Distributed Learning over Unreliable Networks

Chen Yu [1] **Hanlin Tang** [1] **Cedric Renggli** [2] **Simon Kassing** [2] **Ankit Singla** [2] **Dan Alistarh** [3] **Ce Zhang** [2] **Ji Liu** [4 1]

Abstract

Most of today's distributed machine learning systems assume *reliable networks*: whenever two machines exchange information (e.g., gradients or models), the network should guarantee the delivery of the message. At the same time, recent work exhibits the impressive tolerance of machine learning algorithms to errors or noise arising from relaxed communication or synchronization. In this paper, we connect these two trends, and consider the following question: *Can we design machine learning systems that are tolerant to network unreliability during training?* With this motivation, we focus on a theoretical problem of independent interest—given a standard distributed parameter server architecture, if every communication between the worker and the server has a non-zero probability p of being dropped, does there exist an algorithm that still converges, and at what speed? The technical contribution of this paper is a novel theoretical analysis proving that distributed learning over unreliable network can achieve comparable convergence rate to centralized or distributed learning over reliable networks. Further, we prove that the influence of the packet drop rate diminishes with the growth of the number of parameter servers. We map this theoretical result onto a real-world scenario, training deep neural networks over an unreliable network layer, and conduct network simulation to validate the system improvement by allowing the networks to be unreliable.

1. Introduction

Distributed learning has attracted significant interest from both academia and industry. Over the last decade, re-

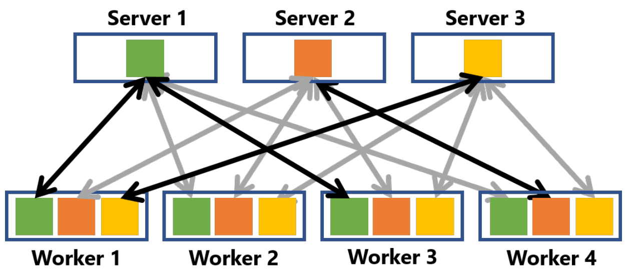


Figure 1: An illustration of the communication pattern of distributed learning with three parameter servers and four workers — each server serves a partition of the model, and each worker holds a replica of the whole model. In this paper, we focus on the case in which every communication between the worker and the server has a non-zero probability p of being dropped.

searchers have come with up a range of different designs of more efficient learning systems. An important subset of this work focuses on understanding the impact of different system relaxations to the convergence and performance of distributed stochastic gradient descent, such as the compression of communication, e.g (Seide & Agarwal, 2016), decentralized communication (Lian et al., 2017a; Sirb & Ye, 2016; Lan et al., 2017; Tang et al., 2018a; Stich et al., 2018), and asynchronous communication (Lian et al., 2017b; Zhang et al., 2013; Lian et al., 2015). Most of these works are motivated by real-world system bottlenecks, abstracted as general problems for the purposes of analysis.

In this paper, we focus on the centralized SGD algorithm in the distributed machine learning scenario implemented using standard AllReduce operator or parameter server architecture and we are motivated by a new type of system relaxation—the reliability of the communication channel. We abstract this problem as a theoretical one, conduct a novel convergence analysis for this scenario, and then validate our results in a practical setting.

The *Centralized SGD* algorithm works as Figure 1 shows. Given n machines/workers, each maintaining its own local model, each machine alternates local SGD steps with global communication steps, in which machines exchange their *local models*. In this paper, we covers two standard

[1]Department of Computer Science, University of Rochester, USA [2]Department of Computer Science, ETH Zurich [3]Institute of Science and Technology Austria [4]Seattle AI Lab, FeDA Lab, Kwai Inc. Correspondence to: Chen Yu <cyu28@ur.rochester.edu>.

Proceedings of the 36th International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019. Copyright 2019 by the author(s).

distributed settings: the Parameter Server model[1] (Li et al., 2014; Abadi et al., 2016), as well as standard implementations of the AllReduce averaging operation in a decentralized setting (Seide & Agarwal, 2016; Renggli et al., 2018). There are two components in the communication step:

1. **Step 0 - Model Partitioning (Only Conducted Once).** In most state-of-the-art implementations of AllReduce and parameter servers (Li et al., 2014; Abadi et al., 2016; Thakur et al., 2005), models are partitioned into n blocks, and each machine is the owner of one block (Thakur et al., 2005). The rationale is to increase parallelism over the utilization of the underlying communication network. The partitioning strategy does not change during training.

2. **Step 1.1 - Reduce-Scatter.** In the Reduce-Scatter step, for each block (model partition) i, the machines average their model on the block by sending it to the corresponding machine.

3. **Step 1.2 - All-Gather.** In the subsequent All-Gather step, each machine broadcasts its block to all others, so that all machines have a consistent model copy.

In this paper, we focus on the scenario in which the communication is unreliable — The communication channel between any two machines has a probability p of not delivering a message, as the Figure 1 shows, where the grey arrows represent the dropping message and the black arrows represent the success-delivering message. We change the two aggregation steps as follows. In the Reduce-Scatter step, a *uniform random* subset of machines will average their model on each model block i. In the All-Gather step, it is again a *uniform random* subset of machines which receive the resulting average. Specifically, machines not chosen for the Reduce-Scatter step do not contribute to the average, and all machines that are not chosen for the All-Gather will not receive updates on their model block i. This is a realistic model of running an AllReduce operator implemented with Reduce-Scatter/All-Gather on unreliable network. We call this revised algorithm the RPS algorithm.

Our main technical contribution is characterizing the convergence properties of the RPS algorithm. To the best of our knowledge, this is a novel theoretical analysis of this faulty communication model. We will survey related work in more details in Section 2.

We then apply our theoretical result to a real-world use case, illustrating the potential benefit of allowing an unreliable network. We focus on a realistic scenario where the network is shared among multiple applications or tenants, for instance in a data center. Both applications communicate using the same network. In this case, if the machine learning traffic is tolerant to some packet loss, the other application can potentially be made faster by receiving priority for its network traffic. Via network simulations, we find that tolerating a 10% drop rate for the learning traffic can make a simple (emulated) Web service up to $1.2\times$ faster (Even small speedups of 10% are significant for such services; for example, Google actively pursues minimizing its Web services' response latency). At the same time, this degree of loss does not impact the convergence rate for a range of machine learning applications, such as image classification and natural language processing.

Organization The rest of this paper is organized as follow. We first review some related work in Section 2. Then we formulate the problem in Section 3 and describe the RPS algorithm in Section 4, with its theoretical guarantee stated in Section 5. We evaluate the scalability and accuracy of the RPS algorithm in Section 6 and study an interesting case of speeding up colocated applications in Section 7. At last, we conclude the paper in Section 8. The proofs of our theoretical results can be found in the supplementary material.

2. Related Work

Distributed Learning There has been a huge number of works on distributing deep learning, e.g. Seide & Agarwal (2016); Abadi et al. (2016); Goyal et al. (2017); Colin et al. (2016). Also, many optimization algorithms are proved to achieve much better performance with more workers. For example, Hajinezhad et al. (2016) utilize a primal-dual based method for optimizing a finite-sum objective function and proved that it's possible to achieve a $\mathcal{O}(n)$ speedup corresponding to the number of the workers. In Xu et al. (2017), an adaptive consensus ADMM is proposed and Goldstein et al. (2016) studied the performance of transpose ADMM on an entire distributed dataset. In Scaman et al. (2017), the optimal convergence rate for both centralized and decentralized distributed learning is given with the time cost for communication included. In Lin et al. (2018); Stich (2018), they investigate the trade off between getting more mini-batches or having more communication. To save the communication cost, some sparse based distributed learning algorithms is proposed (Shen et al., 2018b; wu2, 2018; Wen et al., 2017; McMahan et al., 2016; Wang et al., 2016). Recent works indicate that many distributed learning is delay-tolerant under an asynchronous setting (Zhou et al., 2018; Lian et al., 2015; Sra et al., 2015; Leblond et al., 2016). Also, in Blanchard et al. (2017); Yin et al. (2018); Alistarh et al. (2018) They study the Byzantine-robust distributed learning when the Byzantine worker is included in the network. In Drumond et al. (2018), authors proposed a compressed DNN training strategy in order to save the computational cost of floating

[1]Our modeling above fits the case of n workers and n parameter servers, although our analysis will extend to any setting of these parameters.

point.

Centralized parallel training Centralized parallel (Agarwal & Duchi, 2011; Recht et al., 2011) training works on the network that is designed to ensure that all workers could get information of all others. One communication primitive in centralized training is to average/aggregate all models, which is called *a collective communication operator* in HPC literature (Thakur et al., 2005). Modern machine learning systems rely on different implementations, e.g., parameter server model (Li et al., 2014; Abadi et al., 2016) or the standard implementations of the AllReduce averaging operation in a decentralized setting (Seide & Agarwal, 2016; Renggli et al., 2018). In this work, we focus on the behavior of centralized ML systems under unreliable network, when this primitive is implemented as a distributed parameter servers (Jiang et al., 2017), which is similar to a Reduce-Scatter/All-Gather communication paradigm. For many implementations of collective communication operators, partitioning the model is one key design point to reach the peak communication performance (Thakur et al., 2005).

Decentralized parallel training Another direction of related work considers decentralized learning. Decentralized learning algorithms can be divided into *fixed* topology algorithms and *random* topology algorithms. There are many work related to the *fixed* topology decentralized learning. Specifically, Jin et al. (2016) proposes to scale the gradient aggregation process via a gossip-like mechanism. Lian et al. (2017a) provided strong convergence bounds for a similar algorithm to the one we are considering, in a setting where the communication graph is fixed and regular. In Tang et al. (2018b), a new approach that admits a better performance than decentralized SGD when the data among workers is very different is studied. Shen et al. (2018a) generalize the decentralized optimization problem to a monotone operator. In He et al. (2018), authors study a decentralized gradient descent based algorithm (**CoLA**) for learning of linear classification and regression model. For the *random* topology decentralized learning, the weighted matrix for randomized algorithms can be time-varying, which means workers are allowed to change the communication network based on the availability of the network. There are many works (Boyd et al., 2006; Li & Zhang, 2010; Lobel & Ozdaglar, 2011; Nedic et al., 2017; Nedić & Olshevsky, 2015) studying the random topology decentralized SGD algorithms under different assumptions. Blot et al. (2016) considers a more radical approach, called GoSGD, where each worker exchanges gradients with a random subset of other workers in each round. They show that GoSGD can be faster than Elastic Averaging SGD (Zhang et al., 2015) on CIFAR-10, but provide no large-scale experiments or theoretical justification. Recently, Daily et al. (2018) proposed GossipGrad, a more complex gossip-based scheme with upper bounds on the time for workers to communicate indirectly, periodic rotation of partners and shuffling of the input data, which provides strong empirical results on large-scale deployments. The authors also provide an informal justification for why GossipGrad should converge.

In this paper, we consider a general model communication, which covers both Parameter Server (Li et al., 2014) and AllReduce (Seide & Agarwal, 2016) distribution strategies. We specifically include the uncertainty of the network into our theoretical analysis. In addition, our analysis highlights the fact that the system can handle additional packet drops as we increase the number of worker nodes.

3. Problem Setup

We consider the following distributed optimization problem:

$$\min_{\boldsymbol{x}} \quad f(\boldsymbol{x}) = \frac{1}{n}\sum_{i=1}^{n} \underbrace{\mathbb{E}_{\boldsymbol{\xi}\sim\mathcal{D}_i} F_i(\boldsymbol{x};\boldsymbol{\xi})}_{=:f_i(\boldsymbol{x})}, \qquad (1)$$

where n is the number of workers, D_i is the local data distribution for worker i (in other words, we do not assume that all nodes can access the same data set), and $F_i(\boldsymbol{x};\boldsymbol{\xi})$ is the local loss function of model $\boldsymbol{x}$ given data $\boldsymbol{\xi}$ for worker i.

Unreliable Network Connection Nodes can communicate with all other workers, but with packet drop rate p (here we do not use the common-used phrase "packet loss rate" because we use "loss" to refer to the loss function). That means, whenever any node forwards models or data to any other model, the destination worker *fails* to receive it, with probability p. For simplicity, we assume that all packet drop events are independent, and that they occur with the same probability p.

Definitions and notations Throughout, we use the following notation and definitions:

- $\nabla f(\cdot)$ denotes the gradient of the function f.
- $\lambda_i(\cdot)$ denotes the ith largest eigenvalue of a matrix.
- $\mathbf{1}_n = [1, 1, \cdots, 1]^\top \in \mathbb{R}^n$ denotes the full-one vector.
- $A_n := \frac{\mathbf{1}_n\mathbf{1}_n^\top}{n}$ denotes the all $\frac{1}{n}$'s n by n matrix.
- $\|\cdot\|$ denotes the ℓ_2 norm for vectors.
- $\|\cdot\|_F$ denotes the Frobenius norm of matrices.

4. Algorithm

In this section, we describe our algorithm, namely RPS — *Reliable Parameter Server* — as it is robust to package loss in the network layer. We first describe our algorithm in detail, followed by its interpretation from a global view.

4.1. Our Algorithm: RPS

In the RPS algorithm, each worker maintains an individual local model. We use $\boldsymbol{x}_t^{(i)}$ to denote the local model on worker i at time step t. At each iteration t, each worker first performs a regular SGD step

$$\boldsymbol{v}_t^{(i)} \leftarrow \boldsymbol{x}_t^{(i)} - \gamma \nabla F_i(\boldsymbol{x}_t^{(i)}; \boldsymbol{\xi}_t^{(i)});$$

where γ is the learning rate and $\boldsymbol{\xi}_t^{(i)}$ are the data samples of worker i at iteration t.

We would like to reliably average the vector $\boldsymbol{v}_t^{(i)}$ among all workers, via the RPS procedure. In brief, the RS step perfors communication-efficient model averaging, and the AG step performs communication-efficient model sharing.

The Reduce-Scatter (RS) step: In this step, each worker i divides $\boldsymbol{v}_t^{(i)}$ into n equally-sized blocks.

$$\boldsymbol{v}_t^{(i)} = \left(\left(\boldsymbol{v}_t^{(i,1)}\right)^\top, \left(\boldsymbol{v}_t^{(i,2)}\right)^\top, \cdots, \left(\boldsymbol{v}_t^{(i,n)}\right)^\top \right)^\top. \quad (2)$$

The reason for this division is to reduce the communication cost and parallelize model averaging since we only assign each worker for averaging one of those blocks. For example, worker 1 can be assigned for averaging the first block while worker 2 might be assigned to deal with the third block. For simplicity, we would proceed our discussion in the case that worker i is assigned for averaging the ith block.

After the division, each worker sends its ith block to worker i. Once receiving those blocks, each worker would average all the blocks it receives. As noted, some packets might be dropped. In this case, worker i averages all those blocks using

$$\tilde{\boldsymbol{v}}_t^{(i)} = \frac{1}{|\mathcal{N}_t^{(i)}|} \sum_{j \in \mathcal{N}_t^{(i)}} \boldsymbol{v}_t^{(i,j)},$$

where $\mathcal{N}_t^{(i)}$ is the set of the packages worker i receives (may including the worker i's own package).

The AllGather (AG) step: After computing $\tilde{\boldsymbol{v}}_t^{(i)}$, each worker i attempts to broadcast $\tilde{\boldsymbol{v}}_t^{(i)}$ to all other workers, using the averaged blocks to recover the averaged original vector $\boldsymbol{v}_t^{(i)}$ by concatenation:

$$\boldsymbol{x}_{t+1}^{(i)} = \left(\left(\tilde{\boldsymbol{v}}_t^{(i,1)}\right)^\top, \left(\tilde{\boldsymbol{v}}_t^{(i,2)}\right)^\top, \cdots, \left(\tilde{\boldsymbol{v}}_t^{(i,n)}\right)^\top \right)^\top.$$

Note that it is entirely possible that some workers in the network may not be able to receive some of the averaged blocks. In this case, they just use the original block. Formally,

$$\boldsymbol{x}_{t+1}^{(i)} = \left(\left(\boldsymbol{x}_{t+1}^{(i,1)}\right)^\top, \left(\boldsymbol{x}_{t+1}^{(i,2)}\right)^\top, \cdots, \left(\boldsymbol{x}_{t+1}^{(i,n)}\right)^\top \right)^\top, \quad (3)$$

Algorithm 1 RPS

```
1:  Input: Initialize all $\boldsymbol{x}_1^{(i)}, \forall i \in [n]$ with the same value, learning rate $\gamma$, and number of total iterations $T$.
2:  for $t = 1, 2, \cdots, T$ do
3:      Randomly sample $\boldsymbol{\xi}_t^{(i)}$ from local data of the $i$th worker, $\forall i \in [n]$.
4:      Compute a local stochastic gradient based on $\boldsymbol{\xi}_t^{(i)}$ and current optimization variable $\boldsymbol{x}_t^{(i)}$ : $\nabla F_i(\boldsymbol{x}_t^{(i)}; \boldsymbol{\xi}_t^{(i)}), \forall i \in [n]$
5:      Compute the intermediate model $\boldsymbol{v}_t^{(i)}$ according to
            $$\boldsymbol{v}_t^{(i)} \leftarrow \boldsymbol{x}_t^{(i)} - \gamma \nabla F_i(\boldsymbol{x}_t^{(i)}; \boldsymbol{\xi}_t^{(i)}),$$
        and divide $\boldsymbol{v}_t^{(i)}$ into $n$ blocks $\left( \left(\boldsymbol{v}_t^{(i,1)}\right)^\top, \left(\boldsymbol{v}_t^{(i,2)}\right)^\top, \cdots, \left(\boldsymbol{v}_t^{(i,n)}\right)^\top \right)^\top$.
6:      For any $i \in [n]$, randomly choose one worker $b_t^{(i)}$ [a] without replacement. Then, every worker attempts to send their $i$th block of their intermediate model to worker $b_t^{(i)}$. Then each worker averages all received blocks using
            $$\tilde{\boldsymbol{v}}_t^{(i)} = \frac{1}{|\mathcal{N}_t^{(i)}|} \sum_{j \in \mathcal{N}_t^{(j)}} \boldsymbol{v}_t^{(i,j)}.$$
7:      Worker $b_t^{(i)}$ broadcast $\tilde{\boldsymbol{v}}_t^{(i)}$ to all workers (maybe dropped due to packet drop), $\forall i \in [n]$.
8:      $\boldsymbol{x}_{t+1}^{(i)} = \left( \left(\boldsymbol{x}_{t+1}^{(i,1)}\right)^\top, \left(\boldsymbol{x}_{t+1}^{(i,2)}\right)^\top, \cdots, \left(\boldsymbol{x}_{t+1}^{(i,n)}\right)^\top \right)^\top$,
        where
            $$\boldsymbol{x}_{t+1}^{(i,j)} = \begin{cases} \tilde{\boldsymbol{v}}_t^{(j)} & j \in \widetilde{\mathcal{N}}_t^{(i)} \\ \boldsymbol{v}_t^{(i,j)} & j \notin \widetilde{\mathcal{N}}_t^{(i)} \end{cases},$$
        for all $i \in [n]$.
9:  end for
10: Output: $\boldsymbol{x}_T^{(i)}$
```

[a]Here $b_t^{(i)} \in \{1, 2, \cdots, n\}$ indicates which worker is assigned for averaging the ith block.

where

$$\boldsymbol{x}_{t+1}^{(i,j)} = \begin{cases} \tilde{\boldsymbol{v}}_t^{(j)} & j \in \widetilde{\mathcal{N}}_t^{(i)} \\ \boldsymbol{v}_t^{(i,j)} & j \notin \widetilde{\mathcal{N}}_t^{(i)} \end{cases}$$

We can see that each worker just replace the corresponding blocks of $v_t^{(i)}$ using received averaged blocks. The complete algorithm is summarized in Algorithm 1.

4.2. RPS From a Global Viewpoint

It can be seen that at time step t, the jth block of worker i's local model, that is, $\boldsymbol{x}_t^{(i,j)}$, is a linear combination of jth block of all workers' intermediate model $\boldsymbol{v}_t^{(k,j)} (k \in [n])$,

$$X_{t+1}^{(j)} = V_t^{(j)} W_t^{(j)}, \tag{4}$$

where

$$\begin{aligned} X_{t+1}^{(j)} :=& \left(\boldsymbol{x}_{t+1}^{(1,j)}, \boldsymbol{x}_{t+1}^{(2,j)}, \cdots, \boldsymbol{x}_{t+1}^{(n,j)}\right) \\ V_t^{(j)} :=& \left(\boldsymbol{v}_t^{(1,j)}, \boldsymbol{v}_t^{(2,j)}, \cdots, \boldsymbol{v}_t^{(n,j)}\right) \end{aligned}$$

and $W_t^{(j)}$ is the coefficient matrix indicating the communication outcome at time step t. The (m,k)th element of $W_t^{(j)}$ is denoted by $\left[W_t^{(j)}\right]_{m,k}$. $\left[W_t^{(j)}\right]_{m,k} \neq 0$ means that worker k receives worker m's individual jth block (that is, $v_t^{(m,j)}$), whereas $\left[W_t^{(j)}\right]_{m,k} = 0$ means that the package might be dropped either in RS step (worker m fails to send) or AG step (worker k fails to receive). So $W_t^{(j)}$ is time-varying because of the randomness of the package drop. Also $W_t^{(j)}$ is not doubly-stochastic (in general) because the package drop is independent between RS step and AG step.

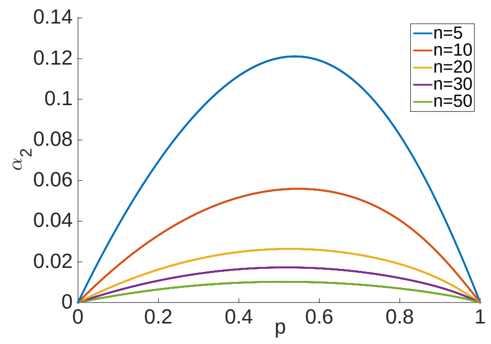


Figure 2: α_2 under different number of workers n and package drop rate p.

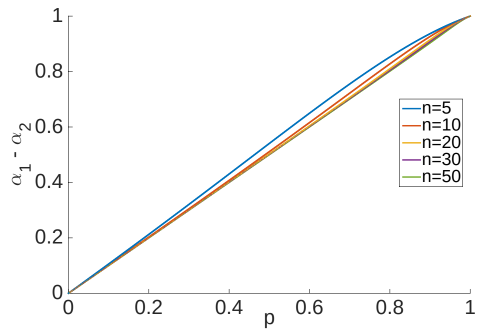


Figure 3: $(\alpha_1 - \alpha_2)$ under different number of workers n and package drop rate p.

The property of $W_t^{(j)}$ In fact, it can be shown that all $W_t^{(j)}$'s $(\forall j, \forall t)$ satisfy the following properties

$$\begin{aligned} \left(\mathbb{E}(W_t^{(j)})\right) A_n =& A_n \\ \mathbb{E}\left[W_t^{(j)} \left(W_t^{(j)}\right)^\top\right] =& \alpha_1 I_n + (1-\alpha_1) A_n \quad (5) \\ \mathbb{E}\left[W_t^{(j)} A_n \left(W_t^{(j)}\right)^\top\right] =& \alpha_2 I_n + (1-\alpha_2) A_n \quad (6) \end{aligned}$$

for some constants α_1 and α_2 satisfying $0 < \alpha_2 < \alpha_1 < 1$ (see Lemmas 6, 7, and 8 in Supplementary Material). Since the exact expression is too complex, we plot the α_1 and α_2 related to different n in Figure 2 and Figure 3 (detailed discussion is included in **Section D** in Supplementary Material.). Here, we do not plot α_2, but plot $\alpha_1 - \alpha_2$ instead. This is because $\alpha_1 - \alpha_2$ is an important factor in our Theorem (See Section 5 where we define $\alpha_1 - \alpha_2$ as β).

5. Theoretical Guarantees and Discussion

Below we show that, for certain parameter values, RPS with unreliable communication rates admits the same convergence rate as the standard algorithms. In other words, the impact of network unreliablity may be seen as negligible.

First let us make some necessary assumptions, that are commonly used in analyzing stochastic optimization algorithms.

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

1. ***Lipschitzian gradient:*** *All function $f_i(\cdot)$'s are with L-Lipschitzian gradients, which means*

$$\|\nabla f_i(\boldsymbol{x}) - \nabla f_i(\boldsymbol{y})\| \leq L\|\boldsymbol{x} - \boldsymbol{y}\|$$

2. ***Bounded variance:*** *Assume the variance of stochastic gradient*

$$\begin{aligned} \mathbb{E}_{\xi\sim\mathcal{D}_i} \|\nabla F_i(\boldsymbol{x};\xi) - \nabla f_i(\boldsymbol{x})\|^2 \leqslant& \sigma^2, \quad \forall i, \forall \boldsymbol{x}, \\ \frac{1}{n}\sum_{i=1}^n \|\nabla f_i(\boldsymbol{x}) - \nabla f(\boldsymbol{x})\|^2 \leqslant& \zeta^2, \quad \forall i, \forall \boldsymbol{x}, \end{aligned}$$

is bounded for any x in each worker i.

3. ***Start from 0:*** *We assume $X_1 = 0$ for simplicity w.l.o.g.*

Next we are ready to show our main result.

Theorem 1 (Convergence of Algorithm 1)**.** *Under Assumption 1, choosing γ in Algorithm 1 to be small enough that satisfies $1 - \frac{6L^2\gamma^2}{(1-\sqrt{\beta})^2} > 0$, we have the following convergence rate for Algorithm 1*

$$\frac{1}{T}\sum_{t=1}^T \left(\mathbb{E}\|\nabla f(\overline{\boldsymbol{x}}_t)\|^2 + (1 - L\gamma)\mathbb{E}\left\|\overline{\nabla} f(X_t)\right\|^2\right)$$

$$
\begin{aligned}
\leq & \frac{2f(\mathbf{0})-2f(\boldsymbol{x}^*)}{\gamma T}+\frac{\gamma L\sigma^2}{n}+4\alpha_2 L\gamma(\sigma^2+3\zeta^2)\\
&+\frac{\left(2\alpha_2 L\gamma+L^2\gamma^2+12\alpha_2 L^3\gamma^3\right)\sigma^2 C_1}{(1-\sqrt{\beta})^2}\\
&+\frac{3\left(2\alpha_2 L\gamma+L^2\gamma^2+12\alpha_2 L^3\gamma^3\right)\zeta^2 C_1}{(1-\sqrt{\beta})^2}, \qquad (7)
\end{aligned}
$$

where

$$
\begin{aligned}
\nabla f(\overline{\boldsymbol{x}}_t) =& f\left(\frac{1}{n}\sum_{i=1}^{n}\boldsymbol{x}_t^{(i)}\right),\\
\overline{\nabla} f(X_t) =& \sum_{i=1}^{n}\nabla f_i\left(\boldsymbol{x}_t^{(i)}\right),\\
\beta =& \alpha_1-\alpha_2,\\
C_1 =& \left(1-\frac{6L^2\gamma^2}{(1-\sqrt{\beta})^2}\right)^{-1},
\end{aligned}
$$

and α_1, α_2 follows the definition in (5) *and* (6).

To make the result more clear, we appropriately choose the learning rate as follows:

Corollary 2. *Choose $\gamma=\frac{1-\sqrt{\beta}}{6L+3(\sigma+\zeta)\sqrt{\alpha_2 T}+\frac{\sigma\sqrt{T}}{\sqrt{n}}}$ in Algorithm 1, under Assumption 1, we have the follow convergence rate for Algorithm 1*

$$
\begin{aligned}
\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left\|\nabla f(\overline{\boldsymbol{x}}_t)\right\|^2 \lesssim & \frac{(\sigma+\zeta)\left(1+\sqrt{n\alpha_2}\right)}{(1-\sqrt{\beta})\sqrt{nT}}+\frac{1}{T}\\
&+\frac{n(\sigma^2+\zeta^2)}{(1+n\alpha_2)\sigma^2 T+n\alpha_2 T\zeta^2},
\end{aligned}
$$

where β, α_1, α_2, $\nabla f(\overline{\boldsymbol{x}})$ follow to the definitions in Theorem 1, and we treat $f(0), f^*$, and L to be constants.

We discuss our theoretical results below

- (**Comparison with centralized SGD and decentralized SGD**) The dominant term in the convergence rate is $O(1/\sqrt{nT})$ (here we use $\alpha_2=\mathcal{O}(p(1-p)/n)$ and $\beta=\mathcal{O}(p)$ which is shown by Lemma 8 in Supplement), which is consistent with the rate for centralized SGD and decentralized SGD (Lian et al., 2017a).

- (**Linear Speedup**) Since the the leading term of convergence rate for $\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left\|\nabla f(\overline{\boldsymbol{x}}_t)\right\|^2$ is $\mathcal{O}(1/\sqrt{nT})$. It suggests that our algorithm admits the linear speedup property with respect to the number of workers n.

- (**Better performance for larger networks**) Fixing the package drop rate p (implicitly included in **Section D**), the convergence rate under a larger network (increasing n) would be superior, because the leading terms' dependence of the $\alpha_2=\mathcal{O}(p(1-p)/n)$. This indicates that the affection of the package drop ratio diminishes, as we increase the number of workers and parameter servers.

- (**Why only converges to a ball of a critical point**) This is because we use a constant learning rate, the algorithm could only converges to a ball centered at a critical point. This is a common choice to make the statement simpler, just like many other analysis for SGD. Our proved convergence rate is totally consistent with SGD's rate, and could converge (in the same rate) to a critical point by choosing a decayed learning rate such as $O(1/\sqrt{T})$ like SGD.

6. Experiments: Convergence of RPS

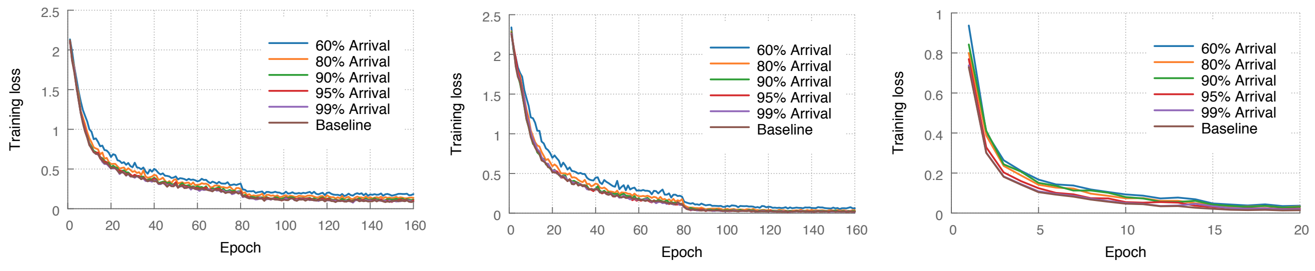
We now validate empirically the scalability and accuracy of the RPS algorithm, given reasonable message arrival rates.

6.1. Experimental Setup

Datasets and models We evaluate our algorithm on two state of the art machine learning tasks: (1) image classification and (2) natural language understanding (NLU). We train ResNet (He et al., 2016) with different number of layers on CIFAR-10 (Krizhevsky & Hinton, 2009) for classifying images. We perform the NLU task on the Air travel information system (ATIS) corpus on a one layer LSTM network.

Implementation We simulate packet losses by adapting the latest version 2.5 of the Microsoft Cognitive Toolkit (Seide & Agarwal, 2016). We implement the RPS algorithm using MPI. During training, we use a local batch size of 32 samples per worker for image classification. We adjust the learning rate by applying a linear scaling rule (Goyal et al., 2017) and decay of 10 percent after 80 and 120 epochs, respectively. To achieve the best possible convergence, we apply a gradual warmup strategy (Goyal et al., 2017) during the first 5 epochs. We deliberately do not use any regularization or momentum during the experiments in order to be consistent with the described algorithm and proof. The NLU experiments are conducted with the default parameters given by the CNTK examples, with scaling the learning rate accordingly, and omit momentum and regularization terms on purpose. The training of the models is executed on 16 NVIDIA TITAN Xp GPUs. The workers are connected by Gigabit Ethernet. We use each GPU as a worker. We describe the results in terms of training loss convergence, although the validation trends are similar.

Convergence of Image Classification We perform convergence tests using the analyzed algorithm, model averaging SGD, on both ResNet110 and ResNet20 with CIFAR-10. Figure 4(a,b) shows the result. We vary probabilities for each packet being correctly delivered at each worker be-



(a) ResNet20 - CIFAR10 (b) ResNet110 - CIFAR10 (c) LSTM - ATIS

Figure 4: Convergence of RPS on different datasets.

tween 80%, 90%, 95% and 99%. The baseline is 100% message delivery rate. The baseline achieves a training loss of 0.02 using ResNet110 and 0.09 for ResNet20. Dropping 1% doesn't increase the training loss achieved after 160 epochs. For 5% the training loss is identical on ResNet110 and increased by 0.02 on ResNet20. Having a probability of 90% of arrival leads to a training loss of 0.03 for ResNet110 and 0.11 for ResNet20 respectively.

Convergence of NLU We perform full convergence tests for the NLU task on the ATIS corpus and a single layer LSTM. Figure 4(c) shows the result. The baseline achieves a training loss of 0.01. Dropping 1, 5 or 10 percent of the communicated partial vectors result in an increase of 0.01 in training loss.

Comparison with Gradient Averaging We conduct experiments with identical setup and a probability of 99 percent of arrival using a *gradient* averaging methods, instead of model averaging. When running data distributed SGD, gradient averaging is the most widely used technique in practice, also implemented by default in most deep learning frameworks(Abadi et al., 2016; Seide & Agarwal, 2016). As expected, the baseline (all the transmissions are successful) convergences to the same training loss as its model averaging counterpart, when omitting momentum and regularization terms. As seen in figures 5(a,b), having a loss in communication of even 1 percentage results in worse convergence in terms of accuracy for both ResNet architectures on CIFAR-10. The reason is that the error of package drop will accumulate over iterations but never decay, because the model is the sum of all early gradients, so the model never converges to the optimal one. Nevertheless, this insight suggests that one should favor a model averaging algorithm over gradient averaging, if the underlying network connection is unreliable.

Extended Analysis We report additional experiments with a brief analysis of the generalization properties regarding the test accuracy in Section E in the supplementary material. We test a wider range of probabilities for each packet being correctly delivered, and finally analyze the impact of varying number of nodes at a fixed arrival rate. We use the default hyperparameter values given by the deep learning frameworks for all the experiments conducted. As a further work, one might envisage tuning the hyper-parameters such as learning rate for a given number of nodes and arrival probability.

7. Case study: Speeding up Colocated Applications

Our results on the resilience of distributed learning to losses of model updates open up an interesting use case. That model updates can be lost (within some tolerance) without the deterioration of model convergence implies that model updates transmitted over the physical network can be de-prioritized compared to other more "inflexible," delay-sensitive traffic, such as for Web services. Thus, we can colocate other applications with the training workloads, and reduce infrastructure costs for running them. Equivalently, workloads that are colocated with learning workers can benefit from prioritized network traffic (at the expense of some model update losses), and thus achieve lower latency.

To demonstrate this in practice, we perform a packet simulation over 16 servers, each connected with a 1 Gbps link to a network switch. Over this network of 16 servers, we run two workloads: (a) replaying traces from the machine learning process of ResNet110 on CIFAR-10 (which translates to a load of 2.4 Gbps) which is sent *unreliably*, and (b) a simple emulated Web service running on all 16 servers. Web services often produce significant background traffic between servers within the data center, consisting typically of small messages fetching distributed pieces of content to compose a response (e.g., a Google query response potentially consists of advertisements, search results, and images). We emulate this intra data center traffic for the Web service as all-to-all traffic between these servers, with small messages of 100 KB (a reasonable size for such services) sent reliably between these servers. The inter-arrival time for these messages follows a Poisson process, parametrized by the expected message rate, λ (aggregated across the 16 servers).

Different degrees of prioritization of the Web service traffic over learning traffic result in different degrees of loss in

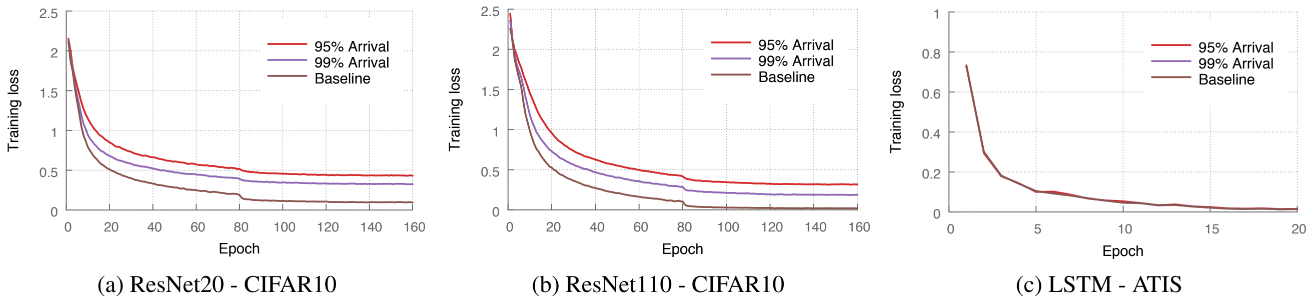


Figure 5: Why RPS? The Behavior of Standard SGD in the Presence of Message Drop.

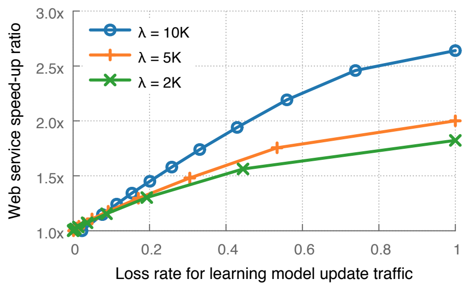


Figure 6: Allowing an increasing rate of losses for model updates speeds up the Web service.

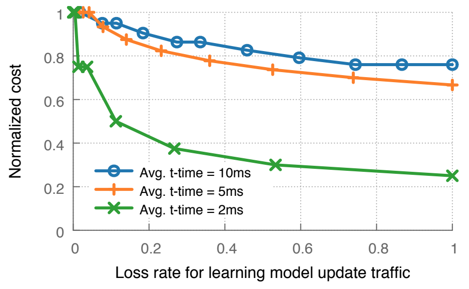


Figure 7: Allowing more losses for model updates reduces the cost for the Web service.

learning updates transmitted over the network. As the Web service is prioritized to a greater extent, its performance improves – its message exchanges take less time; we refer to this reduction in (average) completion time for these messages as a speed-up. Note that even small speedups of 10% are significant for such services; for example, Google actively pursues minimizing its Web services' response latency. An alternative method of quantifying the benefit for the colocated Web service is to measure how many additional messages the Web service can send, while maintaining a fixed average completion time. This translates to running more Web service queries and achieving more throughput over the same infrastructure, thus reducing cost per request / message.

Fig. 6 and Fig. 7 show results for the above described Web service speedup and cost reduction respectively. In Fig. 6, the arrival rate of Web service messages is fixed ($\lambda = \{2000, 5000, 10000\}$ per second). As the network prioritizes the Web service more and more over learning update traffic, more learning traffic suffers losses (on the x-axis), but performance for the Web service improves. With just 10% losses for learning updates, the Web service can be sped up by more than 20% (*i.e.,* $1.2\times$).

In Fig. 7, we set a target average transmission time (2, 5, or 10 ms) for the Web service's messages, and increase the message arrival rate, λ, thus causing more and more losses for learning updates on the x-axis. But accommodating higher λ over the same infrastructure translates to a lower cost of running the Web service (with this reduction shown on the y-axis).

Thus, tolerating small amounts of loss in model update traffic can result in significant benefits for colocated services, while not deteriorating convergence.

8. Conclusion

In this paper, we present a novel analysis for a general model of distributed machine learning, under a realistic unreliable communication model. We present a novel theoretical analysis for such a scenario, and evaluated it while training neural networks on both image and natural language datasets. We also provided a case study of application collocation, to illustrate the potential benefit that can be provided by allowing learning algorithms to take advantage of unreliable communication channels.

Acknowledgements

Chen Yu, Hanlin Tang, and Ji Liu are in part supported by NSF CCF1718513, IBM faculty award, and NEC fellowship. Ce Zhang and the DS3Lab gratefully acknowledge the support from the Swiss National Science Foundation (Project Number 200021_184628), Swiss Data Science Center, Google Focused Research Award, Huawei, MeteoSwiss, Oracle Labs, Swisscom, Zurich Insurance, and the Chinese Scholarship Council.

References

Error compensated quantized sgd and its applications to large-scale distributed optimization. *arXiv preprint arXiv:1806.08054*, 2018.

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. Tensorflow: A system for large-scale machine learning. In *OSDI*, volume 16, pp. 265–283, 2016.

Agarwal, A. and Duchi, J. C. Distributed delayed stochastic optimization. In *Advances in Neural Information Processing Systems*, pp. 873–881, 2011.

Alistarh, D., Allen-Zhu, Z., and Li, J. Byzantine stochastic gradient descent. *arXiv preprint arXiv:1803.08917*, 2018.

Blanchard, P., Guerraoui, R., Stainer, J., et al. Machine learning with adversaries: Byzantine tolerant gradient descent. In *Advances in Neural Information Processing Systems*, pp. 119–129, 2017.

Blot, M., Picard, D., Cord, M., and Thome, N. Gossip training for deep learning. *arXiv preprint arXiv:1611.09726*, 2016.

Boyd, S., Ghosh, A., Prabhakar, B., and Shah, D. Randomized gossip algorithms. *IEEE transactions on information theory*, 52(6):2508–2530, 2006.

Colin, I., Bellet, A., Salmon, J., and Clémençon, S. Gossip dual averaging for decentralized optimization of pairwise functions. *arXiv preprint arXiv:1606.02421*, 2016.

Daily, J., Vishnu, A., Siegel, C., Warfel, T., and Amatya, V. Gossipgrad: Scalable deep learning using gossip communication based asynchronous gradient descent. *arXiv preprint arXiv:1803.05880*, 2018.

Drumond, M., LIN, T., Jaggi, M., and Falsafi, B. Training dnns with hybrid block floating point. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 31*, pp. 451–461. Curran Associates, Inc., 2018.

Goldstein, T., Taylor, G., Barabin, K., and Sayre, K. Unwrapping admm: efficient distributed computing via transpose reduction. In *Artificial Intelligence and Statistics*, pp. 1151–1158, 2016.

Goyal, P., Dollár, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., Tulloch, A., Jia, Y., and He, K. Accurate, large minibatch sgd: training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

Hajinezhad, D., Hong, M., Zhao, T., and Wang, Z. Nestt: A nonconvex primal-dual splitting method for distributed and stochastic optimization. In *Advances in Neural Information Processing Systems*, pp. 3215–3223, 2016.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

He, L., Bian, A., and Jaggi, M. Cola: Decentralized linear learning. In *Advances in Neural Information Processing Systems*, pp. 4541–4551, 2018.

Jiang, J., Cui, B., Zhang, C., and Yu, L. Heterogeneity-aware distributed parameter servers. In *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD '17, pp. 463–478, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-4197-4. doi: 10.1145/3035918.3035933. URL `http://doi.acm.org/10.1145/3035918.3035933`.

Jin, P. H., Yuan, Q., Iandola, F., and Keutzer, K. How to scale distributed deep learning? *arXiv preprint arXiv:1611.04581*, 2016.

Krizhevsky, A. and Hinton, G. Learning multiple layers of features from tiny images. 2009.

Lan, G., Lee, S., and Zhou, Y. Communication-efficient algorithms for decentralized and stochastic optimization. *arXiv preprint arXiv:1701.03961*, 2017.

Leblond, R., Pedregosa, F., and Lacoste-Julien, S. Asaga: asynchronous parallel saga. *arXiv preprint arXiv:1606.04809*, 2016.

Li, M., Andersen, D. G., Park, J. W., Smola, A. J., Ahmed, A., Josifovski, V., Long, J., Shekita, E. J., and Su, B.-Y. Scaling distributed machine learning with the parameter server. In *OSDI*, volume 14, pp. 583–598, 2014.

Li, T. and Zhang, J.-F. Consensus conditions of multi-agent systems with time-varying topologies and stochastic communication noises. *IEEE Transactions on Automatic Control*, 55(9):2043–2057, 2010.

Lian, X., Huang, Y., Li, Y., and Liu, J. Asynchronous parallel stochastic gradient for nonconvex optimization. In *Advances in Neural Information Processing Systems*, pp. 2737–2745, 2015.

Lian, X., Zhang, C., Zhang, H., Hsieh, C.-J., Zhang, W., and Liu, J. Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent. In *Advances in Neural Information Processing Systems*, pp. 5336–5346, 2017a.

Lian, X., Zhang, W., Zhang, C., and Liu, J. Asynchronous decentralized parallel stochastic gradient descent. *arXiv preprint arXiv:1710.06952*, 2017b.

Lin, T., Stich, S. U., and Jaggi, M. Don't use large mini-batches, use local sgd. *arXiv preprint arXiv:1808.07217*, 2018.

Lobel, I. and Ozdaglar, A. Distributed subgradient methods for convex optimization over random networks. *IEEE Transactions on Automatic Control*, 56(6):1291, 2011.

McMahan, H. B., Moore, E., Ramage, D., Hampson, S., et al. Communication-efficient learning of deep networks from decentralized data. *arXiv preprint arXiv:1602.05629*, 2016.

Nedić, A. and Olshevsky, A. Distributed optimization over time-varying directed graphs. *IEEE Transactions on Automatic Control*, 60(3):601–615, 2015.

Nedic, A., Olshevsky, A., and Shi, W. Achieving geometric convergence for distributed optimization over time-varying graphs. *SIAM Journal on Optimization*, 27(4): 2597–2633, 2017.

Recht, B., Re, C., Wright, S., and Niu, F. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in neural information processing systems*, pp. 693–701, 2011.

Renggli, C., Alistarh, D., and Hoefler, T. Sparcml: High-performance sparse communication for machine learning. *arXiv preprint arXiv:1802.08021*, 2018.

Scaman, K., Bach, F., Bubeck, S., Lee, Y. T., and Massoulié, L. Optimal algorithms for smooth and strongly convex distributed optimization in networks. *arXiv preprint arXiv:1702.08704*, 2017.

Seide, F. and Agarwal, A. Cntk: Microsoft's open-source deep-learning toolkit. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 2135–2135. ACM, 2016.

Shen, Z., Mokhtari, A., Zhou, T., Zhao, P., and Qian, H. Towards more efficient stochastic decentralized learning: Faster convergence and sparse communication. In Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 4624–4633, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018a. PMLR. URL `http://proceedings.mlr.press/v80/shen18a.html`.

Shen, Z., Mokhtari, A., Zhou, T., Zhao, P., and Qian, H. Towards more efficient stochastic decentralized learning: Faster convergence and sparse communication. *arXiv preprint arXiv:1805.09969*, 2018b.

Sirb, B. and Ye, X. Consensus optimization with delayed and stochastic gradients on decentralized networks. In *Big Data (Big Data), 2016 IEEE International Conference on*, pp. 76–85. IEEE, 2016.

Sra, S., Yu, A. W., Li, M., and Smola, A. J. Adadelay: Delay adaptive distributed stochastic convex optimization. *arXiv preprint arXiv:1508.05003*, 2015.

Stich, S. U. Local sgd converges fast and communicates little. *arXiv preprint arXiv:1805.09767*, 2018.

Stich, S. U., Cordonnier, J.-B., and Jaggi, M. Sparsified sgd with memory. In *Advances in Neural Information Processing Systems*, pp. 4452–4463, 2018.

Tang, H., Gan, S., Zhang, C., Zhang, T., and Liu, J. Communication compression for decentralized training. In *Advances in Neural Information Processing Systems*, pp. 7663–7673, 2018a.

Tang, H., Lian, X., Yan, M., Zhang, C., and Liu, J. D2: Decentralized training over decentralized data. *arXiv preprint arXiv:1803.07068*, 2018b.

Thakur, R., Rabenseifner, R., and Gropp, W. Optimization of collective communication operations in mpich. *Int. J. High Perform. Comput. Appl.*, 19(1):49–66, February 2005. ISSN 1094-3420. doi: 10.1177/1094342005051521. URL `http://dx.doi.org/10.1177/1094342005051521`.

Wang, J., Kolar, M., Srebro, N., and Zhang, T. Efficient distributed learning with sparsity. *arXiv preprint arXiv:1605.07991*, 2016.

Wen, W., Xu, C., Yan, F., Wu, C., Wang, Y., Chen, Y., and Li, H. Terngrad: Ternary gradients to reduce communication in distributed deep learning. In *Advances in neural information processing systems*, 2017.

Xu, Z., Taylor, G., Li, H., Figueiredo, M., Yuan, X., and Goldstein, T. Adaptive consensus admm for distributed optimization. *arXiv preprint arXiv:1706.02869*, 2017.

Yin, D., Chen, Y., Ramchandran, K., and Bartlett, P. Byzantine-robust distributed learning: Towards optimal statistical rates. *arXiv preprint arXiv:1803.01498*, 2018.

Zhang, S., Zhang, C., You, Z., Zheng, R., and Xu, B. Asynchronous stochastic gradient descent for dnn training. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pp. 6660–6663. IEEE, 2013.

Zhang, S., Choromanska, A. E., and LeCun, Y. Deep learning with elastic averaging sgd. In *Advances in Neural Information Processing Systems*, pp. 685–693, 2015.

Zhou, Z., Mertikopoulos, P., Bambos, N., Glynn, P., Ye, Y., Li, L.-J., and Fei-Fei, L. Distributed asynchronous optimization with unbounded delays: How slow can you go? In *International Conference on Machine Learning*, pp. 5965–5974, 2018.