

# Communication-efficient Decentralized Machine Learning over Heterogeneous Networks

Pan Zhou<sup>1\*</sup>, Qian Lin<sup>2</sup>, Dumitrel Loghin<sup>2</sup>, Beng Chin Ooi<sup>2</sup>, Yuncheng Wu<sup>2</sup>, Hongfang Yu<sup>1</sup>

<sup>1</sup> University of Electronic Science and Technology of China

<sup>2</sup> National University of Singapore

willzhoupan@gmail.com, {linqian, dumitrel, ooibc, wuyc}@comp.nus.edu.sg, yuhf@uestc.edu.cn

**Abstract**—In the last few years, distributed machine learning has been usually executed over heterogeneous networks such as a local area network within a multi-tenant cluster or a wide area network connecting data centers and edge clusters. In these heterogeneous networks, the link speeds among worker nodes vary significantly, making it challenging for state-of-the-art machine learning approaches to perform efficient training. Both centralized and decentralized training approaches suffer from low-speed links. In this paper, we propose a decentralized approach, namely NetMax, that enables worker nodes to communicate via high-speed links and, thus, significantly speed up the training process. NetMax possesses the following novel features. First, it consists of a novel consensus algorithm that allows worker nodes to train model copies on their local dataset asynchronously and exchange information via peer-to-peer communication to synchronize their local copies, instead of a central master node (i.e., parameter server). Second, each worker node selects one peer randomly with a fine-tuned probability to exchange information per iteration. In particular, peers with high-speed links are selected with high probability. Third, the probabilities of selecting peers are designed to minimize the total convergence time. Moreover, we mathematically prove the convergence of NetMax. We evaluate NetMax on heterogeneous cluster networks and show that it achieves speedups of  $3.7\times$ ,  $3.4\times$ , and  $1.9\times$  in comparison with the state-of-the-art decentralized training approaches Prague, Allreduce-SGD, and AD-PSGD, respectively.

**Index Terms**—distributed machine learning, decentralized machine learning, heterogeneous network, communication efficiency

## I. INTRODUCTION

Recently, distributed machine learning has become increasingly popular. The training phase of distributed machine learning is typically executed in a multi-tenant cluster [1, 2, 3, 4] or across data centers and edge clusters [5, 6]. As a result, distributed machine learning faces the emerging problem of communication over heterogeneous networks, where the link speeds among worker nodes are highly different. For machine learning model training in a multi-tenant cluster, one of the main challenges is avoiding resource fragmentation by allocating adequate co-located GPUs for worker nodes [2, 7]. Often, the allocated GPUs are placed across machines or even racks [8]. The link speeds within a machine differ significantly from those across machines or racks due to different capacities of the underlying infrastructure. Moreover, the network

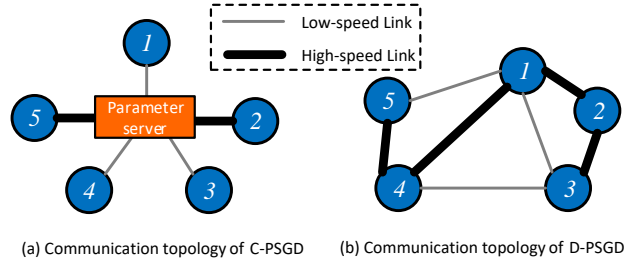


Fig. 1. Communication topology of C-PSGD and D-PSGD. For C-PSGD, worker nodes communicate with a central parameter server to synchronize their training. For D-PSGD, worker nodes communicate with each other in a graph. Both C-PSGD and D-PSGD might be limited by low-speed links.

contention among distributed learning jobs can easily cause network congestion even on a 100 Gbps network, making the difference in link speeds more significant [1, 8]. For model training across data centers or edge clusters, the link speeds of wide area network (WAN) connecting the worker nodes are mainly determined by the geographic distance. For example, the link speed between geographically-close data centers could be up to  $12\times$  faster than between distant ones [9]. In such highly heterogeneous networks, fully utilizing high-speed links is important to reduce the communication cost and thus accelerate the distributed training.

Traditional centralized parallel stochastic gradient descent (C-PSGD) approaches, such as Adam [10] and R2SP [11], require a central node called parameter server (PS) to maintain a global model. At each iteration, all the worker nodes exchange information (i.e., parameters and gradients) with the PS to update the global model, as shown in Fig. 1(a). The training speed of these approaches is typically limited by (1) the network congestion at the central PS, and (2) the low-speed links between the worker nodes and PS since a worker node needs to wait for the updated model from the PS before proceeding to the next iteration. Existing decentralized parallel stochastic gradient descent (D-PSGD) approaches [12, 13, 14, 15, 16, 17] avoid the bottleneck of PS by adopting peer-to-peer communication in a connected graph, as shown in Fig. 1(b). However, they may still suffer from the low-speed links. In synchronous D-PSGD approaches, such as Allreduce-SGD [12] and D2 [13], the worker nodes communicate with their neighbors to exchange information at each iteration, but all the worker nodes need to enter the next iteration simultane-

\*This work was done when this author was a visiting student at National University of Singapore.

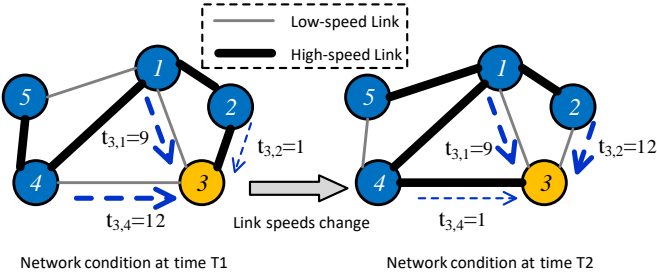


Fig. 2. Heterogeneous and dynamic network condition example.

ously. Thus, the worker nodes with low-speed links will restrict the training process. In asynchronous D-PSGD approaches, such as AD-PSGD [15], GoSGD [16], and Prague [18], a worker node randomly chooses one or several neighbors to communicate with, and starts the next iteration once these communications are completed, independently from the executions of other worker nodes. Nevertheless, the worker node may frequently communicate over low-speed links, resulting in a high communication cost. Take the left side of Fig. 2 as an example, where  $t_{i,j}$  denotes the network latency from node  $j$  to node  $i$ . If node 3 chooses neighbor nodes uniformly random, then in about 67% of its iterations, it pulls models from nodes 1 and 4, but this is time-consuming due to the high network latency. Recently, SAPS-PSGD [19] improves AD-PSGD and GoSGD by letting the worker nodes communicate with each other based on a fixed network subgraph that consists of initially high-speed links. However, the link speeds may dynamically change according to the arrival/leaving of other learning jobs or the network bandwidth reallocation. As a consequence, a link evaluated to be of high-speed at the start of the targeted training may become low-speed at another time. As shown in Fig. 2, if node 3 uses the subgraph of initially high-speed links observed at time T1, then it will only communicate with node 2. However, this link may become low-speed at time T2, resulting in a high communication cost.

In this paper, we aim to answer the following research question: *is it possible to fully use the high-speed links in dynamic heterogeneous networks to accelerate distributed training while providing convergence guarantees in theory?* To tackle this problem, we design a novel communication-efficient decentralized asynchronous distributed machine learning approach (NetMax) consisting of a consensus SGD algorithm and a communication policy generation algorithm. The NetMax approach is also a part of our federated learning [20] system *Falcon*<sup>1</sup>, to improve the communication efficiency during the decentralized training process across data centers located in multiple organizations. NetMax enables worker nodes to communicate preferably through high-speed links in a dynamic heterogeneous network to accelerate the training. In addition to providing guarantees of convergence for the distributed training, NetMax also has the following three novelties. (1) Worker nodes iteratively and asynchronously

train model copies on their local data by using the consensus SGD algorithm. Each worker node randomly selects only one neighbor to communicate with at each iteration. (2) The neighbors with high-speed links are selected with high probabilities, and those with low-speed links are selected with low probabilities. Such a design enables the worker nodes not only to communicate preferably through high-speed links but also to detect the network dynamics. (3) The communication policy generation algorithm updates the communication probabilities for selecting neighbors according to the detected link speeds, which can make the training adapt to network dynamics. Importantly, we analyze the relationship between the convergence of consensus SGD and the communication probabilities. The communication policy generation algorithm is designed such that the total convergence time can be minimized.

The main contributions of this paper are as follows.

- We identify the challenges of distributed learning over a heterogeneous network and propose a communication-efficient asynchronous D-PSGD approach, namely NetMax, to fully utilize fast links and speed up the training.
- We theoretically prove that NetMax provides convergence guarantees for decentralized training.
- We implement NetMax and evaluate it on a heterogeneous multi-tenant cluster. Experimental results show that NetMax achieves up to  $3.7\times$  speedup over state-of-the-art approaches.

The rest of the paper is organized as follows. Section II formulates our decentralized training problem. Section III presents the design of NetMax. Section IV provides the theoretical analysis of NetMax. Section V evaluates NetMax over both homogeneous and heterogeneous networks. Section VI presents related works and Section VII concludes the paper.

## II. PRELIMINARIES

### A. Problem Formulation

We first formulate the problem using the notations in Table I. Let  $M$  denote the number of worker nodes in the network. The communication topology connecting the nodes can be represented as an undirected graph  $\mathcal{G}$  with vertex set  $\mathcal{V} = \{1, 2, \dots, M\}$  and edge set  $\mathcal{E} = \mathcal{V} \times \mathcal{V}$ . Each worker node  $i \in [1, M]$  holds a dataset  $\mathcal{D}_i$ . To collaboratively train a model on the joint dataset, each worker node trains a local model using its own dataset and exchanges the model information with its neighbors in the graph. We formulate such decentralized training as an unconstrained consensus optimization problem:

$$\min_x F(x) = \sum_{i=1}^M \left[ \mathbb{E}[f(x_i; \mathcal{D}_i)] + \frac{\rho}{4} \sum_{m=1}^M d_{i,m} \|x_i - x_m\|^2 \right] \quad (1)$$

where  $x_i$  and  $f(x_i; \mathcal{D}_i)$  are the model parameters and the loss function of worker node  $i$ , respectively.  $d_{i,m}$  is a connection indicator such that  $d_{i,m} = 1$  if the worker nodes  $i$  and  $m$  are neighbors, and  $d_{i,m} = 0$  otherwise.  $\|x_i - x_m\|^2$  measures the model difference between worker node  $i$  and  $m$ , if they are

<sup>1</sup><https://www.comp.nus.edu.sg/~dbsystem/fintech-Falcon/>

TABLE I  
SUMMARY OF NOTATIONS

Notation	Description
$M$	the number of distributed worker nodes
$d_{i,m}$	neighborhood indicator between node $i$ and $m$
$F$	global optimization objective function
$f$	local loss function of worker nodes
$x_i$	local model parameters of node $i$
$\xi_i$	local gradient noise of node $i$
$\rho$	weight of model difference between worker nodes
$\alpha$	SGD learning rate
$k$	the global iteration step
$p_{i,m}$	the probability for node $i$ to select node $m$
$p_i$	the probability that node $i$ communicates with one of its neighbors at a global step
$t_{i,m}$	iteration time for node $i$ communicating with node $m$
$\bar{t}_i$	local average iteration time at node $i$
$\bar{t}$	the global average iteration time

neighbors. The weight  $\rho$  indicates the importance of model difference between neighbors in the objective function. The goal of Eq. (1) is to minimize the loss functions of distributed worker nodes as well as their model differences, ensuring that all nodes will converge to the same optima.

### B. Definitions

**Iteration step.** Let  $n$  be a local iteration step of a worker node, which increases by one when the node communicates with a neighbor. Let  $k$  be a global iteration step, which advances if a local step of any worker node increases. Note that there is only one worker node communicating with a neighbor at any global iteration step.

**Iteration time.** At any local iteration step of worker node  $i$ , let  $C_i$  be the local computation time, and  $N_{i,m}$  be the network communication time between node  $i$  and a neighboring node  $m$ . We denote the iteration time (i.e., the total duration time of this local iteration step) as  $t_{i,m}$ . In this paper, we parallelize the local computation and network communication in each worker node, and thus the iteration time is calculated as

$$t_{i,m} = \max\{C_i, N_{i,m}\}.$$

We observe that the communication time usually dominates. In our experimental setup, detailed in Section V, the iteration time corresponding to inter-machine communication (i.e., via slow links) can be up to  $4\times$  larger compared to the iteration time corresponding to intra-machine communication (i.e., via fast links). Fig.3 demonstrates such comparison measured for the training of two deep learning models, namely ResNet18 and VGG19. Therefore, network communication through a fast link can result in reduced iteration time.

**Probability.** To better utilize high-speed links, we design an adaptive communication strategy that allows a worker node to choose its neighbors with different probabilities, presuming neighbors with high-speed links tend to be selected. The probability distribution may change over time with respect to the dynamic network conditions. This is in contrast to existing decentralized training algorithms such as GoSGD [16] and AD-PSGD [15], which use fixed uniform probability distributions.

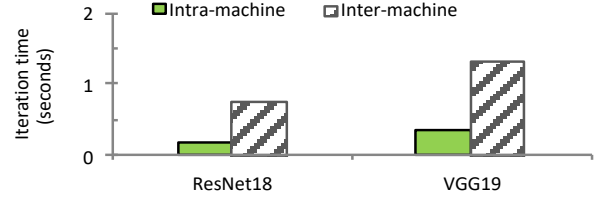


Fig. 3. Average iteration time corresponding to intra-machine (fast) and inter-machine (slow) communication. Nodes are connected by 1000 Mbps Ethernet.

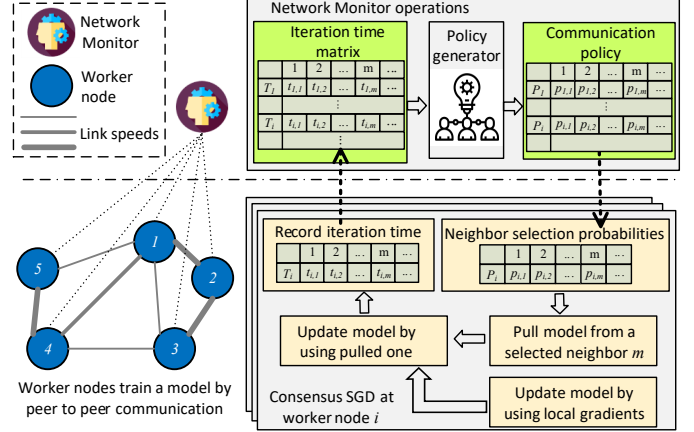


Fig. 4. Overview of the NetMax design.

Let  $\mathbf{P} = [p_{i,m}]_{M \times M}$  be the communication policy matrix (i.e., communication probabilities for worker nodes to select their neighbors), where an entry  $p_{i,m}$  denotes the communication probability for node  $i$  to select node  $m$  as its neighbor. The  $i$ -th row in  $\mathbf{P}$  is the communication probability distribution of node  $i$ . Let  $\bar{t}_i$  denote the average iteration time for node  $i$ ,

$$\bar{t}_i = \sum_{m=1}^M t_{i,m} \cdot p_{i,m} \cdot d_{i,m} \quad (2)$$

Let  $p_i$  denote the probability of worker node  $i$  communicating with one of its neighbors at any global step, which can be derived as

$$p_i = \frac{1/\bar{t}_i}{\sum_{m=1}^M (1/\bar{t}_m)} \quad (3)$$

where  $1/\bar{t}_i$  is the iterative frequency of worker node  $i$ . The larger iterative frequency a worker node has, the higher probability (w.r.t.  $p_i$ ) it can communicate at a global iteration step. According to Eq. (2) and Eq. (3),  $p_i$  is a function of  $p_{i,m}$ . Thus, the design of the communication policy affects the communication frequencies of the worker nodes.

## III. APPROACH

In this section, we present our communication-efficient asynchronous decentralized machine learning approach, termed NetMax.

## A. Overview

Fig.4 illustrates the overall design of NetMax. The worker nodes collaboratively train a model on their local data in a decentralized manner. To adapt to the dynamic network environment, worker nodes rely on a central Network Monitor to track the network status so that they can optimize the utilization of high-speed links to accelerate the training process. One noteworthy aspect is that the central Network Monitor of NetMax does not collect any worker node's training data or model parameters; instead, it only collects a small amount of time-related statistics for evaluating the network condition. Therefore, NetMax is different from existing centralized training methods (e.g., Adam [10] and R2SP [11]), where the central server is responsible for synchronizing the worker nodes and aggregating model parameters from them. This is much likely to become a bottleneck, as discussed in Section VI.

**Network Monitor.** In order to estimate the network condition, the Network Monitor periodically collects the iteration time of each worker node, as this information reflects the link speed between worker nodes (see Section II-B). Let  $T_s$  be the collection period, which can be adjusted according to the network conditions. For example, if the link speeds change quickly,  $T_s$  can be set to a small value to react to the network changes. Algorithm 1 presents the computations done by the Network Monitor. In every period, it collects the time statistics from the worker nodes (lines 3-4), and computes a communication policy matrix  $\mathbf{P}$  and a weight  $\rho$  based on the statistics (line 5). The entries of the matrix  $\mathbf{P}$  are the probabilities for worker nodes to select neighbors, while the weight  $\rho$  defines the importance of the model difference between worker nodes (see Eq. 1). The policy  $\mathbf{P}$  and weight  $\rho$  are adaptively tuned to make the whole training process converge fast. The details of policy design will be presented in Section III-C. Finally, the Network Monitor sends the policy matrix  $\mathbf{P}$  and weight  $\rho$  back to the worker nodes (line 6). The Network Monitor uses the iteration time  $t_{i,m}$  to evaluate the link speed between worker nodes  $i$  and  $m$ , rather than measuring the exact speed. The rationale for doing so is twofold. First, the iteration time reflects well the corresponding link speed. A short iteration time means short communication time as well as fast link speed. Second, the iteration time can be directly measured by the worker nodes during their training, which avoids injecting a large amount of traffic into the network to measure the exact link speed.

**Worker Nodes.** Each worker node  $i$  conducts a consensus SGD algorithm to train a model replica on its local data, as depicted in Algorithm 2. In each iteration, a worker node updates its local model in two steps. First, it updates the model based on its computed gradients. Second, it selects one of its neighbors (e.g., node  $m$ ) according to the communication policy and pulls model parameters from the neighbor to update its local model again. Meanwhile, it maintains an average iteration time  $t_{i,m}$  for the communication with neighbor  $m$ . When receiving a request from the Network Monitor, the

---

## Algorithm 1 NetMax at Network Monitor

---

**Require:** schedule period  $T_s$ , learning rate  $\alpha$ , out-loop search round  $K$ , inner-loop search round  $R$ .

```

1: Initialize iteration time matrix  $[t_{i,m}]_{M \times M} \leftarrow [0]_{M \times M}$ 
2: while TRUE do
3:   Send iteration-time request to worker nodes
4:    $[t_{i,m}]_{M \times M} \leftarrow$  Receive the iteration time
5:    $\mathbf{P}, \rho \leftarrow \text{GENERATEPOLICYMATRIX}(\alpha, K, R, [t_{i,m}]_{M \times M})$ 
6:   Send  $\mathbf{P}, \rho$  to worker nodes
7:    $\text{Sleep}(T_s)$ 
8: end while
```

---

iteration time is sent for computing the new communication policy. After receiving the new policy from the Network Monitor, the worker updates its local policy accordingly.

## B. The Consensus SGD Algorithm

As our decentralized training optimization problem formulated by Eq. (1) is unconstrained, we can solve it by the standard stochastic gradient descent (SGD) method, where the gradients at worker node  $i$  can be derived as

$$\nabla F(x_i; \mathcal{D}_i) = \nabla f(x_i; \mathcal{D}_i) + \frac{\rho}{2} \sum_{m=1}^M (d_{i,m} + d_{m,i})(x_i - x_m) \quad (4)$$

Then the parameters at worker node  $i$  can be updated by

$$x_i^{n+1} = x_i^n - \alpha \nabla F(x_i^n; \mathcal{D}_{i,n}) \quad (5)$$

where  $x_i^n$  are the model parameters and  $\mathcal{D}_{i,n}$  is the sampled dataset from  $\mathcal{D}_i$  of worker node  $i$  at local iteration step  $n$ . Based on Eq. (4) and (5), the exchanged information between neighbors consists of their model parameters. However, simply applying the update rule in Eq. (5) requires a worker node to pull parameters from all its neighbors, which means that the training process might be limited by the low-speed links.

To reduce communication time, we propose a consensus SGD algorithm. At each iteration, a worker node only pulls model parameters from a randomly selected neighbor. The neighbors with high-speed links are more likely to be selected. As training proceeds, each worker node's local information will be propagated in the communication graph and gradually reach other worker nodes. Meanwhile, each worker node uses both local gradients and the model from its neighbor to update its model. Therefore, both the loss functions of worker nodes and the model differences between them can be minimized in the training process. Consequently, the worker nodes converge to the same optima.

The detailed executions of a worker node are presented in Algorithm 2. At the beginning of each iteration, a worker node  $i$  first updates its communication probabilities if a new policy is received from the Network Monitor (lines 5-8). Then it randomly selects a neighbor  $m$  with probability  $p_{i,m}$  and requests the freshest parameters from it (lines 9-10). A neighbor with a high-speed link has a higher probability of being selected. Therefore, the overall communication time can be reduced.

After sending the request, worker node  $i$  continues to compute gradients using its local dataset and updates its model using these gradients (line 11). Meanwhile, it keeps checking whether the requested parameters are returned (line 12). When receiving the response from neighbor  $m$ , it updates the model again using the received parameters (lines 13-15). This two-step update overlaps the gradient computation and parameter communication, which further reduces the training time. Moreover, if a neighbor is chosen with a lower probability, the model update rule can assign a higher weight to the pulled model from that neighbor, which enables the nodes to maintain enough information from neighbors chosen with low probabilities.

At the end of the iteration, worker node  $i$  updates the average iteration time corresponding to the communication with neighbor  $m$  (line 16). Specifically, we adopt the exponential moving average (EMA) [21] to maintain the iteration time (lines 19-22). This is because (1) we do not need to store historical time data, and (2) the window length of moving average can be tuned by changing the smoothing factor  $\beta$ , where a smaller  $\beta$  means a shorter window. The value of  $\beta$  can be adapted to the network dynamics. When the link speeds change quickly,  $\beta$  should be set to a low value to capture the latest speeds. At the same time, the period of policy computing at the Network Monitor,  $T_s$ , should be short too. Then the Network Monitor can quickly update the communication policy for worker nodes, and the entire system adapts to dynamic network conditions.

### C. Communication Policy Generation

The communication policy is derived from solving an optimization problem based on the convergence analysis of the proposed decentralized training approach. The training on the worker nodes can be written in matrix form as

$$\mathbf{x}^{k+1} = \mathbf{D}^k(\mathbf{x}^k - \alpha \mathbf{g}^k) \quad (6)$$

where  $\mathbf{x}^k$  and  $\mathbf{g}^k$  are the vectors containing all worker nodes' model parameters and gradients at global step  $k$ , respectively.  $\mathbf{D}^k$  is a random matrix at global step  $k$  related to the communication policy  $\mathbf{P}$ . The models' deviation from the optima  $\mathbf{x}^*$  can be bounded as follows,

$$\mathbb{E}[\|\mathbf{x}^k - \mathbf{x}^* \mathbf{1}\|^2] \leq \lambda^k \|\mathbf{x}^0 - \mathbf{x}^* \mathbf{1}\|^2 + \alpha^2 \sigma^2 \frac{\lambda}{1 - \lambda} \quad (7)$$

where  $\mathbf{1}$  denotes a vector with entries equal to 1 and  $\lambda$  is the second largest eigenvalue of matrix  $\mathbb{E}[(\mathbf{D}^k)^T \mathbf{D}^k]$ . The detailed derivation of Eq. (6) and (7) is presented in Section IV.

A noteworthy aspect is that most existing decentralized machine learning algorithms, such as Gossiping SGD [22] and GoSGD [16], can be formulated in the form of Eq. (6). Their models' deviation from the optima can also be bounded by Eq. (7). In these algorithms, the worker nodes choose their neighbors uniformly at random, resulting in the fastest convergence rate over a homogeneous network [23]. In other words, they have small global iteration steps  $k$  to convergence.

---

### Algorithm 2 NetMax at worker node $i$

---

**Require:** learning rate  $\alpha$ , weight  $\rho$ , iteration number  $N$ , smoothing factor  $\beta$ .

**Ensure:** Trained model  $x_i$

```

1: Initialize model  $x_i^0$ 
2: Initialize probabilities  $[p_{i,1}, p_{i,2}, \dots, p_{i,M}] \leftarrow [1/M]_{1 \times M}$ 
3: Initialize iteration time vector  $\mathbf{T}_i \leftarrow [0]_{1 \times M}$ 
4: for  $n \in \{1, 2, \dots, N\}$  do
5:   if received new policy then
6:      $[p_{i,1}, p_{i,2}, \dots, p_{i,M}] \leftarrow \text{recv.P}[i]$ 
7:      $\rho \leftarrow \text{recv.}\rho$ 
8:   end if
9:   Random select a neighbor  $m$  with probability  $p_{i,m}$ 
10:  Request parameter  $x_m$  from worker node  $m$ 
11:   $x_i^n \leftarrow x_i^n - \alpha \nabla f(x_i^n; \mathcal{D}_{i,n})$   $\triangleright$  First step update
12:  Wait parameter  $x_m$ 
13:   $\theta_i^n \leftarrow \frac{\rho}{2} \frac{d_{i,m} + d_{m,i}}{p_{i,m}} (x_i^n - x_m)$ 
14:   $x_i^n \leftarrow x_i^n - \alpha \theta_i^n$   $\triangleright$  Second step update
15:   $x_i^{n+1} \leftarrow x_i^n$   $\triangleright$  Store parameter for next iteration
16:  UPDATETIMEVECTOR
17: end for
18: return  $x_i$ 
19: procedure UPDATETIMEVECTOR
20:    $t_{i,m} \leftarrow$  Recorded iteration time
21:    $\mathbf{T}_i[m] \leftarrow \beta \mathbf{T}_i[m] + (1 - \beta)t_{i,m}$ 
22: end procedure
```

---

However, in a heterogeneous network, the average global step time  $\bar{t}$  can be very long since the worker nodes frequently choose neighbors with low-speed links. As a result, the total time to convergence,  $k\bar{t}$ , is very long. In contrast, choosing high-speed-link neighbors with high probability could reduce the average iteration time  $\bar{t}$ , but may lead to a slow convergence rate (i.e., larger global iteration steps  $k$  to convergence). Therefore, there exists a trade-off between fast convergence rate and short iteration time to minimize the total time  $k\bar{t}$  to convergence.

To obtain the communication policy that efficiently reduces the convergence time, we formulate and solve an optimization problem. Our objective is to minimize the total convergence time  $k\bar{t}$ , as shown in Eq. (8).

$$\min \quad k\bar{t} \quad (8)$$

s.t.

$$\lambda^k \leq \varepsilon, \forall \varepsilon > 0 \quad (9)$$

$$\bar{t} = \frac{\bar{t}_i}{M} = \frac{1}{M} \sum_{m=1}^M t_{i,m} p_{i,m} d_{i,m}, \forall i \in [M] \quad (10)$$

$$p_{i,m} > \alpha \rho (d_{i,m} + d_{m,i}), \forall i, m \in [M], m \neq i, d_{i,m} \neq 0 \quad (11)$$

$$p_{i,m} = 0, \forall i, m \in [M], d_{i,m} = 0 \quad (12)$$

$$\sum_{m=1}^M p_{i,m} = 1, \forall i \in [M] \quad (13)$$

**Algorithm 3** Communication Policy Generation

**Require:** learning rate  $\alpha$ , outer-loop search round  $K$ , inner-loop search round  $R$ , iteration time matrix  $\mathbf{T} = [t_{i,m}]_{M \times M}$ .

**Ensure:** Probability matrix  $\mathbf{P}$

```

1: function GENERATEPOLICYMATRIX( $\alpha, K, R, \mathbf{T}$ )
2:    $L_\rho \leftarrow 0.0$ 
3:    $U_\rho \leftarrow 0.5/\alpha$ 
4:    $\delta_\rho \leftarrow (U_\rho - L_\rho)/K$ ;
5:   for  $k \in \{1, 2, \dots, K\}$  do
6:      $\rho \leftarrow L_\rho + \delta_\rho$ 
7:      $\mathbf{P}, T_{\text{convergence}} \leftarrow \text{INNERLOOP}(\alpha, \rho, R, \mathbf{T})$ 
8:      $\text{out}[k] \leftarrow (\mathbf{P}, T_{\text{convergence}}, \rho)$ 
9:   end for
10:   $y \leftarrow \text{find item in out with minimum } T_{\text{convergence}}$ 
11:  return  $y, \mathbf{P}, y, \rho$ 
12: end function
13: function INNERLOOP( $\alpha, \rho, R, \mathbf{T}$ )
14:   $L \leftarrow \max_{i \in [M]} \{ \frac{\alpha \rho}{M} \sum_{m=1}^M t_{i,m} (d_{i,m} + dm, i) \}$ ;
15:   $U \leftarrow \min_{i \in [M]} \{ \frac{1}{M} \max_{m \in [M]} \{ t_{i,m} d_{i,m} \} \}$ ;
16:   $\delta \leftarrow (U - L)/R$ ;
17:  for  $r \in \{1, 2, \dots, R\}$  do
18:     $\bar{t} \leftarrow L + \delta$ 
19:     $\mathbf{P} \leftarrow \text{Solve LP by using standard method}$ 
20:     $\lambda_2 \leftarrow \text{Compute second largest eigenvalue for } \mathbf{Y}_\mathbf{P}$ 
21:     $T_{\text{convergence}} \leftarrow \bar{t} \frac{\ln \varepsilon}{\ln \lambda_2}$ 
22:     $\text{res}[r] \leftarrow (T_{\text{convergence}}, \lambda_2, \bar{t}, \mathbf{P})$ 
23:  end for
24:   $x \leftarrow \text{find item in res with minimum } T_{\text{convergence}}$ 
25:  return  $x, \mathbf{P}, x, T_{\text{convergence}}$ 
26: end function

```

From Eq. (7) we know that when  $\lambda^k$  is less than a very small positive value  $\varepsilon$ , the models of worker nodes converge to a very small domain near the optimal model. Therefore,  $k$  can be constrained by  $\lambda$ , as shown in Eq. (9), where a small  $\lambda$  means a small convergence iteration  $k$  and, thus, a fast convergence rate. Constraint Eq. (10) and (11) ensure that the matrix  $\mathbb{E}[(\mathbf{D}^k)^T \mathbf{D}^k]$  is a doubly stochastic symmetric matrix and its  $\lambda$  is strictly less than 1. Consequently, constraint Eq. (9) always holds. Constraint Eq. (10) defines the global average iteration time  $\bar{t}$ , where  $\bar{t}_i$  is defined in Eq. (2). Eq. (10) is derived by letting the rows of  $\mathbb{E}[(\mathbf{D}^k)^T \mathbf{D}^k]$  sum to 1. Constraint Eq. (11) is derived by letting the entry  $y_{i,m}$  of  $\mathbb{E}[(\mathbf{D}^k)^T \mathbf{D}^k]$  to be larger than 0, for all  $d_{i,m} \neq 0$ , which determines the smallest communication probabilities between neighbors. These smallest communication probabilities are proportional to the adjustable coefficient  $\rho$  given learning rate  $\alpha$  and communication graph  $\mathcal{G}$ . Constraints Eq. (12) and (13) naturally hold.

Eq. (8)-(13) represent a nonlinear programming problem involving the computation of eigenvalues, which is hard to be solved by standard methods. Instead of finding its optimal solution, we propose a faster method to find a feasible sub-optimal solution. Our observation is that when given  $\rho$  and  $\bar{t}$ , there is a feasible policy  $\mathbf{P}$  that minimizes the objective value,  $k\bar{t}$ . Hence, our key idea is to search for a policy  $\mathbf{P}$  that has the smallest objective value under various feasible configurations of  $\rho$  and  $\bar{t}$ .

Algorithm 3 describes our method to find a feasible policy,

which consists of two nested loops. The outer-loop searches for  $K$  values of  $\rho$  within its feasible interval  $[L_\rho, U_\rho]$  (lines 1-6). For any given  $\rho$ , we use an inner-loop to obtain a sub-optimal matrix  $\mathbf{P}$  (line 7). At the end, we return the matrix  $\mathbf{P}$  with the smallest objective value (lines 10-11). The inner-loop searches for  $R$  values of  $\bar{t}$  within its feasible interval  $[L, U]$  when  $\rho$  is given (lines 14-18). For each value of  $\bar{t}$  in the inner-loop, we solve a linear programming (LP) problem defined in Eq. (14) to obtain a sub-optimal matrix  $\mathbf{P}$  (lines 19-22).

$$(LP) \min \sum_{i=1}^M p_{i,i} \text{ s.t. } Eq.(10), Eq.(11), Eq.(12), Eq.(13) \quad (14)$$

The LP problem in Eq. (14) seeks to minimize the probabilities of distributed workers to select themselves. Such a design encourages the workers to exchange information with their neighbors and, thus, achieve a high convergence rate. The derivation of the feasible intervals for  $\rho$  and  $\bar{t}$  can be found in Appendix A. Algorithm 3 can solve the problem stated in Eq. (8)-(13) with an approximation ratio of  $\frac{U}{L} \frac{\ln(M-1) - \ln(M-3)}{\ln(1-2a+a^M) - \ln(1-2a+a^{M+1})}$  when the graph connecting the worker nodes is fully-connected, the network is heterogeneous, and where  $a$  denotes the minimum positive entry of  $\mathbb{E}[(\mathbf{D}^k)^T \mathbf{D}^k]$ . The proof can be found in Appendix B.

#### D. Extension to Existing Decentralized PSGD Approaches

A noteworthy aspect is that the standalone Network Monitor of NetMax can be used to improve existing D-PSGD approaches (e.g., Gossiping SGD [22] and AD-PSGD [15]) as long as they can be re-written as the form of Eq. (6). Specifically, the extension includes two steps. First, we can derive a similar constraint to replace Eq. (11) by requiring each entry  $y_{i,m}$  of  $\mathbb{E}[(\mathbf{D}^k)^T \mathbf{D}^k]$  to be larger than 0, for all  $d_{i,m} \neq 0$ . Second, the corresponding probabilities for worker nodes to choose neighbors can be obtained by solving the optimization problem Eq. (8)-(13). By doing so, existing D-PSGD approaches can be enhanced to adapt to the dynamics of heterogeneous networks.

#### IV. CONVERGENCE ANALYSIS

In this section, we present the convergence analysis of our proposed approach. More specifically, we prove that the models of distributed worker nodes converge to the same optima under Assumption 1.

**Assumption 1.** We make the following commonly used assumptions for this analysis:

- **Connected graph.** The undirected graph  $\mathcal{G}$  connecting worker nodes is connected.
- **Lipschitzian gradient.** The local loss function  $f$  is a  $\mu$ -strongly convex with  $L$ -Lipschitz gradients.
- **Bounded gradient variance.** For any worker node  $i$ , its gradient can be bounded as  $\mathbb{E}[\nabla f(x_i^k)^T \nabla f(x_i^k)] \leq \eta^2$ .
- **Additive noise.** For any worker node  $i$ , its stochastic gradients have additive noise  $\xi_i$  caused by stochastic sampling from the dataset.  $\xi_i$  is with zero mean  $\mathbb{E}[\xi_i] = 0$  and bounded variance  $\mathbb{E}[\xi_i^T \xi_i] \leq \sigma^2$ .

For simplicity, let  $\gamma_{i,m} = \frac{d_{i,m} + d_{m,i}}{2p_{i,m}}$  and  $g_i^k = \nabla f(x_i; \mathcal{D}_i) + \xi_i^k$ , where  $\xi_i^k$  is the additive noise at global step  $k$ .

At global step  $k$ , worker node  $i$  pulls parameters from its neighbor  $m$  with probability  $p_{i,m}$ . According to Algorithm 2, the first update of worker node  $i$  using local gradients (line 11) can be written as

$$x_i^{k+\frac{1}{2}} = x_i^k - \alpha g_i^k \quad (15)$$

The second update of worker node  $i$  using neighbor's parameters (lines 13-15) can be written as

$$\begin{aligned} x_i^{k+1} &= x_i^{k+\frac{1}{2}} - \alpha \rho \gamma_{i,m} (x_i^{k+\frac{1}{2}} - x_m^k) \\ &= (1 - \alpha \rho \gamma_{i,m}) (x_i^k - \alpha g_i^k) + \alpha \rho \gamma_{i,m} x_m^k \end{aligned} \quad (16)$$

For any worker node  $j \neq i$ ,  $x_j^{k+1} = x_j^k$ . Thus, at each global step  $k$ , we have

$$\begin{cases} x_i^{k+1} = (1 - \alpha \rho \gamma_{i,m}) (x_i^k - \alpha g_i^k) + \alpha \rho \gamma_{i,m} x_m^k \\ x_j^{k+1} = x_j^k, \forall j \in [M], j \neq i \end{cases} \quad (17)$$

We can rewrite Eq. (17) in matrix form as

$$\mathbf{x}^{k+1} = \mathbf{D}^k (\mathbf{x}^k - \alpha \mathbf{g}^k) \quad (18)$$

where  $\mathbf{x}^k = [x_1^k, \dots, x_i^k, \dots, x_m^k, \dots, x_M^k]^T$  is a vector containing all worker nodes' models at global step  $k$ ,  $\mathbf{g}^k = [0, \dots, g_i^k, \dots, 0, \dots, 0]^T$  is a vector containing all worker nodes' gradients at global step  $k$  and  $\mathbf{D}^k$  is an  $M \times M$  matrix, which is expressed as

$$\mathbf{D}^k = \mathbf{I} + \alpha \rho \gamma_{i,m} \mathbf{e}_i (\mathbf{e}_m - \mathbf{e}_i)^T \quad (19)$$

where  $\mathbf{e}_i = [0, \dots, 1, \dots, 0, \dots, 0]^T$  is an  $M \times 1$  unit vector with  $i$ -th component equal to 1. Note that index  $i$  is a random variable drawn from  $\{1, 2, \dots, M\}$  with probability  $p_i$  and  $m$  is a random variable drawn from  $\{1, 2, \dots, M\}$  with probability  $p_{i,m}$ . The expectation below is derived with respect to the random variables  $i$  and  $m$ :

$$\begin{aligned} \mathbf{Y} &= \mathbb{E}[(\mathbf{D}^k)^T \mathbf{D}^k] \\ &= \mathbb{E}[\mathbf{I} + \alpha \rho \gamma_{i,m} \mathbf{e}_i (\mathbf{e}_m - \mathbf{e}_i)^T + \alpha \rho \gamma_{i,m} [\mathbf{e}_i (\mathbf{e}_m - \mathbf{e}_i)^T]^T \\ &\quad + \alpha^2 \rho^2 \gamma_{i,m}^2 [\mathbf{e}_i (\mathbf{e}_m - \mathbf{e}_i)^T]^T \mathbf{e}_i (\mathbf{e}_m - \mathbf{e}_i)^T] \\ &= [\mathbf{y}_{i,m}]_{M \times M} \end{aligned} \quad (20)$$

where

$$\begin{cases} y_{i,i} = 1 - 2\alpha \rho \sum_{\substack{m \in [M] \\ m \neq i}} p_i p_{i,m} \gamma_{i,m} \\ \quad + \alpha^2 \rho^2 \sum_{\substack{m \in [M] \\ m \neq i}} (p_i p_{i,m} \gamma_{i,m}^2 + p_m p_{m,i} \gamma_{m,i}^2), \forall i \in [M] \\ y_{i,m} = \alpha \rho (p_i p_{i,m} \gamma_{i,m} + p_m p_{m,i} \gamma_{m,i}) \\ \quad - \alpha^2 \rho^2 (p_i p_{i,m} \gamma_{i,m}^2 + p_m p_{m,i} \gamma_{m,i}^2), \forall i, m \in [M], m \neq i \end{cases} \quad (22)$$

The matrix  $\mathbf{Y}$  is associated with the communication policy matrix  $\mathbf{P}$ , denoted by  $\mathbf{Y}_{\mathbf{P}}$ . Let  $\lambda_1$  and  $\lambda_2$  be the largest and second largest eigenvalue of matrix  $\mathbf{Y}_{\mathbf{P}}$ , respectively. Let  $\lambda =$

$\lambda_2$  if  $\mathbf{Y}_{\mathbf{P}}$  is a doubly stochastic matrix; otherwise let  $\lambda = \lambda_1$ . Then we can derive the following two theorems under static and dynamic networks. The proofs can be found in Appendix C and Appendix D, respectively.

**Theorem 1.** (static network) *If Assumption 1 is satisfied and the applied decentralized learning algorithm can be written in the form of Eq. (18) with learning rate  $0 < \alpha \leq \frac{2}{\mu+L}$ , the squared sum of the local model parameters' deviation from the optimal  $\mathbf{x}^*$  can be bounded as*

$$\mathbb{E}[\|\mathbf{x}^k - \mathbf{x}^* \mathbf{1}\|^2] \leq \lambda^k \|\mathbf{x}^0 - \mathbf{x}^* \mathbf{1}\|^2 + \alpha^2 \sigma^2 \frac{\lambda}{1 - \lambda} \quad (23)$$

The decentralized training will converge to a small domain and achieve consensus if  $\lambda < 1$ .

In a dynamic network, when network conditions change, so does  $\lambda$ . Let  $\lambda_{max}$  be the maximum among historical eigenvalues  $\lambda$ . Then we can formulate Theorem 2.

**Theorem 2.** (dynamic network) *If Assumption 1 is satisfied and the applied decentralized learning algorithm can be written in the form of Eq. (18) with a learning rate  $0 < \alpha \leq \frac{2}{\mu+L}$ , the squared sum of the local model parameters' deviation from the optimal  $\mathbf{x}^*$  can be bounded as*

$$\mathbb{E}[\|\mathbf{x}^k - \mathbf{x}^* \mathbf{1}\|^2] \leq \lambda_{max}^k \|\mathbf{x}^0 - \mathbf{x}^* \mathbf{1}\|^2 + \alpha^2 \sigma^2 \frac{\lambda_{max}}{1 - \lambda_{max}} \quad (24)$$

The decentralized training will converge to a small domain and achieve consensus if  $\lambda_{max} < 1$ .

From Theorem 1 and Theorem 2, we obtain that the communication policy can determine the eigenvalues of  $\mathbf{Y}_{\mathbf{P}}$  under any given network conditions, and, thus, determine the convergence of the training. The convergence rate can be bounded by  $\lambda^k$  and  $\lambda_{max}^k$  under static and dynamic networks, respectively. Moreover, we can formulate the following theorem.

**Theorem 3.** *For any feasible solution  $\mathbf{P}$  that is obtained by Algorithm 3,  $\mathbf{Y}_{\mathbf{P}}$  is a doubly stochastic matrix and the second largest eigenvalue  $\lambda$  of matrix  $\mathbf{Y}_{\mathbf{P}}$  is strictly less than 1. Hence, the decentralized training converges. Moreover, for a chosen learning rate  $\alpha = \frac{c}{\sqrt{k}}$  where  $c$  is a pre-defined positive number, the distributed learning converges with a rate  $O(1/\sqrt{k})$ .*

Any feasible solution of the optimization problem in Eq. (8)-(13) ensures that the training converges. As the policy  $\mathbf{P}$  obtained by Algorithm 3 is always a feasible solution for the optimization problem in Eq. (8)-(13), the training performed by our approach always converges. The detailed proof of Theorem 3 is presented in Appendix E.

## V. PERFORMANCE EVALUATION

We have implemented NetMax in Python on top of PyTorch. In this section, we conduct experiments to evaluate NetMax against three state-of-the-art decentralized machine learning (ML) techniques: Allreduce-SGD [12], AD-PSGD [15] and Prague [18]. Note that, for a fair comparison, all the above implementations share the same runtime environment.

### A. Experimental Setup

**Cluster.** Experiments are conducted in a multi-tenant cluster consisting of 18 servers connected via 1000 Mbps Ethernet. Each server is equipped with an Intel(R) Xeon(R) W-2133 CPU @ 3.60GHz, 65 GB DDR3 RAM, and 3 or 8 GPUs of GeForce RTX 2080 Ti. Worker nodes and Network Monitor of NetMax are run in docker containers, each of worker nodes is assigned with an exclusive GPU. Due to the limited number of GPUs available on each server, we run 4, 8 and 16 worker nodes across 2, 3 and 4 servers, respectively. Unless otherwise specified, the number of worker nodes is 8 by default throughout the experiments.

**Network.** Some links among worker nodes are purposely slowed down in order to emulate a dynamic heterogeneous network. In particular, by following [15], we randomly slow down one of the communication links among nodes by 2x to 100x. Other than that, we further change the slow link every 5 minutes. As the processing time of distributed ML jobs varies from tens of minutes to several days [2, 15], the above frequency of link speed change is appropriate to mimic the dynamic arriving/leaving of distributed ML jobs in a multi-tenant cluster.

On the other hand, to emulate a homogeneous network, we reserve a server with 8 GPUs and scale the number of worker nodes running within the server from 4 to 8. The worker nodes are connected via a virtual switch [24] with 10 Gbps bandwidth, which implements a homogeneous peer-to-peer communication.

**Datasets and models.** We evaluate the experiments using five real-world datasets: MNIST, CIFAR10, CIFAR100, Tiny-ImageNet, and ImageNet. We use four models in the experiments: MobileNet, ResNet18, ResNet50, and VGG19 whose numbers of parameters are approximately 4.2M, 11.7M, 25.6M, and 143.7M, respectively.

**Configurations.** The hyper-parameter settings as applied in both AD-PSGD [15] and Prague [18] are adopted in our experimental study. Specifically, unless noted otherwise, the models are trained with *batch size* 128, *momentum* 0.9, and *weight decay*  $10^{-4}$ . The *learning rate* starts from 0.1 and decays by a factor of 10 once the loss does not decrease any more. We train ResNet18 and VGG19 with *epoch numbers* of 64 and 82 respectively. For NetMax, we alter the communication policy every  $T_s = 2$  minutes, as such a period is long enough for the detection of link speed change at runtime.

### B. Performance of Decentralized Training

Fig. 5 presents the computation cost and communication cost of the average epoch time of the four approaches under the heterogeneous network setting. As can be seen, the computation costs of all the approaches are almost the same. This is expected since they train the same models under the same runtime environment. However, the communication costs differ significantly. In particular, NetMax incurs the lowest communication cost, mainly due to its improved utilization of high-speed links. For ResNet18, NetMax reduces the communication cost by up to 83.4%, 81.7% and 63.7% comparing

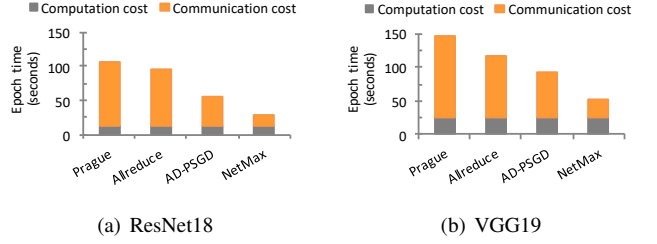


Fig. 5. Average epoch time with 8 worker nodes in a heterogeneous network.

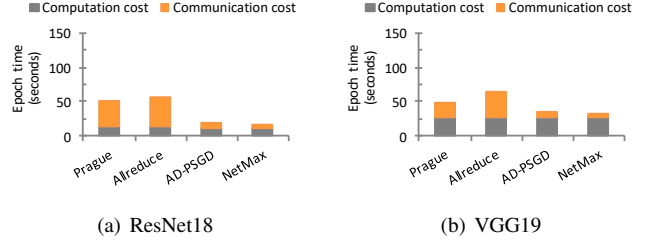


Fig. 6. Average epoch time with 8 worker nodes in a homogeneous network.

with Prague, Allreduce-SGD, and AD-PSGD, respectively. For VGG19, NetMax reduces the communication cost by up to 76.8%, 69.2% and 58.6%, correspondingly. In contrast, Prague suffers from the highest communication cost. This is because Prague divides the worker nodes into multiple groups in every iteration, and each group executes a *partial-allreduce* operation to average the models. The concurrent executions of partial-allreduce of different groups compete for the limited bandwidth capacity, resulting in network congestion. Moreover, the partial-allreduce operation is agnostic to the link speed. As a consequence, the heavy use of low-speed links further deteriorates its performance.

We repeat the above experiments but alternatively under the homogeneous network setting, and the results are shown in Fig. 6. As can be seen, the computation costs of the four approaches remain similar, whereas the communication costs are fairly lower comparing with Fig. 5. This is due to the expanded bandwidth used in the homogeneous network. Overall, NetMax and AD-PSGD are superior in low communication cost over Prague and Allreduce-SGD. This is because Allreduce-SGD and Prague average gradients or model parameters from multiple worker nodes in each iteration, which introduces extra rounds of communication between worker nodes and thus results in high communication cost. In contrast, each worker node in NetMax and AD-PSGD pulls the model updates from only one neighbor and immediately starts the next iteration once that communication completes, leading to the reduction of communication cost.

### C. Source of Performance Improvement in NetMax

Two strategies are implemented in NetMax to reduce the epoch time, as elaborated in Algorithm 2. First, the gradient computation and network communication are executed in parallel. Second, each worker node selects its neighbors based on the adaptive probabilities. In this subsection, we evaluate the performance improvement brought about by these two

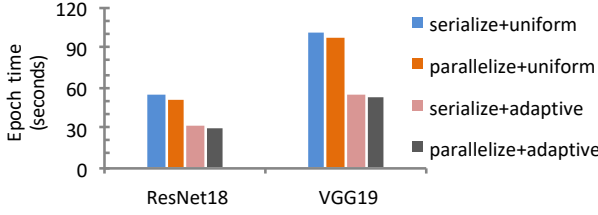


Fig. 7. Average epoch time w.r.t. the four settings.

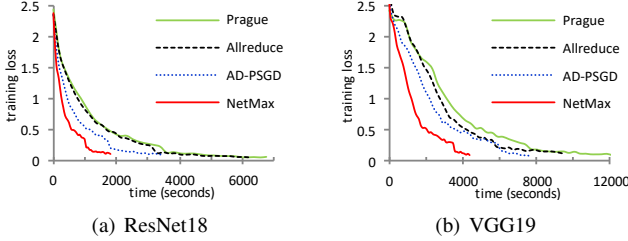


Fig. 8. Training loss with 8 worker nodes in a heterogeneous network.

strategies when training the ResNet18 and VGG19 models on the CIFAR10 dataset. In particular, we analyze the following four settings.

- Setting 1: serial execution (of gradient computation and network communication) + uniform probabilities (for selecting neighbors). This represents the baseline.
- Setting 2: parallel execution + uniform probabilities. This is to verify the effect of parallelism.
- Setting 3: serial execution + adaptive probabilities. This is to verify the effect of adaptive probabilities.
- Setting 4: parallel execution + adaptive probabilities. This represents the full functioning NetMax.

As shown in Fig. 7, the use of adaptive probabilities contributes to the majority of performance gain in NetMax. For example, when comparing serial+uniform with serial+adaptive, we observe that the average epoch times are reduced from 54s and 100.5s to 30.3s and 55.4s for ResNet18 and VGG19, respectively. On the other hand, the performance improvement due to parallelization is marginal because the time of gradient computation on GPUs is much shorter than the time of network communication.

#### D. Convergence and Accuracy

Fig. 8 shows the comparison of training loss with respect to training time under the heterogeneous network setting. As can be seen, NetMax converges much faster than the other approaches. For ResNet18, NetMax achieves about  $3.7\times$ ,  $3.4\times$  and  $1.9\times$  speedup over Prague, Allreduce-SGD and AD-PSGD, respectively. For VGG19, the speedup is about  $2.8\times$ ,  $2.2\times$  and  $1.7\times$ , correspondingly. Such performance gain is rooted at NetMax's capability such that it can adapt the probabilities for neighbor selection with respect to the change of link speed to fine-tune the communication frequencies between worker node pairs (referring to the policy design in Section III-C). This not only allows the worker nodes to communicate preferably over high-speed links, but also

TABLE II  
ACCURACY OF MODELS TRAINED OVER A HETEROGENEOUS NETWORK

		Prague	Allreduce	AD-PSGD	NetMax
ResNet18	4 nodes	89.56%	90.11%	90.24%	90.46%
	8 nodes	90.54%	90.16%	90.39%	91.14%
	16 nodes	90.44%	90.37%	90.23%	90.78%
VGG19	4 nodes	90.09%	89.98%	89.8%	90.24%
	8 nodes	89.76%	89.05%	90.47%	90.74%
	16 nodes	90.14%	90.13%	90.22%	91.07%

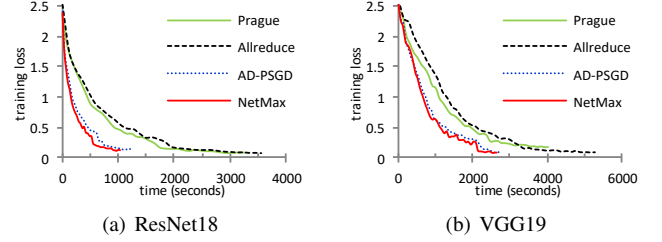


Fig. 9. Training loss with 8 worker nodes in a homogeneous network.

achieves a fast convergence rate, bringing about the decrement of the overall convergence time in NetMax.

Table II summarizes the test accuracy of all the approaches corresponding to the aforementioned convergences. The result shows that all the approaches can achieve around 90% test accuracy for both ResNet18 and VGG19, while NetMax performs slightly better than the others. Such an accuracy gain of NetMax is attributed to the adjustable probabilities for worker nodes to select neighbors introducing more randomness in the distributed training, which could benefit the training to escape from a shallow and sharp sub-optima that leads to inferior generalization [25, 26, 27].

For the homogeneous network setting, Fig. 9 exhibits that NetMax is also advanced in the fastest convergence time. Note that NetMax and AD-PSGD have similar convergence trends. This is because the communication links in the homogeneous network share roughly equivalent link speed. As a consequence, NetMax lets worker nodes choose their neighbors randomly and uniformly to favor fast convergence, resulting in being similar to AD-PSGD in terms of communication. In contrast, Allreduce-SGD and Prague converge much slower, since they both suffer from higher communication cost than NetMax and AD-PSGD as illustrated in Fig. 6.

Correspondingly, Table III summarizes the test accuracy of all the approaches with respect to the above convergences, and the results are basically consistent with Table II.

#### E. Scalability

We evaluate the scalability of the four decentralized ML approaches. For ease of illustration, we set the training time after finishing a specified epoch (i.e., 64 for ResNet18 and 82 for VGG19) in Allreduce-SGD with 4 worker nodes as the baseline, and refer to it to calculate the speedup of other runs. The results shown in Fig. 10 and Fig. 11 indicate that NetMax is superior in scalability over the other approaches under both the heterogeneous and homogeneous network settings. Moreover, the saving of communication cost in NetMax further enlarges the performance gap along with the increment of nodes.

TABLE III  
ACCURACY OF MODELS TRAINED OVER A HOMOGENEOUS NETWORK

		Prague	Allreduce	AD-PSGD	NetMax
ResNet18	4 nodes	89.37%	89.45%	89.83%	90.19%
	6 nodes	89.65%	90.25%	90.17%	89.98%
	8 nodes	90.32%	90.16%	89.95%	90.24%
VGG19	4 nodes	89.97%	90.41%	90.69%	90.62%
	6 nodes	91.02%	90.52%	90.88%	91.07%
	8 nodes	89.62%	89.86%	90.30%	90.47%

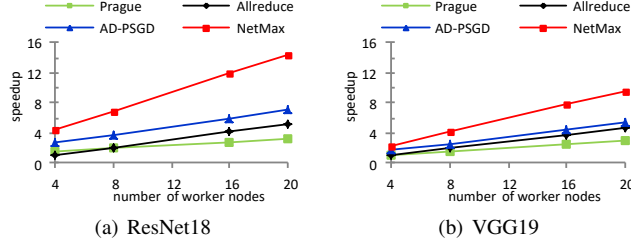


Fig. 10. Speedup w.r.t. number of worker nodes in a heterogeneous network.

### F. Non-uniform Data Partitioning

While the previous experiments are based on uniform data partitioning, in the following subsections, we evaluate NetMax with non-uniform data partitioning. We train different models on various datasets with the following three settings.

- We train ResNet18 on CIFAR10, CIFAR100, and Tiny-ImageNet datasets, respectively, with 8 worker nodes instantiated in two GPU servers. Each server hosts 4 worker nodes, and each node occupies one GPU. Each dataset is partitioned into 10 segments. The worker nodes on the first server, indexed by  $\langle w_0, w_1, w_2, w_3 \rangle$ , process one segment of data, respectively. The worker nodes on the second server, indexed by  $\langle w_4, w_5, w_6, w_7 \rangle$ , process  $\langle 2, 1, 2, 1 \rangle$  segments of data, respectively. The batch size of each worker node is set to  $64 \times$  the segment number. For example, the batch size of  $w_4$  is  $64 \times 2 = 128$ . The learning rate starts from 0.1 and decays by a factor of 10 at epoch 80. The total epoch number is set to 120.
- We train ResNet50 on ImageNet dataset with 16 worker nodes instantiated in two GPU servers. Each server hosts 8 worker nodes, and each node occupies one GPU. The dataset is partitioned into 20 segments. The worker nodes on the first server process one segment of data, respectively. The worker nodes on the second server process  $\langle 2, 1, 2, 1, 2, 1, 2, 1 \rangle$  segments of data, respectively. The batch size of each worker node is set to  $64 \times$  the segment number. The learning rate starts from 0.1 and decays by a factor of 10 at epoch 40. The total epoch number is 75.
- We train MobileNet on MNIST dataset with 8 worker nodes instantiated in two GPU servers. We consider an extreme condition where the worker nodes' data distributions are non-IID, as shown in Table IV. The batch size is set to 32, and the learning rate is set to 0.01.

Fig. 12 and Fig. 13 show the results of the former two settings. We can observe that NetMax exhibits a similar convergence rate (i.e., converging efficiency in terms of epochs/iterations) comparing to the other approaches (from

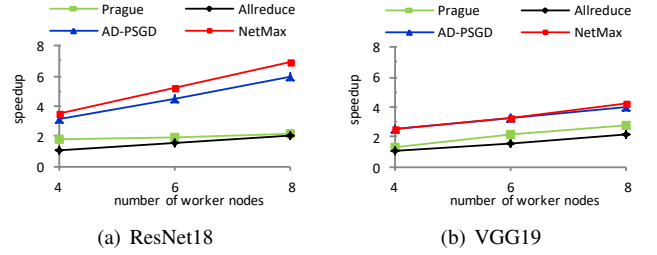


Fig. 11. Speedup w.r.t. number of worker nodes in a homogeneous network.

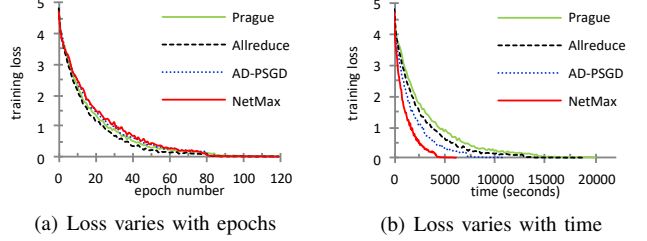


Fig. 12. Training ResNet18 on CIFAR100.

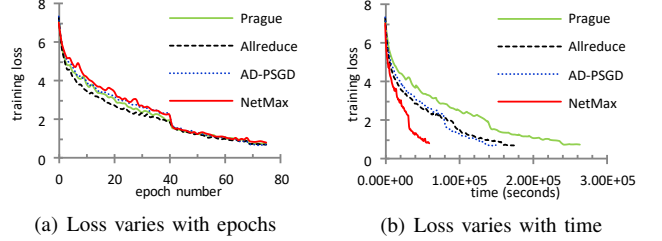


Fig. 13. Training ResNet50 on ImageNet.

Fig. 12(a) and Fig. 13(a)), and it converges much faster than the others approaches in terms of training time (from Fig. 12(b) and Fig. 13(b)). As the experimental results of training on MNIST, Tiny-ImageNet, and CIFAR10 are similar to the results of CIFAR100 and ImageNet shown in Fig.13 and Fig.12, we defer them to Appendix F for better readability.

Table V summarizes the test accuracy of all the approaches with non-uniform data distribution. We observe that the test accuracy of MobileNet trained on MNIST is around 93%, which is much lower than the typical MNIST accuracy of around 99%. This accuracy loss is caused by the non-IID data distribution.

Comparing to the other approaches, NetMax achieves a high training speed and a comparable accuracy. There are two main reasons. First, a worker node  $i$  can obtain the model from a low-link-speed neighbor  $j$  in two ways. One is pulling the model of node  $j$  directly with a low selection probability. The other way is pulling the models from other neighbors, by which the model information of node  $j$  can be propagated to node  $i$ . Second, as described in Algorithm 2 (lines 13-15), a worker node assigns a higher weight to the pulled models from neighbors with lower selection probabilities for updating its own model. This design helps the worker nodes maintain enough information from the low-speed neighbors.

TABLE IV  
DISTRIBUTION OF MNIST ACROSS WORKER NODES

Worker	Lost labels	Server	Worker	Lost labels	Server
$w_0$	0, 1, 2	server 1	$w_4$	5,6,7	server 2
$w_1$	0, 1, 3	server 1	$w_5$	5,6,8	server 2
$w_2$	0, 1, 4	server 1	$w_6$	5,6,9	server 2
$w_3$	0, 1, 5	server 1	$w_7$	5,6,0	server 2

TABLE V  
ACCURACY OF MODELS TRAINED OVER A HETEROGENEOUS NETWORK  
WITH NON-UNIFORM DATA PARTITIONING

Dataset	Model	Prague	Allreduce	AD-PSGD	NetMax
CIAR10	ResNet18	89.16%	89.38%	89.58%	89.63%
CIAR100	ResNet18	71.87%	71.28%	71.88%	72.17%
MNIST	MobileNet	92.29%	91.58%	91.41%	93.36%
Tiny-ImageNet	ResNet18	56.84%	57.02%	56.15%	57.42%
ImageNet	ResNet50	72.37%	73.64%	73.08%	73.27%

### G. Training Small Models on Complex Datasets

To evaluate the effectiveness of NetMax for training a small model on a complex dataset, we train MobileNet on CIFAR100. MobileNet is a small model with approximately 4.2M parameters, designed for the mobile and embedded platforms. In this set of experiments, we use the same setting as for training ResNet18 on CIFAR100 in Section V-F. In addition, we include two parameter server (PS) implementations (synchronous and asynchronous) as other baselines. For both implementations, the PS is assigned to one GPU server.

As shown in Fig. 14, NetMax achieves comparable convergence rate to Prague, Allreduce-SGD, AD-PSGD, and PS with synchronous training. Furthermore, it converges much faster *w.r.t.* the training time than the other competitors. From Fig. 14(a), PS with asynchronous training achieves the worst convergence rate. The reason is that the worker nodes located on the same server with the PS iterate much faster than the other nodes. The model maintained by the PS enhances the information from the faster nodes and weakens the information from the slower nodes, leading to a low convergence rate. Fig. 14(b) shows the training loss over time. We can see that the training speed of PS with synchronous training is the worst, while the training speed of PS with asynchronous training is similar to Allreduce-SGD. This is mainly because of the high communication cost between the PS and the worker nodes.

Table VI shows the test accuracy of MobileNet trained with various approaches. NetMax achieves a similar accuracy comparing to the other competitors. In addition, by comparing Table V and Table VI, we observe that the accuracy of MobileNet on CIFAR100 is about 63%, which is smaller than the 71% accuracy of ResNet18 on CIFAR100. The reason is that MobileNet is very simple, and its capacity to learn on complex data is not as good as larger models.

### H. Extension of AD-PSGD with Network Monitor

We implement AD-PSGD with a Network Monitor similar to the one used in our NetMax and evaluate its performance with the same setting described in Section V-F for training ResNet18 on CIFAR100. As shown Fig. 15, AD-PSGD with a Network Monitor exhibits lower training time than the

TABLE VI  
ACCURACY OF MOBILENET TRAINED ON CIFAR100 WITH  
NON-UNIFORM DATA PARTITIONING

Prague	Allreduce	AD-PSGD	PS-syn	PS-asyn	NetMax
63.32%	63.83%	63.72%	63.40%	63.77%	64.28%

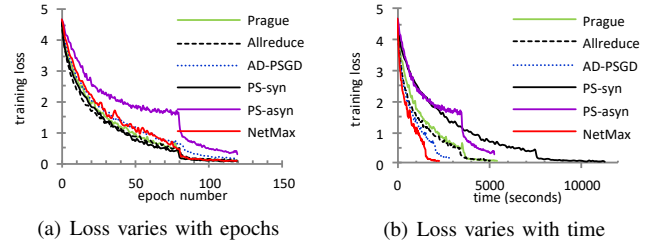


Fig. 14. Training MobileNet on CIFAR100.

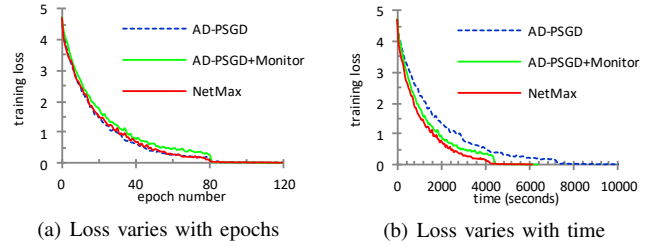


Fig. 15. Extension of AD-PSGD with Network Monitor

standard AD-PSGD. However, its convergence rate is lower than that of the standard AD-PSGD. The reason is similar to the one presented in Section V-C. Besides, the convergence rate of AD-PSGD with a Network Monitor is slightly lower compared to NetMax. This is because NetMax assigns a higher weight to the model pulled from a lower-speed neighbor, while AD-PSGD uses the same weight. Consequently, the worker nodes in AD-PSGD need more epochs to get the model information from low-speed neighbors.

## VI. RELATED WORK

**Centralized Parallel SGD.** The most popular implementation of the Centralized Parallel SGD (C-PSGD) is the parameter server architecture, where the models of distributed worker nodes are synchronized by a central parameter server [10, 11]. All the worker nodes exchange information with the parameter server to synchronize their training. The training is constrained by the network capacity at the parameter server.

**Decentralized Parallel SGD.** To address the communication bottleneck of the central server, a number of Decentralized Parallel SGD (D-PSGD) approaches are proposed. The worker nodes in D-PSGD are assumed to be connected to each other such that they can synchronize their models via peer-to-peer communication. In this way, the traffic among worker nodes can be evenly distributed. Synchronous D-PSGD approaches [12, 13, 14] keep worker nodes training and communicating at the same pace. In contrast, asynchronous D-PSGD approaches [15, 16] enable worker nodes to train their local models and to communicate independently. Although these decentralized approaches address the problem of the centralized

bottleneck, they still suffer from frequent communication via low-speed links in heterogeneous networks.

**Heterogeneity-aware Distributed Training.** To better adapt to the heterogeneous resources, several approaches are proposed for decentralized training [18, 19, 28]. Prague [18] randomly divides the worker nodes into multiple groups at each iteration such that the worker nodes in each group conduct ring-allreduce collective operation to average their models. The model averaging operations of different groups are independent, which can tolerate random worker slowdown. However, in heterogeneous networks, the node groups may still frequently communicate through low-speed links, resulting in a high communication cost. SAPS-PSGD [19] assumes that the network is static and lets the worker nodes communicate with each other in a fixed topology consisting of initially high-speed links. However, in dynamic networks, some links of the topology used by SAPS-PSGD may become low-speed links during the training, leading to a high communication cost. DLion [28] adjusts the size of the transferred model partition according to the network link capacity. However, exchanging only part of the model may cause divergence of the training. Hop [29] and Gaia [5] use bounded staleness for distributed settings to guarantee convergence. But when network links experience a continuous slowdown, the whole system would be dragged down by these low-speed links.

In contrast, NetMax enables the worker nodes to communicate efficiently via high-speed links to achieve low communication time even in dynamic networks, while guaranteeing model convergence.

## VII. CONCLUSION

In this paper, we propose NetMax, a communication-efficient asynchronous decentralized PSGD approach, to accelerate distributed machine learning over heterogeneous networks. Compared to C-PSGD, NetMax adopts decentralized training to eliminate the communication bottleneck at the central node. Compared to prior decentralized approaches, NetMax can adapt to the network dynamics and enable worker nodes to communicate preferably via high-speed links. This can significantly speed up the training. In addition, we formally prove the convergence of our proposed approach.

## ACKNOWLEDGMENTS

The work of Qian Lin, Dumitrel Loghin, Beng Chin Ooi, and Yuncheng Wu was supported by Singapore Ministry of Education Academic Research Fund Tier 3 under MOEs official grant number MOE2017-T3-1-007. The work of Pan Zhou was supported by the China Scholarship Council.

## REFERENCES

- [1] W. Xiao, R. Bhardwaj, R. Ramjee, M. Sivathanu, N. Kwatra, Z. Han, P. Patel, X. Peng, H. Zhao, Q. Zhang *et al.*, “Gandiva: Introspective cluster scheduling for deep learning,” in *OSDI*, 2018, pp. 595–610.
- [2] M. Jeon, S. Venkataraman, A. Phanishayee, J. Qian, W. Xiao, and F. Yang, “Analysis of large-scale multi-tenant gpu clusters for dnn training workloads,” in *USENIX ATC*, 2019, pp. 947–960.
- [3] B. C. Ooi, K. Tan, S. Wang, W. Wang, Q. Cai, G. Chen, J. Gao, Z. Luo, A. K. H. Tung, Y. Wang, Z. Xie, M. Zhang, and K. Zheng, “SINGA: A distributed deep learning platform,” in *ACM Multimedia*, 2015, pp. 685–688.
- [4] Q. Cai, W. Guo, H. Zhang, D. Agrawal, G. Chen, B. C. Ooi, K. Tan, Y. M. Teo, and S. Wang, “Efficient distributed memory management with RDMA and caching,” *Proc. VLDB Endow.*, vol. 11, no. 11, pp. 1604–1617, 2018.
- [5] K. Hsieh, A. Harlap, N. Vijaykumar, D. Konomis, G. R. Ganger, P. B. Gibbons, and O. Mutlu, “Gaia: Geo-distributed machine learning approaching lan speeds,” in *NSDI*, 2017, pp. 629–647.
- [6] K. Zheng, S. Cai, H. R. Chua, W. Wang, K. Y. Ngiam, and B. C. Ooi, “Tracer: A framework for facilitating accurate and interpretable analytics for high stakes applications,” in *SIGMOD*, 2020, pp. 1747–1763.
- [7] W. Wang, M. Zhang, G. Chen, H. V. Jagadish, B. C. Ooi, and K. Tan, “Database meets deep learning: Challenges and opportunities,” *SIGMOD Rec.*, vol. 45, pp. 17–22, 2016.
- [8] J. Gu, M. Chowdhury, K. G. Shin, Y. Zhu, M. Jeon, J. Qian, H. Liu, and C. Guo, “Tiresias: A gpu cluster manager for distributed deep learning,” in *NSDI*, 2019, pp. 485–500.
- [9] A. C. Zhou, Y. Xiao, Y. Gong, B. He, J. Zhai, and R. Mao, “Privacy regulation aware process mapping in geo-distributed cloud data centers,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 8, pp. 1872–1888, 2019.
- [10] T. Chilimbi, Y. Suzue, J. Apacible, and K. Kalyanaraman, “Project adam: Building an efficient and scalable deep learning training system,” in *OSDI*, 2014, pp. 571–582.
- [11] C. Chen, W. Wang, and B. Li, “Round-robin synchronization: Mitigating communication bottlenecks in parameter servers,” in *INFOCOM*, 2019, pp. 532–540.
- [12] X. Jia, S. Song, W. He, Y. Wang, H. Rong, F. Zhou, L. Xie, Z. Guo, Y. Yang, L. Yu *et al.*, “Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes,” *arXiv preprint arXiv:1807.11205*, 2018.
- [13] H. Tang, X. Lian, M. Yan, C. Zhang, and J. Liu, “D2: Decentralized training over decentralized data,” in *ICML*, 2018, pp. 4855–4863.
- [14] R. Xin, S. Kar, and U. A. Khan, “Decentralized stochastic first-order methods for large-scale machine learning,” *arXiv preprint arXiv:1907.09648*, 2019.
- [15] X. Lian, W. Zhang, C. Zhang, and J. Liu, “Asynchronous decentralized parallel stochastic gradient descent,” in *ICML*, 2018, pp. 3049–3058.
- [16] I. Hegedűs, G. Danner, and M. Jelasity, “Gossip learning as a decentralized alternative to federated learning,” in *DAIS*, 2019, pp. 74–90.
- [17] A. Lalitha, O. C. Kilinc, T. Javidi, and F. Koushanfar, “Peer-to-peer federated learning on graphs,” *arXiv preprint arXiv:1901.11173*, 2019.
- [18] Q. Luo, J. He, Y. Zhuo, and X. Qian, “Prague: High-performance heterogeneity-aware asynchronous decentralized training,” in *ASPLOS*, 2020, pp. 401–416.
- [19] Z. Tang, S. Shi, and X. Chu, “Communication-efficient decentralized learning with sparsification and adaptive peer selection,” *arXiv preprint arXiv:2002.09692*, 2020.
- [20] Y. Wu, S. Cai, X. Xiao, G. Chen, and B. C. Ooi, “Privacy preserving vertical federated learning for tree-based models,” *Proc. VLDB Endow.*, vol. 13, no. 11, pp. 2090–2103, 2020.
- [21] J. S. Hunter, “The exponentially weighted moving average,” *Journal of Quality Technology*, vol. 18, no. 4, pp. 203–210, 1986.
- [22] P. H. Jin, Q. Yuan, F. Iandola, and K. Keutzer, “How to scale distributed deep learning?” *arXiv preprint arXiv:1611.04581*, 2016.
- [23] S. P. Boyd, A. Ghosh, and B. Prabhakar, “Randomized gossip algorithms,” *IEEE Trans. Inf. Theory*, vol. 52, no. 6, pp. 2508–2530, 2006.
- [24] B. Pfaff, J. Pettit, and Koponen. (2019) Open vswitch. [Online]. Available: <http://www.openvswitch.org/>
- [25] A. Neelakantan, L. Vilnis, Q. V. Le, L. Kaiser, K. Kurach, I. Sutskever, and J. Martens, “Adding gradient noise improves learning for very deep networks,” *arXiv preprint arXiv:1511.06807*, 2017.
- [26] R. Kleinberg, Y. Li, and Y. Yuan, “An alternative view: When does sgd escape local minima?” *arXiv preprint arXiv:1802.06175*, 2018.
- [27] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang, “On large-batch training for deep learning: Generalization gap and sharp minima,” in *ICLR*, 2017.
- [28] R. Hong and A. Chandra, “Dlion: decentralized distributed deep learning in micro-clouds,” in *HotCloud*, 2019.
- [29] Q. Luo, J. Lin, Y. Zhuo, and X. Qian, “Hop: Heterogeneity-aware decentralized training,” in *ASPLOS*, 2019, pp. 893–907.

- [30] W. Fulton, "Eigenvalues, invariant factors, highest weights, and schubert calculus," *Bulletin of the American Mathematical Society*, vol. 37, no. 3, 1999.
- [31] Kirkland and Steve, "A cycle-based bound for subdominant eigenvalues of stochastic matrices," *Linear and Multilinear Algebra*, vol. 57, no. 3, pp. 247–266, 2009.
- [32] M. Rigo, *Advanced graph theory and combinatorics*. John Wiley & Sons, 2016.
- [33] S. Gershgorin, "Gershgorin circle theorem," 2014.

## APPENDIX

In this Appendix, we first derive the feasible intervals of  $\rho$  and  $\bar{t}$  used in Algorithm 3, followed by the bound of the approximation ratio of the feasible communication policy. Next, we prove all three theorems formulated in Section IV. We end by presenting additional results when (i) training models on non-uniform data partitioning and (ii) training across multiple cloud regions.

### A. Feasible Intervals of $\rho$ and $\bar{t}$

The lower bound  $L_\rho$  of  $\rho$  is set to 0 as  $\rho$  is non-negative. According to Eq. (11),  $\alpha\rho(d_{i,m} + d_{m,i}) \leq p_{i,m} \leq 1$ . If worker  $i$  and  $m$  are neighbors, we have  $\rho \leq \frac{1}{(d_{i,m} + d_{m,i})\alpha} = 0.5/\alpha$ . Thus, the upper bound  $U_\rho$  is set to  $0.5/\alpha$ .

By combining the constraints in Eq. (10) and (11), we have

$$\bar{t} \geq \frac{\alpha\rho}{M} \sum_{m=1}^M t_{i,m}(d_{i,m} + d_{m,i}) = L_i, \forall i \in [M] \quad (25)$$

Then the lower bound  $L$  of  $\bar{t}$  is the largest  $L_i$  as follows

$$\bar{t} \geq L = \max_{i \in [M]} L_i = \max_{i \in [M]} \frac{\alpha\rho}{M} \sum_{m=1}^M t_{i,m}(d_{i,m} + d_{m,i}) \quad (26)$$

For any worker node  $i$ , any iteration time is no larger than the iteration time when communicating with the slowest neighbor. From Eq. (10), we have

$$\begin{aligned} \bar{t} &\leq \frac{1}{M} \sum_{m=1}^M p_{i,m} \max_{m \in [M]} t_{i,m} d_{i,m} \\ &= \frac{1}{M} \max_{m \in [M]} t_{i,m} d_{i,m} = U_i, \forall i \in [M] \end{aligned} \quad (27)$$

Then the upper bound  $U$  of  $\bar{t}$  is the smallest  $U_i$ ,

$$\bar{t} \leq U = \min_{i \in [M]} U_i = \min_{i \in [M]} \frac{1}{M} \max_{m \in [M]} t_{i,m} d_{i,m}. \quad (28)$$

### B. Approximation Ratio of the Feasible Communication Policy

We assume that the number  $M$  of worker nodes is more than 3, the network connecting worker nodes is heterogeneous and fully-connected. From Eq. (22), the matrix  $\mathbf{Y}_P$  can be rewritten as

$$\mathbf{Y}_P = (1 - 4\alpha\rho)\mathbf{I} + \frac{4\alpha\rho}{M}\mathbf{1}\mathbf{1}^T + \mathbf{W} \quad (29)$$

where the entries of  $\mathbf{W}$  can be written as

$$\begin{cases} w_{i,i} = \alpha^2 \rho^2 \sum_{m \in [M], m \neq i} (p_i p_{i,m} \gamma_{i,m}^2 + p_m p_{m,i} \gamma_{m,i}^2), \forall i \in [M] \\ w_{i,m} = -\alpha^2 \rho^2 (p_i p_{i,m} \gamma_{i,m}^2 + p_m p_{m,i} \gamma_{m,i}^2), \forall i, m \in [M], m \neq i \end{cases} \quad (30)$$

From Eq. (30), the sum of the rows of  $\mathbf{W}$  is 0. Therefore,  $\lambda_{\mathbf{W}} = 0$  is one of the eigenvalues of  $\mathbf{W}$ . In addition, as  $\text{Tr}(\mathbf{W})$  is positive,  $\lambda_{\mathbf{W}} = 0$  is not the largest eigenvalue  $\lambda_{\mathbf{W},1}$  of  $\mathbf{W}$ . Let  $\mathbf{Z} = (1 - 4\alpha\rho)\mathbf{I} + \frac{4\alpha\rho}{M}\mathbf{1}\mathbf{1}^T$ , then the eigenvalues  $\lambda_{\mathbf{Z},1} \geq \lambda_{\mathbf{Z},2} \geq \dots \geq \lambda_{\mathbf{Z},M}$  of  $\mathbf{Z}$  are  $1, 1 - 4\alpha\rho, \dots, 1 - 4\alpha\rho$ . According to [30], there exists the following relationship among the eigenvalues of  $\mathbf{Y}_P$ ,  $\mathbf{W}$  and  $\mathbf{Z}$ ,

$$\lambda_{M-i-j} \geq \lambda_{\mathbf{Z},M-i} + \lambda_{\mathbf{W},M-j} \quad (31)$$

Let  $i = 0$  and  $\lambda_{\mathbf{W},M-j} = 0$ , then Eq. (31) becomes

$$\lambda_2 \geq \lambda_{\mathbf{Z},M} \geq 1 - 4\alpha\rho \quad (32)$$

When the graph connecting the worker nodes is fully-connected, from Eq. (11), we have that  $p_{i,m} \geq 2\alpha\rho, \forall i, m \in [M], i \neq m$ . As  $\sum_{m \in [M], m \neq i} p_{i,m} \leq 1, \forall i \in [M]$ , we have

$$\alpha\rho \leq \frac{1}{2(M-1)} \quad (33)$$

By using Eq. (32) and Eq. (33), we obtain the lower bound of the second largest eigenvalue of  $\mathbf{Y}_P$  as follows

$$\lambda_2 \geq \frac{M-3}{M-1} \quad (34)$$

Let  $a$  be the minimum positive entry in  $\mathbf{Y}_P$ . Based on [31], the second largest eigenvalue  $\lambda_2$  of  $\mathbf{Y}_P$  can be bounded by

$$\lambda_2 \leq \frac{1 - 2a + a^{M+1}}{1 - 2a + a^M} \quad (35)$$

Let  $\lambda^*$  be the second largest eigenvalue when the objective function in Eq. (8) reaches the optimal  $l(\lambda^*)$ . By combining Eq. (9) and Eq. (34), we have that

$$l(\lambda^*) = \bar{t} \frac{\ln \varepsilon}{\ln \lambda^*} \geq \bar{t} \frac{\ln \varepsilon}{\ln \frac{M-3}{M-1}} \geq L \frac{\ln \varepsilon}{\ln \frac{M-3}{M-1}} \quad (36)$$

Let  $l(\lambda_2)$  denote the objective function of the feasible communication policy derived by Algorithm 3. By combining Eq. (9) and Eq. (35), we have that

$$l(\lambda_2) = \bar{t} \frac{\ln \varepsilon}{\ln \lambda_2} \leq U \frac{\ln \varepsilon}{\ln \frac{1 - 2a + a^{M+1}}{1 - 2a + a^M}} \quad (37)$$

By combining Eq. (36) and Eq. (37), we derive the bound of the approximation ratio as

$$\frac{l(\lambda_2)}{l(\lambda^*)} \leq \frac{U}{L} \frac{\ln(M-1) - \ln(M-3)}{\ln(1 - 2a + a^M) - \ln(1 - 2a + a^{M+1})} \quad (38)$$

### C. Proof of Theorem 1

*Proof.* For simplicity,  $\nabla f(x_i^k; \mathcal{D}_i)$  denotes  $\nabla f(x_i^k)$ . Let  $\lambda_1$  and  $\lambda_2$  be the largest and second largest eigenvalue of matrix  $\mathbb{E}[(\mathbf{D}^k)^T \mathbf{D}^k]$ , respectively. We can bound the following expected sum of squares deviation by applying Eq. (18).

$$\begin{aligned} &\mathbb{E}[\|\mathbf{x}^{k+1} - x^* \mathbf{1}\|^2 | \mathbf{x}^k] \\ &= \mathbb{E}[\|\mathbf{D}^k(\mathbf{x}^k - \alpha \mathbf{g}^k) - x^* \mathbf{1}\|^2 | \mathbf{x}^k] \\ &= \mathbb{E}[(\mathbf{x}^k - x^* \mathbf{1} - \alpha \mathbf{g}^k)^T (\mathbf{D}^k)^T \mathbf{D}^k (\mathbf{x}^k - x^* \mathbf{1} - \alpha \mathbf{g}^k) | \mathbf{x}^k] \\ &\leq \lambda \mathbb{E}[(\mathbf{x}^k - x^* \mathbf{1} - \alpha \mathbf{g}^k)^T (\mathbf{x}^k - x^* \mathbf{1} - \alpha \mathbf{g}^k) | \mathbf{x}^k] \end{aligned} \quad (39)$$

where  $\lambda = \lambda_1$ . If  $\mathbb{E}[(\mathbf{D}^k)^T \mathbf{D}^k]$  is doubly stochastic matrix,  $\lambda = \lambda_2$  [15, 23].

$$\begin{aligned}
& \mathbb{E}[(\mathbf{x}^k - x^* \mathbf{1} - \alpha \mathbf{g}^k)^T (\mathbf{x}^k - x^* \mathbf{1} - \alpha \mathbf{g}^k) | \mathbf{x}^k] \\
&= \mathbb{E}[\|\mathbf{x}^k - x^* \mathbf{1}\|^2 - 2\alpha (\mathbf{g}^k)^T (\mathbf{x}^k - x^* \mathbf{1}) + \alpha^2 (\mathbf{g}^k)^T \mathbf{g}^k | \mathbf{x}^k] \\
&= \|\mathbf{x}^k - x^* \mathbf{1}\|^2 - 2\alpha \sum_{n=1}^M p_n \nabla f(x_n^k)^T (\mathbf{x}^k - x^*) \\
&\quad + \alpha^2 \sum_{n=1}^M p_n \nabla f(x_n^k)^T \nabla f(x_n^k) + \alpha^2 \sum_n p_n \mathbb{E}[(\xi_n^k)^T \xi_n^k]
\end{aligned} \tag{40}$$

In Eq. (40), we drop the terms that are linear in  $\xi_i^k$  since  $\xi_i^k$  has zero mean assumption. To bound the second term at the right side of Eq. (40), we apply the property of convex functions,

$$\begin{aligned}
& (\nabla f(x) - \nabla f(z))^T (x - z) \\
&\geq \frac{\mu L}{\mu + L} (x - z)^T (x - z) \\
&\quad + \frac{1}{\mu + L} (\nabla f(x) - \nabla f(z))^T (\nabla f(x) - \nabla f(z)), \forall x, z
\end{aligned} \tag{41}$$

With  $x = x_i^k$  and  $z = x^*$ , we have that

$$\begin{aligned}
& -\nabla f(x_n^k)^T (x_n^k - x^*) = -\nabla f(x_n^k) - 0)^T (x_n^k - x^*) \\
&\leq -\frac{\mu L}{\mu + L} (x_n^k - x^*)^T (x_n^k - x^*) - \frac{1}{\mu + L} \nabla f(x_n^k)^T \nabla f(x_n^k)
\end{aligned} \tag{42}$$

By applying Eq. (42), we can bound

$$\begin{aligned}
& -2\alpha \sum_{n=1}^M p_n \nabla f(x_n^k)^T (x_n^k - x^*) \\
&\leq -\frac{2\alpha \mu L}{\mu + L} \sum_{n=1}^M p_n (x_n^k - x^*)^T (x_n^k - x^*) \\
&\quad - \frac{2\alpha}{\mu + L} \sum_{n=1}^M p_n \nabla f(x_n^k)^T \nabla f(x_n^k) \\
&\leq -\frac{2\alpha \mu L}{\mu + L} p_{\min} \sum_{n=1}^M (x_n^k - x^*)^T (x_n^k - x^*) \\
&\quad - \frac{2\alpha}{\mu + L} \sum_{n=1}^M p_n \nabla f(x_n^k)^T \nabla f(x_n^k)
\end{aligned} \tag{43}$$

where  $p_{\min}$  is the smallest value among  $p_n, \forall n \in [M]$ . Combining Eq.(40) and Eq. (43), we have that

$$\begin{aligned}
& \mathbb{E}[(\mathbf{x}^k - x^* \mathbf{1} - \alpha \mathbf{g}^k)^T (\mathbf{x}^k - x^* \mathbf{1} - \alpha \mathbf{g}^k) | \mathbf{x}^k] \\
&\leq (1 - \frac{2\alpha \mu L}{\mu + L} p_{\min}) \|\mathbf{x}^k - x^* \mathbf{1}\|^2 + \alpha^2 \sigma^2
\end{aligned} \tag{44}$$

$$+ (\alpha^2 - \frac{2\alpha}{\mu + L}) \sum_{n=1}^M p_n \nabla f(x_n^k)^T \nabla f(x_n^k) \tag{45}$$

The term in Eq. (45) can be dropped if  $0 < \alpha < \frac{2}{\mu + L}$ . Thus, we have

$$\begin{aligned}
& \mathbb{E}[(\mathbf{x}^k - x^* \mathbf{1} - \alpha \mathbf{g}^k)^T (\mathbf{x}^k - x^* \mathbf{1} - \alpha \mathbf{g}^k) | \mathbf{x}^k] \\
&\leq (1 - \frac{2\alpha \mu L}{\mu + L} p_{\min}) \|\mathbf{x}^k - x^* \mathbf{1}\|^2 + \alpha^2 \sigma^2
\end{aligned} \tag{46}$$

By applying Eq. (46) in Eq (39), we have that

$$\begin{aligned}
& \mathbb{E}[\|\mathbf{x}^{k+1} - x^* \mathbf{1}\|^2 | \mathbf{x}^k] \\
&\leq \lambda \mathbb{E}[(\mathbf{x}^k - x^* \mathbf{1} - \alpha \mathbf{g}^k)^T (\mathbf{x}^k - x^* \mathbf{1} - \alpha \mathbf{g}^k) | \mathbf{x}^k] \\
&\leq \lambda (1 - \frac{2\alpha \mu L}{\mu + L} p_{\min}) \|\mathbf{x}^k - x^* \mathbf{1}\|^2 + \lambda \alpha^2 \sigma^2 \\
&\leq \lambda \|\mathbf{x}^k - x^* \mathbf{1}\|^2 + \lambda \alpha^2 \sigma^2
\end{aligned} \tag{47}$$

By unrolling the recursion, we have that

$$\begin{aligned}
\mathbb{E}[\|\mathbf{x}^k - x^* \mathbf{1}\|^2] &\leq \lambda^k \|\mathbf{x}^0 - x^* \mathbf{1}\|^2 + \frac{1 - \lambda^k}{1 - \lambda} \lambda \alpha^2 \sigma^2 \\
&\leq \lambda^k \|\mathbf{x}^0 - x^* \mathbf{1}\|^2 + \frac{\lambda}{1 - \lambda} \alpha^2 \sigma^2. \square
\end{aligned} \tag{48}$$

#### D. Proof of Theorem 2

*Proof.* For the convergence analysis under dynamic networks, we assume that the network changes at each step  $k$ . Thus, we consider different values of  $\lambda$  at each step, denoted by  $\lambda_k$ . Then, we have

$$\mathbb{E}[\|\mathbf{x}^k - x^* \mathbf{1}\|^2 | \mathbf{x}^{k-1}] \leq \lambda_k \|\mathbf{x}^{k-1} - x^* \mathbf{1}\|^2 + \alpha^2 \sigma^2 \lambda_k \tag{49}$$

By unrolling the recursion, we have that

$$\begin{aligned}
& \mathbb{E}[\|\mathbf{x}^k - x^* \mathbf{1}\|^2] \\
&\leq \left[ \prod_{n=1}^k \lambda_n \right] \|\mathbf{x}^0 - x^* \mathbf{1}\|^2 + \sum_{n=1}^k \lambda_n \alpha^2 \sigma^2 \prod_{m=n+1}^k \lambda_{max} \\
&\leq \lambda_{max}^k \|\mathbf{x}^0 - x^* \mathbf{1}\|^2 + \sum_{n=1}^k \lambda_{max} \alpha^2 \sigma^2 \lambda_{max}^{k-n} \\
&\leq \lambda_{max}^k \|\mathbf{x}^0 - x^* \mathbf{1}\|^2 + \frac{1 - \lambda_{max}^k}{1 - \lambda_{max}} \lambda_{max} \alpha^2 \sigma^2 \\
&\leq \lambda_{max}^k \|\mathbf{x}^0 - x^* \mathbf{1}\|^2 + \frac{\lambda_{max}}{1 - \lambda_{max}} \alpha^2 \sigma^2. \square
\end{aligned} \tag{50}$$

#### E. Proof of Theorem 3

Before proving Theorem 3, we first introduce three lemmas.

**Lemma 1.** If the probability matrix  $\mathbf{P}$  is a feasible solution of the optimization problem in Eq. (8)-(13), then  $\mathbf{Y}_{\mathbf{P}}$  is a symmetric matrix and each row/column of  $\mathbf{Y}_{\mathbf{P}}$  sums to 1.

*Proof.* As  $\mathbf{P}$  is a feasible solution, the constraints in Eq. (10) are satisfied and we have  $\bar{t}_i = \sum_{m=1}^M p_{i,m} t_{i,m} d_{i,m} =$

$M\bar{t}, \forall i \in [M]$ . Using Eq. (3), we have  $p_i = \frac{1/\bar{t}_i}{\sum_{m=1}^M 1/\bar{t}_m} = \frac{1}{M}, \forall i \in [M]$ . Using Eq. (22), the rows of  $\mathbf{Y}_P$  sum to

$$\begin{aligned} \sum_{m=1}^M y_{i,m} &= 1 - \alpha\rho \sum_{\substack{m \in [M] \\ m \neq i}} p_i p_{i,m} \gamma_{i,m} + \alpha\rho \sum_{\substack{m \in [M] \\ m \neq i}} p_m p_{m,i} \gamma_{i,m} \\ &= 1 - \alpha\rho \sum_{\substack{m \in [M] \\ m \neq i}} \frac{1}{M} (p_{i,m} \gamma_{i,m} - p_{m,i} \gamma_{m,i}) \\ &= 1, \forall i \in [M] \end{aligned} \quad (51)$$

From Eq. (22),  $y_{i,m} = y_{m,i}$ , thus  $\mathbf{Y}_P$  is symmetric and the columns of  $\mathbf{Y}_P$  also sum to 1.  $\square$

**Lemma 2.** *If the probability matrix  $\mathbf{P}$  is a feasible solution of optimization problem in Eq. (8)-(13), then  $\mathbf{Y}_P$  is non-negative.*

*Proof.* As  $\mathbf{P}$  is a feasible solution, the constraints in Eq. (11) are satisfied. By applying Eq. (11), for any  $p_{i,m} \neq 0, i \neq m$ , we have that

$$\alpha\rho\gamma_{i,m} = \frac{\alpha\rho(d_{i,m} + d_{m,i})}{p_{i,m}} < 1 \quad (52)$$

By applying Eq. (22) and (52), for any  $p_{i,m} \neq 0, i \neq m$ , we have that

$$\begin{aligned} y_{i,m} &= \alpha\rho p_i p_{i,m} \gamma_{i,m} (1 - \alpha\rho\gamma_{i,m}) + \alpha\rho p_m p_{m,i} \gamma_{m,i} (1 - \alpha\rho\gamma_{m,i}) \\ &> 0 \end{aligned} \quad (53)$$

For any  $p_{i,m} = 0, i \neq m$ ,  $y_{i,m} = 0$ . Thus, for any  $i, m \in [M], i \neq m$ , we have  $y_{i,m} \geq 0$ .

Combining Eq. (22) with  $p_i = \frac{1}{M}$ , for any  $i \in [M]$ , we have

$$\begin{aligned} y_{i,i} &= 1 - \frac{2\alpha\rho}{M} \sum_{\substack{m \in [M] \\ m \neq i}} (d_{i,m} + d_{m,i}) \\ &\quad + \frac{\alpha^2 \rho^2}{M} \sum_{\substack{m \in [M] \\ m \neq i}} \left[ \frac{(d_{i,m} + d_{m,i})^2}{p_{i,m}} + \frac{(d_{i,m} + d_{m,i})^2}{p_{m,i}} \right] \\ &\geq 1 - \frac{2\alpha\rho}{M} \sum_{\substack{m \in [M] \\ m \neq i}} (d_{i,m} + d_{m,i}) + \frac{2\alpha^2 \rho^2}{M} \sum_{\substack{m \in [M] \\ m \neq i}} (d_{i,m} + d_{m,i})^2 \\ &= 1 - \frac{2\alpha\rho}{M} \sum_{\substack{m \in [M] \\ m \neq i}} \left[ (d_{i,m} + d_{m,i}) (1 - \alpha\rho(d_{i,m} + d_{m,i})) \right] \end{aligned} \quad (54)$$

$$\geq 0 \quad (55)$$

When  $\alpha\rho > 0.5$ , the term  $(d_{i,m} + d_{m,i})(1 - \alpha\rho(d_{i,m} + d_{m,i}))$  in Eq. (54) is negative. Eq. (54) reaches the smallest value of  $1 - \frac{4\alpha\rho}{M}(1 - 2\alpha\rho)$  when worker node  $i$  has only one neighbor. Since  $M \geq 2$ , the item  $1 - \frac{4\alpha\rho}{M}(1 - 2\alpha\rho)$  is non-negative and, thus, Eq. (54) is non-negative. When  $\alpha\rho \leq 0.5$ , the term  $(d_{i,m} + d_{m,i})(1 - \alpha\rho(d_{i,m} + d_{m,i}))$  is non-negative. Eq. (54) reaches the smallest value of  $1 - \frac{M-1}{M}4\alpha\rho(1 - 2\alpha\rho)$  when worker node  $i$  and all the other nodes are neighbors. As  $1 - \frac{M-1}{M}4\alpha\rho(1 - 2\alpha\rho)$  is larger than the positive polynomial

$1 - 4\alpha\rho(1 - 2\alpha\rho)$ , Eq. (54) is positive. Therefore, Eq. (55) always holds. In summary,  $\mathbf{Y}_P$  is non-negative.  $\square$

Let graph  $\mathcal{G}_A$  be an  $n \times n$  matrix  $A = [a_{i,m}]_{n \times n}$  with a vertex set  $\mathcal{V} = \{1, 2, \dots, n\}$ . There is an edge from vertex  $m$  to  $i$  if and only if  $a_{i,m} \neq 0$ . Then  $\mathcal{G}_A$  is **connected** if there is a path between every pair of vertices [32].

**Lemma 3.** *Let  $\mathbf{P}$  be a feasible solution of the optimization problem in Eq. (8)-(13). If the graph  $\mathcal{G}_P$  of  $\mathbf{P}$  is connected, then the graph  $\mathcal{G}_Y$  of  $\mathbf{Y}_P$  is also connected.*

*Proof.* From Eq. (52)-(53), we have that if  $p_{i,m} \neq 0$ , then  $y_{i,m} \neq 0$ . This means that for any path in  $\mathcal{G}_P$ , there is also a path in  $\mathcal{G}_Y$ . Thus, if the graph  $\mathcal{G}_P$  is connected, then the graph  $\mathcal{G}_Y$  is also connected.  $\square$

*Proof of Theorem 3.* From Lemma 2 and Lemma 3, we obtain that  $\mathbf{Y}_P$  is non-negative and the graph  $\mathcal{G}_Y$  of  $\mathbf{Y}_P$  is connected under Assumption 1, which means that  $\mathbf{Y}_P$  is irreducible [32]. Meanwhile, according to the Perron-Frobenius theorem [32], the non-negative irreducible  $\mathbf{Y}_P$  has a unique positive real eigenvalue equal to spectral radius. Moreover, the eigenvalues of  $\mathbf{Y}_P$  are all real since  $\mathbf{Y}_P$  is symmetric (Lemma 1). Thus, the largest eigenvalue of  $\mathbf{Y}_P$  is equal to the spectral radius and is unique. From Lemma 1 and Lemma 2, we derive that  $\mathbf{Y}_P$  is doubly stochastic symmetric. Thus, for any feasible solution  $\mathbf{P}$ , the largest eigenvalue of  $\mathbf{Y}_P$  is 1 [15, 33] and the second largest eigenvalue of  $\mathbf{Y}_P$  is strictly less than 1. Since the policy  $\mathbf{P}$  derived by Algorithm 3 is a feasible solution for the optimization problem in Eq. (8)-(13), then  $\mathbf{Y}_P$  is a doubly stochastic matrix with the second largest eigenvalue strictly less than 1. Based on Theorem 1 and Theorem 2, our decentralized training approach converges.

Combining Eq. (39) and Eq. (40), we have that

$$\begin{aligned} &\mathbb{E}[\|\mathbf{x}^{k+1} - \mathbf{x}^* \mathbf{1}\|^2] \\ &\leq \mathbb{E}[\|\mathbf{x}^k - \mathbf{x}^* \mathbf{1}\|^2] - 2\alpha \sum_{n=1}^M p_n \mathbb{E}[\nabla f(x_n^k)^T (x_n^k - \mathbf{x}^*)] \\ &\quad + \alpha^2 \sum_{n=1}^M p_n \mathbb{E}[\nabla f(x_n^k)^T \nabla f(x_n^k)] + \alpha^2 \sum_n p_n \mathbb{E}[(\xi_n^k)^T \xi_n^k] \end{aligned} \quad (56)$$

By applying Assumption 1 and  $p_n = \frac{1}{M}$ , Eq. (56) can be rewritten as follows

$$\begin{aligned} &\mathbb{E}[\|\mathbf{x}^{k+1} - \mathbf{x}^* \mathbf{1}\|^2] \\ &\leq \mathbb{E}[\|\mathbf{x}^k - \mathbf{x}^* \mathbf{1}\|^2] - \frac{2\alpha}{M} \sum_{n=1}^M \mathbb{E}[\nabla f(x_n^k)^T (x_n^k - \mathbf{x}^*)] \\ &\quad + \alpha^2 \eta^2 + \alpha^2 \delta^2 \end{aligned} \quad (57)$$

which we rearrange as

$$\begin{aligned} & \frac{2\alpha}{M} \sum_{n=1}^M \mathbb{E} \left[ \nabla f(x_n^k)^T (x_n^k - x^*) \right] \\ & \leq \mathbb{E} \left[ \|x^k - x^* \mathbf{1}\|^2 \right] - \mathbb{E} \left[ \|x^{k+1} - x^* \mathbf{1}\|^2 \right] + \alpha^2 (\eta^2 + \delta^2) \end{aligned} \quad (58)$$

From the convexity of function  $f$ , we have that

$$\nabla f(x)^T (x - x^*) \geq f(x) - f(x^*) \quad (59)$$

Combining Eq. (58) and (59), we have that

$$\begin{aligned} & 2\alpha \mathbb{E} \left[ \frac{1}{M} \sum_{n=1}^M (f(x_n^k) - f(x^*)) \right] \\ & \leq \mathbb{E} \left[ \|x^k - x^* \mathbf{1}\|^2 \right] - \mathbb{E} \left[ \|x^{k+1} - x^* \mathbf{1}\|^2 \right] + \alpha^2 (\eta^2 + \delta^2) \end{aligned} \quad (60)$$

Summarizing the inequality above from  $l = 1, 2, \dots, k$ , we have

$$\begin{aligned} & 2\alpha \sum_{l=1}^k \mathbb{E} \left[ \frac{1}{M} \sum_{n=1}^M (f(x_n^l) - f(x^*)) \right] \\ & \leq \mathbb{E} \left[ \|x^1 - x^* \mathbf{1}\|^2 \right] - \mathbb{E} \left[ \|x^{k+1} - x^* \mathbf{1}\|^2 \right] + k\alpha^2 (\eta^2 + \delta^2) \\ & \leq \mathbb{E} \left[ \|x^1 - x^* \mathbf{1}\|^2 \right] + k\alpha^2 (\eta^2 + \delta^2) \end{aligned} \quad (61)$$

Combining Eq. (48) and (61), we have

$$\begin{aligned} & 2\alpha \sum_{l=1}^k \mathbb{E} \left[ \frac{1}{M} \sum_{n=1}^M (f(x_n^l) - f(x^*)) \right] \\ & \leq \lambda \|x^0 - x^* \mathbf{1}\|^2 + \lambda \alpha^2 \delta^2 + k\alpha^2 (\eta^2 + \delta^2) \end{aligned} \quad (62)$$

which we rearrange as

$$\begin{aligned} & \sum_{l=1}^k \mathbb{E} \left[ \frac{1}{M} \sum_{n=1}^M (f(x_n^l) - f(x^*)) \right] \\ & \leq \frac{\lambda \|x^0 - x^* \mathbf{1}\|^2 + \lambda \alpha^2 \delta^2 + k\alpha^2 (\eta^2 + \delta^2)}{2k\alpha} \end{aligned} \quad (63)$$

For a chosen constant  $\alpha = \frac{c}{\sqrt{k}}$ , then we have

$$\begin{aligned} & \sum_{l=1}^k \mathbb{E} \left[ \frac{1}{M} \sum_{n=1}^M (f(x_n^l) - f(x^*)) \right] \\ & \leq \frac{\lambda \|x^0 - x^* \mathbf{1}\|^2}{2c\sqrt{k}} + \frac{c\lambda\delta^2}{2k\sqrt{k}} + \frac{c(\eta^2 + \delta^2)}{2\sqrt{k}} \end{aligned} \quad (64)$$

The above inequality means that the decentralized training converges to the optima  $x^*$  with a rate  $O(1/\sqrt{k})$ . This completes the proof.  $\square$

#### F. Training Models on Non-uniform Data Partitioning

In this section, we present additional results for training the models on non-uniform data partitioning. We use the setup described in Section V-F. When training ResNet18 on CIFAR10, NetMax achieves almost the same convergence rate compared to the other competitors due to the simplicity of classifying 10 classes, as shown in Fig. 16. When training

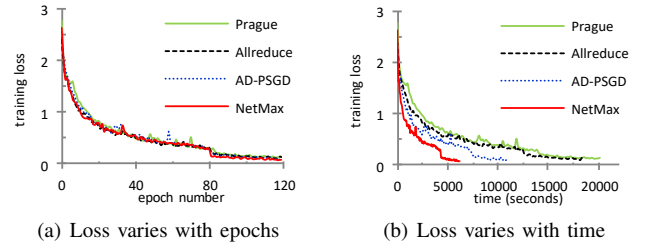


Fig. 16. Training ResNet18 on CIFAR10.

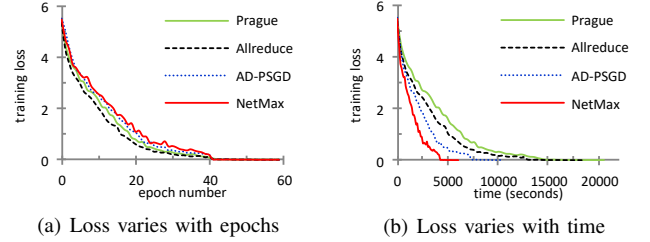


Fig. 17. Training ResNet18 on Tiny-ImageNet.

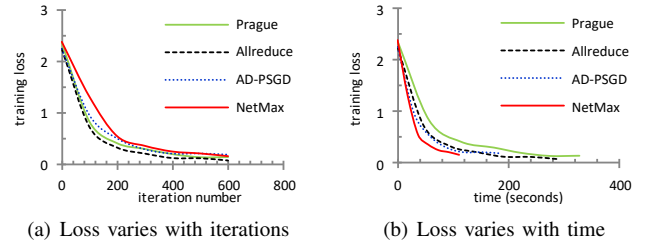


Fig. 18. Training MobileNet on MNIST.

ResNet18 on Tiny-ImageNet, NetMax achieves slightly lower convergence rate while converging much faster over time than the other competitors, as shown in Fig. 17. The final test accuracy of ResNet18 on Tiny-ImageNet for all the approaches is approximately 57%, as shown in Table V. Training models on Tiny-ImageNet can hardly achieve a high accuracy since Tiny-ImageNet is only a subset of ImageNet and it is hard to train a model with high accuracy on such a complex dataset without enough data samples.

When training MobileNet on MNIST with non-IID data distribution, NetMax achieves a lower convergence rate compared to the other competitors, as shown in Fig. 18(a). However, the lost convergence rate does not surpass the improvement in converging time, as shown in Fig. 18(b). NetMax achieves  $2.45\times$ ,  $2.35\times$ , and  $1.39\times$  speedup in terms of converging time compared to Prague, Allreduce-SGD, and AD-PSGD, respectively. Moreover, the improvement in converging time achieved by NetMax can be enlarged when the model is larger or the network congestion is more serious, as shown in Fig. 13 and Fig. 12.

#### G. Distributed Training Across Clouds

We conducted experiments on a public cloud by training models across six Amazon EC2 regions, as summarized in Table VII. In each EC2 region, we created a  $c5.4\times large$

TABLE VII  
DATA DISTRIBUTION ACROSS CLOUD REGIONS

Regions	Lost labels	Regions	Lost labels
US West	0, 1, 2	Mumbai	4, 5, 6
US East	1, 2, 3	Singapore	5, 6, 7
Ireland	2, 3, 4	Tokyo	6, 7, 8

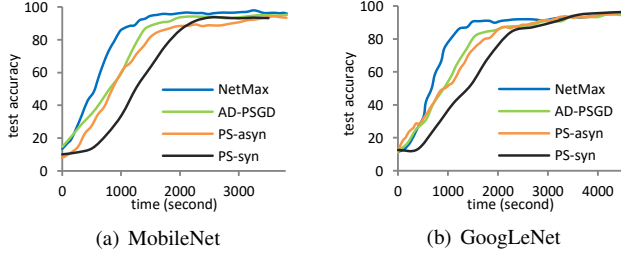


Fig. 19. Test accuracy varies with time. NetMax converges much faster over time compared to other approaches.

instance which has 16 vCPU and 32GB memory, running Ubuntu 16.04 LTS.

We train two CNN models (MobileNet and GoogLeNet) on the MNIST dataset to evaluate the performance of our proposed NetMax approach. The parameter numbers of MobileNet and GoogLeNet are approximately 4.2M and 6.8M, respectively. Since users in different regions may have different habits and, thus, different data distribution, we sort the training dataset by labels, and assign part of the labels to the worker nodes, as described in Table VII. In addition, each worker node has the complete test dataset.

The test accuracy of NetMax converges much faster over time compared to the other approaches, as shown in Fig. 19. We observe that parameter server with synchronous training (PS-syn) is the slowest since its training speed is determined by the slowest link connecting the worker nodes and the parameter server. Parameter server with asynchronous training (PS-asyn) significantly outperforms PS-syn on model training, but it is slightly slower than AD-PSGD, the state-of-the-art asynchronous decentralized PSGD. On the other hand, AD-PSGD is much slower than NetMax. In summary, NetMax converges  $1.9\times$ ,  $1.9\times$ , and  $2.1\times$  faster than AD-PSGD, PS-asyn and PS-syn, respectively.