# Fast Parameter Synchronization for Distributed Learning with Selective Multicast

Shouxi Luo[1] Pingzhi Fan[1] Ke Li[1] Huanlai Xing[1] Long Luo[2] Hongfang Yu[2]
[1]*Southwest Jiaotong University* [2]*University of Electronic Science and Technology of China*

***Abstract*—Recent advances in distributed machine learning show theoretically and empirically that, for many models, provided workers would participate in the synchronizations eventually, *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, having motivated several optimization designs.**

**In this paper, we focus on optimizing the parameter synchronization for *peer-to-peer* distributed learning, in which workers generally broadcast or multicast their updated parameters to others for synchronization, and propose SELMCAST, an expressive and *Pareto-optimal* multicast receiver selection algorithm, to achieve the goal. Compared with the state-of-the-art design that 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. Firstly, it could optimize the bottleneck sending rate, thus cutting down the time cost of parameter synchronization. Secondly, when more than $p$ receivers are with sufficient bandwidth, they would be selected as many as possible, bringing benefits to the convergence of training. Extensive evaluations show that SELMCAST is efficient and always achieves near-optimal performance.**

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

## I. INTRODUCTION

Over the past decade, machine learning techniques have obtained huge success and have been widely employed for various applications like *email filtering*, *advertising recommendation*, *speech recognition*, *machine translation*, *computer vision*, etc [1]–[4]. With the increasing popularity of machine learning and rapid developments 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 becomes the key to obtaining the resulting model over such a massive of data within reasonable times [1]–[3].

In datacenter-based data-parallel distributed training, the dataset is generally split then distributed among a group of homogeneous 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 like the *star* (i.e., parameter server), *tree*, *ring*, and *peer-to-peer* [2]. These schemes have various different properties regarding *latency*, *traffic overheads*, and *reliability*, thus having been 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 [5]. 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 [1], [2], [6]. Recent advances show theoretically and empirically that, for many models, provided workers would participate in the synchronization eventually, *i*) the training would still converge, even if there are only $p$ workers taking part in each round of synchronization, and *ii*) a larger value of $p$ in average generally leads to a smaller round of training to converge [3], [7]–[11]. These findings shed light on the optimization of parameter synchronization, having motivated the improvements of serval learning algorithms and systems [3], [7]–[11].

In this paper, we focus on accelerating the parameter synchronization for *peer-to-peer* distributed learning, following the works of Orpheus [3], Malt [11], and SFB [10], and propose SELMCAST an expressive and *Pareto-optimal* 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, or the model update, or the sufficient factor of the model update to all other workers [3], [10]. As a result, the synchronization of $n$ workers would yield $O(n^2)$ traffic volumes. For this issue, Orpheus [3], the state-of-the-art proposal, lets each worker only deliver model updates to $p$ (out of $n-1$) randomly selected receivers in a bandwidth-agnostic way. Such a design does reduce the total traffic volume and guarantees eventual convergence. However, it still suffers from two serious problems. Firstly, since the completion of model

The work of Shouxi Luo was supported by NSFC Project 62002300 and China Postdoctoral Science Foundation Project 2019M663552. The work of Pingzhi Fan was supported by NSFC Project 62020106001 and 111 Project 111-2-14. The work of Ke Li was supported by Project of Network and Data Security Key Laboratory in Sichuan Province NDS2022-1. The work of Long Luo was supported by NSFC Project 62102066.

Shouxi Luo is the corresponding author (sxluo@swjtu.edu.cn).

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 a fewer rounds of training to converge [7], [8]; while Orpheus misses this opportunity of optimization since it only chooses $p$ receivers even if more receivers are with sufficient bandwidth.

As a comparison, SELMCAST picks receivers based on the global view of their available bandwidth and loads heuristically. Despite that the original selection problem is hard in theory, powered by insights stemming from a model-based analysis of the problem, SELMCAST is efficient to achieve *Pareto-optimal* selections efficiently. Extensive evaluations imply that it makes near-optimal selections, outperforming Orpheus significantly, in terms of both the completion time of synchronization and the number of selected receivers.

In summary, the contributions of this paper are four-fold:

1) A thorough analysis of the drawbacks of random bandwidth-agnostic multicast receiver selection (§II-B).
2) A pair of expressive mixed-integer and integer programmings that describe the optimal receiver selection problem and motivate our algorithm designs (§III-B).
3) SELMCAST, an $O(n^3)$ bandwidth-aware receiver selection algorithm that constructs *Pareto-optimal* multicast topologies for parameter synchronization (§III-C).
4) Extensive evaluations, showing that SELMCAST is effective, efficient, and *near-optimal* (§IV).

Next, we first overview the background and motivation in §II, then formulate the problem, analyze it, and propose SELMCAST to solve in §III. After that, performance evaluation and related work discussion follow in §IV and §V, respectively. Finally, Section VI concludes the paper.

## II. BACKGROUND AND MOTIVATION

### 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, based on which, these workers train their local replicas of the model iteratively in parallel. To guarantee convergence, workers synchronize their updated local models periodically, e.g., in every $k$ epochs of local training. Regarding the implementation of model synchronization, there are four types of basic communication topology designs widely used by DDP training algorithms today, namely, *stars (i.e., parameter server)*, *trees*, *rings*, and *peer-to-peer*, respectively [2].

These designs have different properties in terms of *latency*, *traffic overheads*, *reliability*, etc., targetting for various application scenarios. Among them, the *peer-to-peer* architecture is fully decentralized, in which workers communicate with each other directly, thus eliminating the single point of failure and bottleneck. Currently, peer-to-peer parameter synchronization has been supported by many distributed machine learning frameworks like TensorFlow, Pytorch, MXNet, MALT, Orpheus, and used by numerous training algorithms [3], [10], [11]. Thus, in this paper, we focus on optimizing the parameter synchronization for peer-to-peer distributed learning.

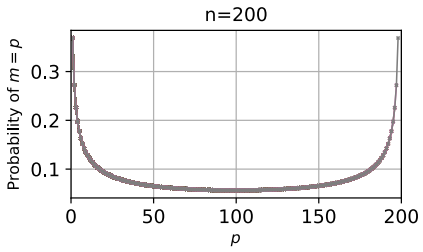



Fig. 1: The probability that a given worker happens to obtain the balanced load (i.e., $m = p$) is small; it decreases drastically then increases slowly, with the growth of $p$.

### B. Drawback of State-of-the-art

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 [2], [11]. 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 [3], Malt [11], and partial SFB [10] 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 [3] as an example. Suppose that there are $n$ workers training a model with data parallelism. At each round of synchronization, each worker in Orpheus would deliver its model to other $p$ ($1 \le p \le n-1$) 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 receivers 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 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 link capacity, random receiver selection would lead to 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 mismatching

between the selection of receivers and the available bandwidth makes the time costs of model synchronization unoptimized with very high probabilities.

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

## III. SELECTIVE MULTICAST

SELMCAST is a multicast receiver selection algorithm running at a centralized manager that helps distributed training workers determine the communication topologies for the multicast of their parameters. 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) [1], [12]. Accordingly, SELMCAST should be expressive to support both L3 and L7 multicast implementations at the same time. During the training, the manager collects the available bandwidth of each worker, based on which, SELMCAST computes *Pareto-optimal* multicast topologies for each worker to control the synchronization. Once selected by SELMCAST, workers will launch either real multicast (L3) or unicast (L7) transfers to conduct the delivery, according to the underlying network [12], [13].

### A. Design Overview

As load-agnostic, the random multicast adopted by [3] is likely to meet with load imbalance with high probability. The core idea of SELMCAST is to choose receivers for parameter multicast transfers carefully, such that it would take less time for the synchronization to complete. We not only control the minimum number of receives for each multicast, but also ensure that each worker always multicast its parameter to all other workers at least one time in every $k$ rounds of synchronization, such that eventual (global) synchronization is ensured. Moreover, to accelerate the convergence, we let more workers participate in the multicast, in case their joins would not postpone the multicast's completion.

### B. Problem Analysis

**Math Formulation**. Given the non-blocking design of modern data center networks [12], [14], [15], we abstract the entire data center network out as one big switch, in which congestions only occur at ingresses and egresses. Without loss of generality, we assume that the training task involves $n$ workers 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 worker $i$ in this round of synchronization, or not; then, we have constraints (2) and (3).

$$\sum_{j:j\neq i} x_{i,j} \geq p_i, \qquad \forall i \tag{2}$$

$$x_{i,j} \in \{0,1\}, \qquad \forall i, j \neq i \tag{3}$$

Moreover, to guarantee convergence, each worker should synchronize with all other workers at least once every $k$ rounds. Let $D$ be the set of worker pairs out of model delivery in the last $k-1$ rounds. For $(i,j) \in D$, the corresponding $x_{i,j}$ would be enforced to 1 at this round, as (4) shows.

$$x_{i,j} = 1, \qquad (i,j) \in D \tag{4}$$

Obviously, 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 [12]. 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 [14], a parameter synchronization is treated as done if and only if all the selected deliveries have finished. As all workers are training exactly the same model, just enforcing all involved transfers to send data at the same rate would not harm the completion of the entire synchronization. Let $y$ be the rate of model delivery; to avoid congestions, we would have constraints (5) and (6). Here, the value of $\Gamma_i$ is either 1 or $\sum_{j:j\neq i} x_{i,j}$ ($\geq p_i \geq 1$), as (7) denotes, respecting whether L3 multicast is employed or not.

$$0 < y\Gamma_i \leq b_i^I, \qquad \forall i \tag{5}$$

$$\sum_{i:i\neq j} x_{i,j} y \leq b_j^E, \qquad \forall j \tag{6}$$

$$\Gamma_i = \begin{cases} 1 & \text{if multicast at Layer 3} \\ \sum_{j:j\neq i} x_{i,j} & \text{if multicast at Layer 7} \end{cases} \tag{7}$$

Given that all workers train the same model synchronously, pursuing the objective of maximizing the minimum multicast rate among all workers, as the non-linear programming of (8) shows, leads to the minimum time cost of synchronization.

$$\textit{Maximize} \ \{y: \ (2) \wedge (3) \wedge (4) \wedge (5) \wedge (6) \wedge (7)\} \tag{8}$$

For the acceleration of convergence, once the maximum delivery rate is obtained, saying $y^*$ for instance, the training task would prefer more receivers to participate in the synchronization. To this end, we maximize the number of receivers, as the integer programming of (11) formulates.

$$\Gamma_i \leq \frac{b_i^I}{y^*}, \qquad \forall i \tag{9}$$

$$\sum_{i:i\neq j} x_{i,j} \leq \frac{b_j^E}{y^*}, \qquad \forall j \tag{10}$$

$$\textit{Maximize} \left\{ \sum_{(i,j):i\neq j} x_{i,j} : (2) \wedge (3) \wedge (4) \wedge (7) \wedge (9) \wedge (10) \right\} \tag{11}$$

**Reformulation and Hardness**. Now, we show that Problem (8) can be equivalently reformated as *mixed-integer programming*, which is generally NP-hard in theory.

Let $z$ be the value of $\frac{1}{y}$; then, the constraints of (5) and (6) can be equivalently reformated to (12) and (13), respectively.

$$z \geq \frac{1}{b_i^I}\Gamma_i, \qquad \forall i \tag{12}$$

$$z \geq \frac{1}{b_j^E}\sum_{i:i\neq j} x_{i,j}, \qquad \forall j \tag{13}$$

Following this, the model of (8) can be rewritten as the *mixed-integer linear programming* problem of (14). To obtain efficient and effective topology, next, we design *Pareto-optimal* heuristic algorithms based on the problem structure.

$$\textit{Minimize} \quad \{z: \ (2) \wedge (3) \wedge (4) \wedge (7) \wedge (12) \wedge (13)\} \tag{14}$$

### C. Algorithm Design

**Insight**. For the above problem, if the constraint of (12) is relaxed, the optimization objective can be rewritten as

$$\textit{Minimize} \quad \max_j \frac{1}{b_j^E}\sum_{i:i\neq j} x_{i,j} \tag{15}$$

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: it first selects receivers to meet the minimum scale requirement of each multicast (i.e., *Basic Selection*); then, 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*).

**Basic Selection**. Algorithm 1 shows the design of how SELMCAST selects receivers to satisfy the basic requirements of receiver number for each parameter multicast request. Suppose that multicast $i$ needs $q$ receivers more (Line 4). Motivated by (15), Algorithm 1 first computes the load of each egress, provided the corresponding receiver was selected as the receiver (Line 5). Then, it selects the $q$-lightest loaded egresses as receivers and updates the corresponding $x_{i,j}$ (Line 6). Note that, to guarantee convergence, each multicast request might have some pre-determined receivers (i.e. these specified by $D$, Line 1). In consideration of that the more determined receivers a multicast has, the less flexibility it would have in selecting their receivers, Algorithm 1 processes multicast requests in non-increasing order of their determined receivers (Line 2,3).

**Pareto Improvement**. It is obvious that the selections made by Algorithm 1 are not *Pareto-optimal* thus 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 Algorithm 2. Basically, it first computes the global bottleneck sending rate that transfers (either multicast or unicast depending on the underlying network) would obtain under per-flow fair sharing (Line 1-4). Here, $\epsilon$ is a small value that avoids the error of zero denominators in Line 4. Then, to ensure 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 8). Given the fact that some worker pairs might have not been selected to synchronize for rounds of training, Algorithm 2 checks worker pairs in non-increasing order of the number of their un-selected rounds (Line 6-10).

Finally, after the receiver sets of all multicast tasks are determined, training workers launch either multicast or unicast transfers to carry out the delivery, and these transfers would share the network bandwidth fairly. Regarding their time complexities, Algorithms 1 and 2 could be implemented as $O(n^3)$ and $O(n^2 \ln n)$, respectively.

**Algorithm 1** SELMCAST: Basic Selection

**Inputs:** demand: $\{p_i\}$, $D$; bandwidth:$\{b_i^E\}$
**Output:** selection: $\{x_{i,j} : i \neq j\}$
1: $x_{i,j} \leftarrow 1$ if $(i,j) \in D$ else 0, for all allowable $(i,j)$
2: $L \leftarrow$sort multicasts in non-increasing of their $\sum_{j:j\neq i} x_{i,j}$s
3: **for** each $i \in L$ **do**
4: $\quad q \leftarrow p_i - \sum_{j:j\neq i} x_{i,j}$ ▷ need $q$ receivers more
5: $\quad l_j \leftarrow \frac{1+\sum_{i:i\neq j} x_{i,j}}{b_j^E}$ for each $j \in \{j : x_{i,j} = 0 \wedge j \neq i\}$
6: $\quad x_{i,j} \leftarrow 1$ for the $q$-lightest loaded receivers acc. to $l_j$s
7: **return** $\{x_{i,j} : i \neq j\}$

**Algorithm 2** SELMCAST: Pareto Improvement

**Inputs:** selection: $\{x_{i,j} : i \neq j\}$; bandwidth: $\{b_i^I\}$, $\{b_i^E\}$
number of rounds without multicast: $\{c_{i,j} : i \neq j\}$
**Output:** selection: $\{x_{i,j} : i \neq j\}$
1: $r_b \leftarrow +\inf$ ▷ bottleneck rate
2: **for** $i \leftarrow 1, \cdots, n$ **do** ▷ find the global bottleneck rate
3: $\quad$ calculate $\Gamma_i$ (7) with $\{x_{i,j} : i \neq j\}$
4: $\quad r_b \leftarrow \min(r_b, \frac{b_i^I}{\Gamma_i}, \frac{b_i^E}{\max(\epsilon, \sum_{l:l\neq i} x_{l,i})})$
5: $G \leftarrow$sort $\{(i,j) : x_{i,j}=0\}$ in non-increasing of their $c_{i,j}$s
6: **for** each $(i,j) \in G$ **do** ▷ perform pareto improvements
7: $\quad$ Let $\Gamma_i^*$ be the new value of $\Gamma_i$ provided $x_{i,j} \leftarrow 1$
8: $\quad$ **if** $\Gamma_i^* \leq \frac{b_i^I}{r_b}$ **and** $\sum_{l:l\neq j} x_{l,j} \leq \frac{b_j^E}{r_b}$ **then**
9: $\quad\quad x_{i,j} \leftarrow 1$ ▷ select a new receiver if possible
10: $\quad\quad$ Update the value of $\Gamma_i$ (7)
11: **return** $\{x_{i,j} : i \neq j\}$

## IV. PERFORMANCE EVALUATION

In this section, we evaluate SELMCAST through simulations. Extensive results indicate that SELMCAST is near-optimal and scaleable. With optimized designs, it not only reduces the completion time of parameter synchronization but also increases the number of receivers very efficiently. outperforming the state-of-the-art Orpheus significantly (e.g., up to about 1.67× and 33%, respectively, in some cases), and approximating the *Optimal* tightly (e.g., almost overlapped).

### A. Methodology

**Metrics and Baselines**. In tests, we mainly use Orpheus, i.e., selecting receivers randomly [3], and *Optimal*, i.e., selecting

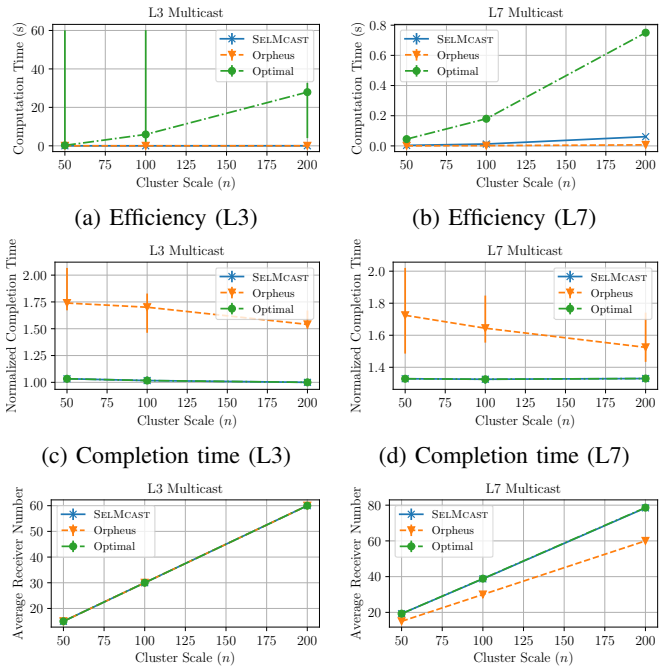
receivers following the results of model (14) and (11), as baselines. Since $\sum_{j:j\neq i} x_{i,j} \geq p_i$, when solving (14) for L7 multicast, we directly let $\Gamma_i$ be $p_i$ for simplification without hurting optimality. For each scheme, it is assessed by the quality of the multicast topology it conducts, in terms of the time normalized by the ideal completion time that the involved synchronization would take to complete (primary), and the number of selected receivers per multicast task (secondary). Consistent with today's network design, we assume that concurrent transfers share link capacities fairly. And when the multicast is implemented at the network layer (i.e., L3), its throughput is determined by the slowest receiver. Besides, we also test the computation time to study its scalability.

**Network and Workload**. In tests, we consider that $n$ homogeneous workers, networked with an abstract non-blocking switch [12], [15], are training a model collectively. 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 train can use is $1+\lambda x$ and $\mu(1+\lambda x)$, respectively. Here, $x$ is a random value following the uniform distribution of $U[-1;1]$; $\lambda$ ($0 \leq \lambda < 1$) and $\mu$ ($\mu > 0$) are two turnable parameters, controlling the bound of bandwidth variation, and the relative bandwidth of the ingress over that of the ingress, respectively. Regarding the workload, we assume that each worker would multicast the model with the size of 1 unit to other $p$ workers. By default, $\lambda = 0.5$, $\mu = 1$, $n = 200$, and $p = 0.3n$. When the multicast is implemented at L3, congestions would mainly occur at egresses. To study the algorithm performances under the situation that ingresses also become the bottlenecks, we also set $\mu = \frac{1}{p}$ for L3 multicast in some tests.

**Simulator**. We develop a flow-level simulator with Python 3, which precisely simulates the aforementioned synchronization process under the schedules of Orpheus, *Optimal*, and SELMCAST. When performing *Optimal*, the simulator employs the off-the-shelf commercial Gurobi solver [16] for model solving. For each parameter setting, we conduct 10 trials to compute and report the *minimum*, *medium*, and *maximum* values.

### B. Performance

Figure 2 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 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 2a and 2b show, the computation time costs of *Optimal* grow super-linearly for both L3 and L7 multicast tasks. For instance, in the case of selecting L3




(a) Efficiency (L3) (b) Efficiency (L7)

(c) Completion time (L3) (d) Completion time (L7)

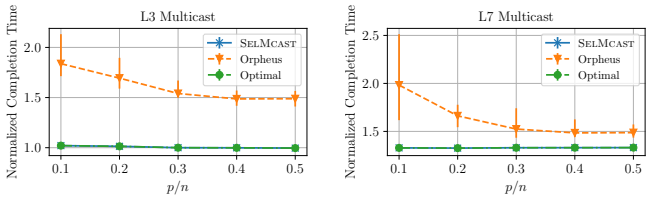
(e) Average receiver number (L3) (f) Average receiver number (L7)

Fig. 2: SELMCAST is very efficient to achieve near-optimal performances, in terms of both the normalized completion times and average selected receiver numbers. Note that the results of SELMCAST generally overlap with those of *Optimal*.

multicast receivers for 200 workers, the medium computation time of *Optimal* is larger than 27 seconds. Results also show that *i*) the math 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 60 seconds in some cases. By contrast, both SELMCAST and Orpheus are very fast, finishing the computations within tens of milliseconds with a single core.

Regarding the completion times of selective parameter synchronization, as Figures 2c and 2d 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 with Orpheus, SELMCAST reduces the normalized completion time of synchronization from about 1.7 to 1.02 and 1.64 to 1.33 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 3 shows, although the normalized completion time achieved by Orpheus drops first, it finally approximates about 1.5 for both L3 and L7 multicast. As a comparison, SELMCAST is able to achieve consistently high performance.

As for the average number of finally selected receivers, SELMCAST outperforms Orpheus about 1.3× on selecting receivers for L7 multicast (Figure 2f); while all schemes would select exactly $p$ receivers to meet the requirements thus yielding no performance gains when L3 multicast is employed




(a) Completion time (L3) (b) Completion time (L7)

Fig. 3: SELMCAST achieves consistent normalized completion times under different $p$ values, while the results of Orpheus decrease and approach 1.5, for both L3 and L7 multicast.

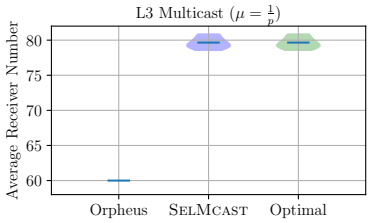



Fig. 4: When $\mu = \frac{1}{p}$, SELMCAST would select about 33% more receivers per worker than Orpheus for L3 multicast.

(Figure 2e). 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 $\mu$ 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 4).

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.

## V. Related Work

In this section, we briefly overview the recent proposals sharing the idea of partial parameter synchronization and refer the readers to [1], [2] for comprehensive surveys.

P-Reduce [7] and Prague [8] achieve partial parameter synchronization by executing the AllReduce collective operations on a selected set of workers. Similarly, Dutta et al. [9] 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. [17] further study the performance of Gossip-based partial synchronization. Different from these works, SELMCAST mainly focuses on optimizing peer-to-peer distributed machine learning. Its design is motivated by the recent work of Orpheus [3], which selects receivers randomly and is the follow-up work of [10] in turn.

Besides the level of participating workers, the idea of partial synchronization could be implemented at the level of parameters as well. For example, papers like [18] show that only delivering the top-$k$ model gradients works well for many training tasks; BTP finds that many algorithms are robust to random gradients drops with bounded amounts [19], and DGT [20] 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 [15].

## 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 datacenter distributed learning. SELMCAST is *Pareto-optimal* and supports both L3 or L7 multicast-based parameter synchronization by design, Extensive simulations indicate that SELMCAST is efficient to achieve near-optimal performance very efficient.

## References


[1] 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.

[2] J. Verbraeken, M. Wolting *et al.*, "A survey on distributed machine learning," *ACM Computing Surveys*, vol. 53, no. 2, Mar. 2020.

[3] P. Xie, J. K. Kim *et al.*, "Orpheus: Efficient distributed machine learning via system and algorithm co-design," in *ACM Symposium on Cloud Computing*, 2018, p. 1–13.

[4] S. Luo, P. Fan *et al.*, "Eliminating communication bottlenecks in cross-device federated learning with in-network processing at the edge," in *IEEE ICC*, 2022.

[5] A. Sapio, M. Canini *et al.*, "Scaling distributed machine learning with in-network aggregation," in *USENIX NSDI*, Apr. 2021, pp. 785–808.

[6] L. Luo, Y. Zhang *et al.*, "Fast synchronization of model updates for collaborative learning in micro-clouds," in *IEEE HPCC*, 2021.

[7] X. Miao, X. Nie *et al.*, "Heterogeneity-aware distributed machine learning training via partial reduce," in *SIGMOD/PODS*, Jun 2021.

[8] Q. Luo, J. He *et al.*, "Prague: High-performance heterogeneity-aware asynchronous decentralized training," in *25th ASPLOS*, Mar 2020.

[9] 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.

[10] P. Xie, J. K. Kim *et al.*, "Lighter-communication distributed machine learning via sufficient factor broadcasting," in *32nd Conference on Uncertainty in Artificial Intelligence (UAI'16)*, 2016, p. 795–804.

[11] H. Li, A. Kadav *et al.*, "Malt: Distributed data-parallelism for existing ml applications," in *10th Euorsys*, 2015.

[12] 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.

[13] S. Luo, H. Xing, and P. Fan, "Softwarized ip multicast in the cloud," *IEEE Network*, vol. 35, no. 6, pp. 233–239, 2021.

[14] 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.

[15] S. Luo, P. Fan *et al.*, "Selective coflow completion for time-sensitive distributed applications with poco," in *49th ICPP*, 2020.

[16] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2021. [Online]. Available: https://www.gurobi.com

[17] I. Hegedűs, G. Danner, and M. Jelasity, "Decentralized learning works: An empirical comparison of gossip learning and federated learning," *J Parallel Distrib Comput*, vol. 148, pp. 109–124, 2021.

[18] S. Shi, Q. Wang *et al.*, "A distributed synchronous sgd algorithm with global top-k sparsification for low bandwidth networks," in *39th ICDCS*, 2019, pp. 2238–2247.

[19] J. Xia, G. Zeng *et al.*, "Rethinking transport layer design for distributed machine learning," in *3rd APNet*, 2019, p. 22–28.

[20] 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.