reasonable time, its time limit of model solving is set to 60 seconds. As Figures 4a and 4b show, the (median) computation time costs of *Optimal* grow super-linearly for both L3 and L7 multicast tasks. For instance, in the case of selecting L3 multicast receivers for 200 workers, the median computation time of *Optimal* is larger than 27s. Results also show that *i*) the mathematical model of selecting receivers for L7 multicast is much simpler to solve than that of L3 multicast, and *ii*) the speed of *Optimal* highly depends on the problem instance, reaching the limit of 60s in some cases. By contrast, both SelMcast and Orpheus are fast, finishing the computations within tens of milliseconds with a single core.

2) Completion Time: Regarding the completion times of selective parameter synchronization, as Figures 4c and 4d show, despite the achieved performance gain decreasing slightly with the increase of cluster scale, SelMcast always outperforms Orpheus significantly. Take the instance when $n = 100$ as an example, compared to Orpheus, SelMcast reduces the average normalized completion time of synchronization from about 1.67 to 1.02, and from about 1.67 to 1.32, for L3 and L7 multicast, respectively, by using bandwidth-aware receiver selection. To understand the impact of p/n on the selection, we vary its value from 0.1 to 0.5. As Figure 5 shows, although the average normalized completion time achieved by Orpheus drops first, it finally approximates about 1.5 for both L3 and L7 multicast. As a comparison, both SelMcast and the *Optimal* are able to achieve consistently high performances.

3) Number of Receivers: As for the average number of finally selected receivers, SelMcast outperforms Orpheus about 1.3× on selecting receivers for L7 multicast (Figure 4f); while all schemes would select exactly p receivers to meet the requirements thus yielding no performance gains when L3 multicast is employed (Figure 4e). It is reasonable, since in the case of L3 multicast, network congestions generally occur at egresses hence there is no more worker with sufficient bandwidth to select. To verify this, we return the tests by reducing the value of μ from 1 to $\frac{1}{p}$, in which ingresses would be the bottleneck as well. As expected, compared with Orpheus, about 33% more receivers are selected by SelMcast in this instance (Figures 6).

Noteworthily, in all these tests, the results of SelMcast and *Optimal* almost overlap, implying that SelMcast is able to achieve near-optimal receiver selections for both L3 and L7 multicast very efficiently.

C. Performance for SSP Training

Different from BSP training, when workers are conducting heterogeneous distributed training with SSP, multicast requests are likely to appear one after another. Quite different from the case of BSP training, when the L7-based implementation is employed, for each multicast task, the communication bottlenecks mainly occur at the ingress, and SelMcast would select as few receivers as possible to accelerate the completion. Accordingly, we mainly look into the performance of SelMcast under L3-based multicast for SSP training.

1) Case study: Figure 7 shows the details of both the NPU utilization and average multicast tasks of the tested instance. Obviously, SelMcast shows significant performance upon Orpheus in all these three training scenarios—For the training of LSTM, Transformer, and ResNet-50 models, it not only

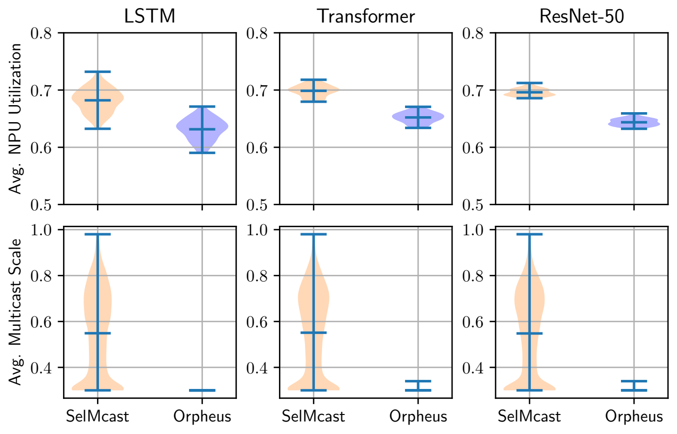


Fig. 7: [SSP] Compared with Orpheus, SELMCAST not only significantly enlarges the scale of multicast but also increases the utilization of computation resources.

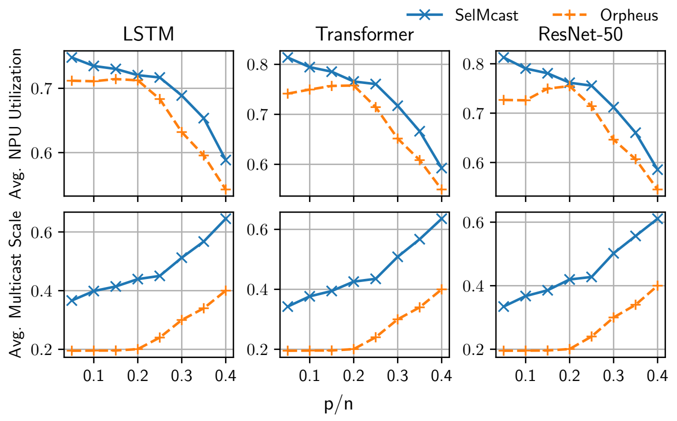


Fig. 8: [SSP] As expected, SELMCAST outperforms Orpheus in all scenarios; and for both schemes, a higher value generally leads to larger average multicast scales but lower average NPU utilizations. Due to the setting of $k = 4$, the results of Orpheus roughly remain consistent when $p/n \leq 0.2$.

enlarges the average normalized scale of multicast from 0.3 to 0.55, but also improves the average utilization of computation resources from about 0.63, 0.65, and 0.64 to about 0.68, 0.70, and 0.70, respectively. It is worth noting that, in the cases of training the models of Transformer and ResNet-50, the normalized scale of multicast scheduled by Orpheus might be larger than the default value of 0.3 in some cases. This is mainly due to the experiment settings—A worker is required to multicast its local result to another receiver if it has not done this in the last k rounds. As a result, the number of receivers for a multicast might be larger than p even under the control of Orpheus. Results also imply that the benefits of communications optimization depend on the type and characteristics of training workloads.

2) Impacts of p/n: To study the impact of the requirement of p on the performance of SELMCAST, we have re-conducted the tests by increasing the value of p from $0.05n$ to $0.4n$. As Figure 8 shows, under the schedule of SELMCAST, with the value of p/n increasing from 0.05 to 0.4, for these three training workloads, the achieved average multicast scale has grown from 0.37, 0.34, and 0.33, to 0.65, 0.64, and 0.61 respectively, while the average NPU utilization has decreased from 0.75, 0.81, and 0.81, to 0.59, 0.59, and 0.59, respectively. This is reasonable since larger multicast scales generally trigger more traffic volumes, thus making the time cost of communication increasing and leading to less NPU utilization. Results also show that SELMCAST significantly outperforms Orpheus in both metrics. For example, when $p = 0.4n$, it could not only improve the average multicast scale more than $0.2n$ but also enlarge the average NPU utilization by about 0.05 at the same time, for all three instances. We also observed that when $p \leq 0.2n$, the increase of p has little impact on the performance of Orpheus. This is mainly due to the setting of k, which ensures that a worker must multicast its result to another worker if it has not done this in its last k rounds of multicasting. Accordingly, when $k = 4$, for each multicast task, the average number of selected receivers would not be less than $1/(k + 1) = 1/(1 + 4) = 0.2$.

3) Impacts of k and ssp: Figure 9 shows the results of both SELMCAST and Orpheus in our scenario. Again, SELMCAST

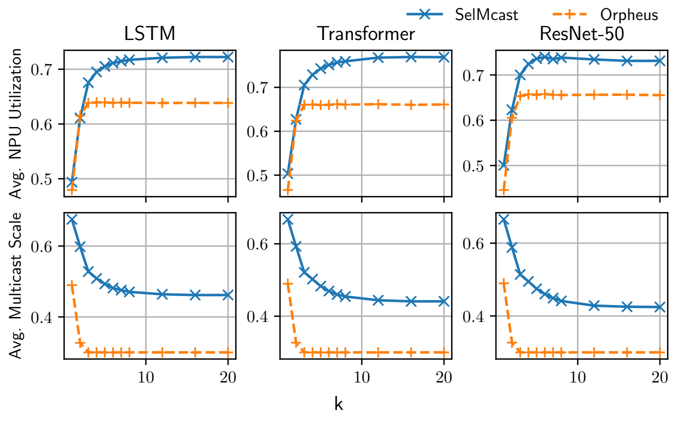


Fig. 9: [SSP] SELMCAST always outperforms Orpheus. For both schemes, with the increase of k, the increments of achieved average NPU utilization and the degradation of achieved average multicast scales decrease, approximating the results of the case when the requirement of k is removed.

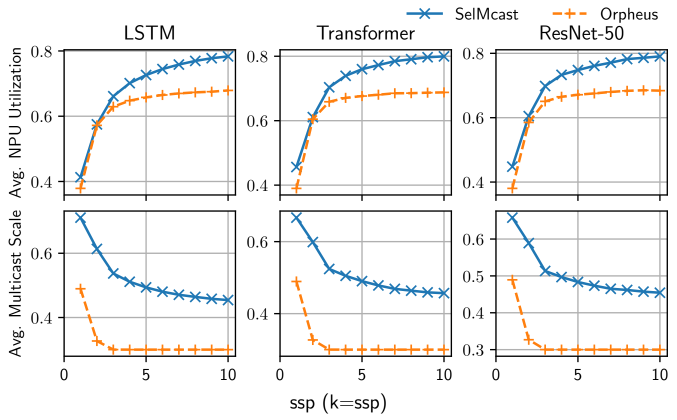


Fig. 10: [SSP] The impacts of ssp on the performance of SELMCAST and Orpheus under the setting of k = ssp are similar to the impacts of k when fixing ssp. SELMCAST consistently achieves better performances.

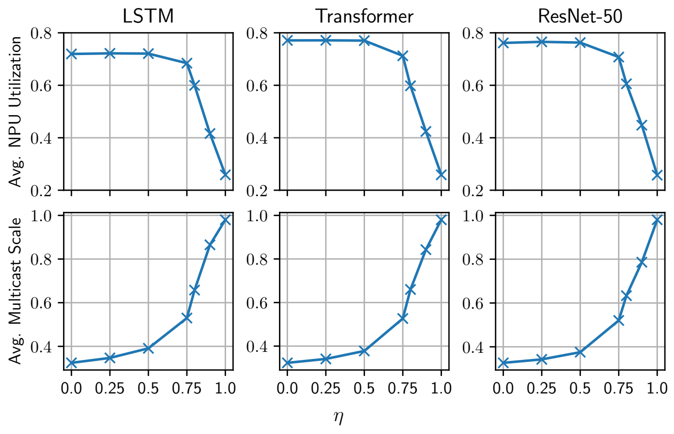


Fig. 11: [SSP] Similar to the impact of parameter k, by tuning the value of η, SELMCAST can trade reduced NPU utilization for a larger average multicast scale. The best setting of η depends on the characteristics of the training workloads.

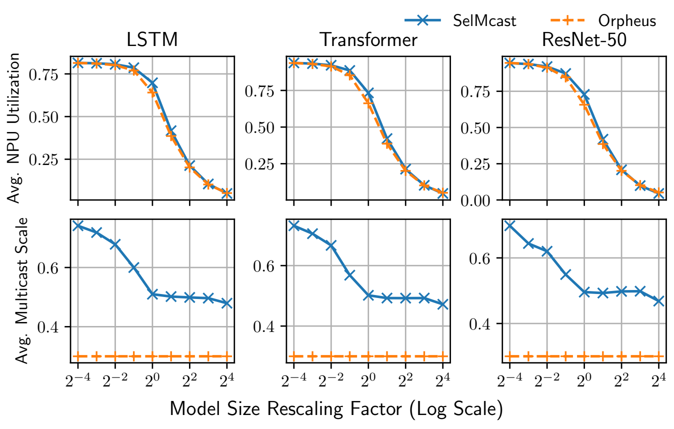


Fig. 12: [SSP] Both the achieved NPU utilization and average multicast scale would decrease with the data size increasing, and SELMCAST could always achieve much larger average multicast scales.

always outperforms Orpheus. With the requirement of k being relaxed, their achieved average multicast scales achieved by both SELMCAST and Orpheus decrease, while their achieved NPU utilization would grow. However, both the increments of achieved average NPU utilization and the degradation of achieved average multicast scales decrease very fast at the same time. As a result, their results would finally approximate the case when the requirement of k is totally removed; in that case, SELMCAST has improved the average NPU utilization from 0.64, 0.66, and 0.66, to 0.72, 0.77, and 0.73, respectively, and the average multicast scale from 0.3 to 0.46, 0.44, and 0.42, respectively. These results indicate up to about 1.17× and 1.53× improvements, in terms of the achieved average NPU utilization and average multicast scale, respectively. Next, we investigate the impacts of ssp on their performance, while keeping the setting of k = ssp. As Figure 10 shows, the results are quite similar to what we obtain in Figure 9.

4) Impacts of η: To study the impacts of the setting of η on the performance of SELMCAST, we range its value from 0.0 to 1.0 to re-conduct the tests. As Figure 11 indicates, with the increase of η, SELMCAST could achieve larger average multicast scales with the cost of lower NPU utilization; also, when $\eta \leq 0.5$, its impact is inconspicuous. Thus, η provides a way to control SELMCAST to trade the NPU utilization for larger average multicast scales. We argue that just letting $\eta = 0.75$ yields a good default choice, as the average multicast scale could be increased by about 66%, 66%, and 58%, while the demotion of NPU utilization is less than 6%, 8%, and 7%, respectively. However, the effects of the improved multicast scale on the coverage speed of the model training highly depend on both the model's property and hyper-parameter settings. Thus, the sweet spot of the setting of η also calls for case-by-case tuning and future study.

5) Impacts of data volume: Figure 12 shows the results when the size of the data to multicast is rescaled by a factor, ranging from 2^{-4} to 2^4. Basically, with the growth of multicast data, the time cost of communication will dominate the entire distributed training. As a result, both the achieved NPU utilization and average multicast scale would decrease. Although the gap between the achieved NPU utilization of SELMCAST and Orpheus might become small, SELMCAST could always achieve much larger average multicast scales.

6) Impacts of bandwidth variability: Last but not least, we study the impacts of bandwidth variability on the performance of SELMCAST by updating the bandwidth of each link every Δ_T using $b = b^{\#}(1-\gamma)+\gamma b^{*}$. Here, $b^{\#}$ is randomly generated at the beginning of the training and b^{*} is randomly generated each time, using the schemes described in Section IV-A. We observe consistent results under the different Δ_T settings and Figure 13 shows results of varying γ from 0 to 1 while letting $\Delta_T = 100ms$. The results of Orpheus show that, even selecting p receivers randomly, the increase of γ would make the achieved average NPU utilization climb up and then decline. We argue that this is mainly caused by the interaction of the involved random factors. A similar result of the NPU utilization is observed for SELMCAST. By using Orpheus as the baseline, Figure 13 also implies that SELMCAST's improvements of NPU utilization also decrease slowly with the increase of γ. When $\gamma = 1$, the NPU utilization of SELMCAST might be even slightly smaller than that of Orpheus. Nevertheless, SELMCAST always achieves larger average multicast scales, with an improvement of up to 67% in all scenarios. Again, the change of SELMCAST's achieved average multicast scale also reflects the fluctuation of the achieved NPU utilization: i.e., a smaller multicast scale leads to a higher NPU utilization.

Overall, all the above performance studies confirm that for both BSP and SSP distributed training, SELMCAST can achieve significantly better performances in terms of the utilization of the computation resources and the scale of scheduled multicast than the state-of-the-art solution Orpheus.

V. RELATED WORK

Optimizing the synchronization of model parameters for large-scale distributed training is a hot topic. The design insight behind SELMCAST is to leverage the fact that a lot of distributed training tasks can tolerate partial parameter synchronization by design. In this section, we briefly overview

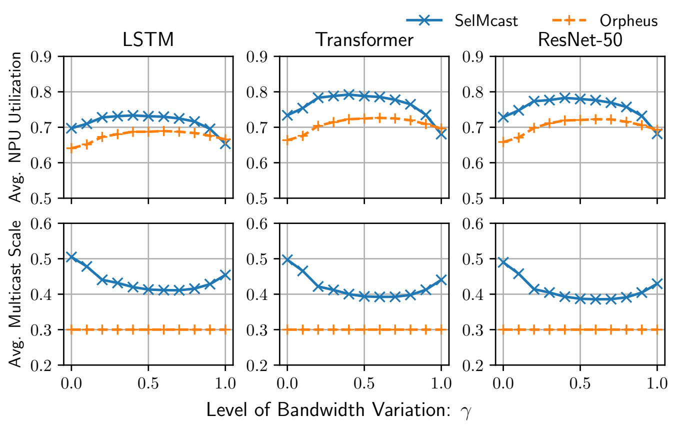


Fig. 13: [SSP] Despite that the performance of SELMCAST would get impacted by the variability of the bandwidth, it generally achieves larger average multicast scales than Orpheus.

the recent proposals sharing this idea and refer the readers to [2], [3] for comprehensive surveys.

P-Reduce [11] and Prague [12] achieve partial parameter synchronization by executing the AllReduce collective operations on a selected set of workers. Similarly, Dutta et al. [13] systematically analyze the benefits of partial synchronization schemes named k-sync SGD, k-batch-sync SGD, k-async SGD, and k-batch-async SGD, for parameter server based distributed training. And Hegedűs et al. [31] further study the performance of Gossip-based partial synchronization. Different from these works, SELMCAST mainly focuses on optimizing peer-to-peer distributed machine learning, where each worker would directly send its updated model to other workers [3]. Our design is motivated by the recent work of Orpheus [4], which selects receivers randomly and is the follow-up work of [14]. Recently, such peer-to-peer communication is also employed by emerging distributed training frameworks built upon serverless computing for model synchronization [22], showing the increasing application scenarios of SELMCAST. Notably, training workers in some systems might be networked with heterogeneous links and parts of them do not have direct connections. In these contexts, topology-aware communication scheduling schemes are needed for efficient model synchronization [10]. For example, works like [10], [32], [33] have explored the idea of generating trees for the synchronization of workers' local results. Accordingly, how to achieve selectively partial synchronizations in such heterogeneous environments yields an interesting future direction.

Besides the level of participating workers, the idea of partial synchronization could also be implemented at the level of parameters. For example, papers like [34] show that only delivering the top-k model gradients works well for many training tasks; BTP finds that many algorithms are robust to random gradient drops with bounded amounts [35]; and DGT [36] further shows that providing different reliabilities to gradients respecting their contributions would release the power of partial synchronization more refined. Based on these observations, more generally, Poco proposes to achieve selective partial completion for collective flows globally [27]. Differently, schemes based on *adaptive parameter freezing* reduce the traffic volume of model synchronization by dynamically selecting the current stable parameters to not synchronize [37]. Obviously, by jointly leveraging the tasks' tolerance of both participating workers and parameters at the same time, it is promising to improve the performance of parameter synchronization further. We leave this as future work.

VI. CONCLUSION

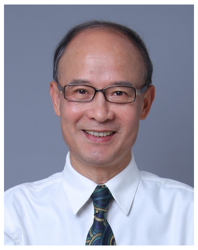
This paper proposes SELMCAST, an expressive bandwidth-aware multicast receiver selection algorithm to manage the communication topology of parameter synchronization for peer-to-peer intra-datacenter distributed learning. SELMCAST is efficient and supports both L3 and L7 multicast-based parameter synchronization by design. Extensive simulations indicate that SELMCAST efficiently achieves near-optimal performance for both BSP and SSP training.

REFERENCES

[1] S. Luo, P. Fan *et al.*, "Fast parameter synchronization for distributed learning with selective multicast," in *Proceedings of the IEEE ICC*, 2022, pp. 4775–4780.
[2] S. Shi, Z. Tang *et al.*, "A quantitative survey of communication optimizations in distributed deep learning," *IEEE Network*, vol. 35, no. 3, pp. 230–237, 2021.
[3] J. Verbraeken, M. Wolting *et al.*, "A survey on distributed machine learning," *ACM Computing Surveys*, vol. 53, no. 2, Mar. 2020.
[4] P. Xie, J. K. Kim *et al.*, "Orpheus: Efficient distributed machine learning via system and algorithm co-design," in *Proceedings of the ACM SoCC*, 2018, pp. 1–13.
[5] S. Luo, P. Fan *et al.*, "Eliminating communication bottlenecks in cross-device federated learning with in-network processing at the edge," in *Proceedings of the IEEE ICC*, 2022, pp. 4601–4606.
[6] S. Luo, X. Yu *et al.*, "Releasing the power of in-network aggregation with aggregator-aware routing optimization," *IEEE/ACM Transactions on Networking*, vol. 32, no. 5, pp. 4488–4502, 2024.
[7] A. Sapio, M. Canini *et al.*, "Scaling distributed machine learning with in-network aggregation," in *Proceedings of the 18th NSDI*, Apr. 2021, pp. 785–808.
[8] L. Luo, Y. Zhang *et al.*, "Fast synchronization of model updates for collaborative learning in micro-clouds," in *Proceedings of the IEEE 23rd HPCC*, 2021, pp. 831–836.
[9] S. Luo, R. Wang *et al.*, "Efficient cross-cloud partial reduce with crew," *IEEE Transactions on Parallel and Distributed Systems*, vol. 35, no. 11, pp. 2224–2238, 2024.
[10] S. Luo, R. Wang, and H. Xing, "Efficient inter-datacenter allreduce with multiple trees," *IEEE Transactions on Network Science and Engineering*, vol. 11, no. 5, pp. 4793–4806, 2024.
[11] X. Miao, X. Nie *et al.*, "Heterogeneity-aware distributed machine learning training via partial reduce," in *Proceedings of SIGMOD*. ACM, Jun 2021, pp. 2262–2270.
[12] Q. Luo, J. He *et al.*, "Prague: High-performance heterogeneity-aware asynchronous decentralized training," in *Proceedings of the 25th ASPLOS*. ACM, Mar 2020, pp. 401–416.
[13] S. Dutta, J. Wang, and G. Joshi, "Slow and stale gradients can win the race," *IEEE Journal on Selected Areas in Information Theory*, vol. 2, no. 3, pp. 1012–1024, 2021.
[14] P. Xie, J. K. Kim *et al.*, "Lighter-communication distributed machine learning via sufficient factor broadcasting," in *Proceedings of the 32nd Conference on Uncertainty in Artificial Intelligence (UAI)*, 2016, pp. 795–804.
[15] H. Li, A. Kadav *et al.*, "Malt: Distributed data-parallelism for existing ml applications," in *Proceedings of the 10th EuroSys*. ACM, 2015.
[16] Q. Hu, Z. Ye *et al.*, "Hydro: Surrogate-Based hyperparameter tuning service in datacenters," in *Proceedings of the 17th OSDI*. Boston, MA: USENIX Association, Jul. 2023, pp. 757–777.
[17] L. Mai, G. Li *et al.*, "KungFu: Making training in distributed machine learning adaptive," in *Proceedings of the 14th OSDI*. USENIX Association, Nov. 2020, pp. 937–954.

[18] Q. Ho, J. Cipar *et al.*, "More effective distributed ml via a stale synchronous parallel parameter server," in *Proceedings of the 26th International Conference on Neural Information Processing Systems (NIPS)*. Red Hook, NY, USA: Curran Associates Inc., 2013, pp. 1223–1231.
[19] H. Cui, J. Cipar *et al.*, "Exploiting bounded staleness to speed up big data analytics," in *Proceedings of the USENIX ATC*. USA: USENIX Association, 2014, pp. 37–48.
[20] S. Li, T. Ben-Nun *et al.*, "Taming unbalanced training workloads in deep learning with partial collective operations," in *Proceedings of the 25th PPoPP*. ACM, 2020, pp. 45–61.
[21] X. Zhao, A. An *et al.*, "Dynamic stale synchronous parallel distributed training for deep learning," in *Proceedings of the 39th IEEE ICDCS*, 2019, pp. 1507–1517.
[22] A. Barrak, "The promise of serverless computing within peer-to-peer architectures for distributed ml training," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 21, pp. 23 383–23 384, Mar. 2024.
[23] S. Luo, H. Yu *et al.*, "Efficient file dissemination in data center networks with priority-based adaptive multicast," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 6, pp. 1161–1175, Jun 2020.
[24] S. Luo, H. Xing, and P. Fan, "Softwarized ip multicast in the cloud," *IEEE Network*, vol. 35, no. 6, pp. 233–239, 2021.
[25] S. Li, O. Mangoubi *et al.*, "Sync-switch: Hybrid parameter synchronization for distributed deep learning," in *Proceedings of the 41st IEEE ICDCS*, 2021, pp. 528–538.
[26] S. Luo, H. Yu *et al.*, "Towards practical and near-optimal coflow scheduling for data center networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 11, pp. 3366–3380, 2016.
[27] S. Luo, P. Fan *et al.*, "Selective coflow completion for time-sensitive distributed applications with poco," in *Proceedings of the 49th ICPP*, 2020.
[28] S. Luo, P. Fan *et al.*, "Meeting coflow deadlines in data center networks with policy-based selective completion," *IEEE/ACM Transactions on Networking*, vol. 31, no. 1, pp. 178–191, 2023.
[29] T. H. Cormen, C. E. Leiserson *et al.*, *Introduction to Algorithms, Third Edition*, 3rd ed., 2009.
[30] Lawrence L. Larmore, "The Partition Problem is NP–complete," https://web.cs.unlv.edu/larmore/Courses/CSC456/ssPart.pdf, [Online; accessed 10-Oct-2024].
[31] I. Hegedűs, G. Danner, and M. Jelasity, "Decentralized learning works: An empirical comparison of gossip learning and federated learning," *Journal of Parallel and Distributed Computing*, vol. 148, pp. 109–124, 2021.
[32] Z. Zhang, C. Wu, and Z. Li, "Near-optimal topology-adaptive parameter synchronization in distributed dnn training," in *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, 2021, pp. 1–10.
[33] Z. Li, W. Feng *et al.*, "Accelerating geo-distributed machine learning with network-aware adaptive tree and auxiliary route," *IEEE/ACM Transactions on Networking*, pp. 1–16, 2024.
[34] S. Shi, Q. Wang *et al.*, "A distributed synchronous sgd algorithm with global top-k sparsification for low bandwidth networks," in *Proceedings of the 39th IEEE ICDCS*, 2019, pp. 2238–2247.
[35] J. Xia, G. Zeng *et al.*, "Rethinking transport layer design for distributed machine learning," in *Proceedings of the 3rd Asia-Pacific Workshop on Networking (APNet)*, 2019, pp. 22–28.
[36] H. Zhou, Z. Li *et al.*, "DGT: A contribution-aware differential gradient transmission mechanism for distributed machine learning," *Future Generation Computer Systems*, vol. 121, pp. 35–47, 2021.
[37] C. Chen, H. Xu *et al.*, "Synchronize only the immature parameters: Communication-efficient federated learning by freezing parameters adaptively," *IEEE Transactions on Parallel and Distributed Systems*, vol. 35, no. 7, pp. 1155–1173, 2024.

Shouxi Luo (Member, IEEE) received the bachelor's degree in communication engineering and Ph.D. degree in communication and information systems from the University of Electronic Science and Technology of China, China, in 2011 and 2016, respectively. He is currently an Associate Professor with Southwest Jiaotong University. His research interests include data center networks, software-defined networking, and networked systems.



Pingzhi Fan (Fellow, IEEE) received the M.Sc. degree in computer science from Southwest Jiaotong University, China, in 1987, and the Ph.D. degree in electronic engineering from Hull University, U.K., in 1994. He is currently a Presidential Professor with Southwest Jiaotong University. His research interests include high mobility wireless communications, massive random-access techniques, etc. He is a fellow of IEEE, IET, CIE, and CIC.

Ke Li received the B.S. degree in electronic and information engineering from Sichuan University, China, and the Ph.D. degree in communication and information systems from the University of Electronic Science and Technology of China, China. She is currently a Lecturer with Southwest Jiaotong University. Her research interests include machine learning, the Internet of Things, etc.

Huanlai Xing (Member, IEEE) received the B. Eng. degree in communications engineering from Southwest Jiaotong University, China, in 2006, the M. Eng. degree in electromagnetic fields and wavelength technology from the Beijing University of Posts and Telecommunications, China, in 2009, and the Ph.D. degree in computer science from the University of Nottingham, U.K., in 2013. Currently, he is an Associate Professor with Southwest Jiaotong University. His research interests include mobile edge computing, evolutionary computation, etc.

Long Luo received the M.S. and Ph.D. in communication and information systems from the University of Electronic Science and Technology of China, China, in 2015 and 2020, respectively. She is currently an Associate Professor with the University of Electronic Science and Technology of China. Her research interests include networking and distributed systems, etc.

Hongfang Yu (Member, IEEE) received the Ph.D. degree in communication and information systems from the University of Electronic Science and Technology of China, China, in 2006. She is currently a Professor with the University of Electronic Science and Technology of China. Her research interests include SDN/NFV, data center networks, networking for AI systems, and network security, etc.