Faster Distributed Deep Net Training: Computation and Communication Decoupled Stochastic Gradient Descent

Shuheng Shen[1,2], **Linli Xu**[1,2,*], **Jingchang Liu**[1,2], **Xianfeng Liang**[1,2] and **Yifei Cheng**[1,3]
[1]Anhui Province Key Laboratory of Big Data Analysis and Application
[2]School of Computer Science and Technology, University of Science and Technology of China
[3]School of Data Science, University of Science and Technology of China
vaip@mail.ustc.edu.cn, linlixu@ustc.edu.cn, {xdjcl, zeroxf, chengyif}@mail.ustc.edu.cn

Abstract

With the increase in the amount of data and the expansion of model scale, distributed parallel training becomes an important and successful technique to address the optimization challenges. Nevertheless, although distributed stochastic gradient descent (SGD) algorithms can achieve a linear iteration speedup, they are limited significantly in practice by the communication cost, making it difficult to achieve a linear time speedup. In this paper, we propose a computation and communication decoupled stochastic gradient descent (CoCoD-SGD) algorithm to run computation and communication in parallel to reduce the communication cost. We prove that CoCoD-SGD has a linear iteration speedup with respect to the total computation capability of the hardware resources. In addition, it has a lower communication complexity and better time speedup comparing with traditional distributed SGD algorithms. Experiments on deep neural network training demonstrate the significant improvements of CoCoD-SGD: when training ResNet18 and VGG16 with 16 Geforce GTX 1080Ti GPUs, CoCoD-SGD is up to 2-3$\times$ faster than traditional synchronous SGD.

1 Introduction

The training of deep neural networks is resource intensive and time-consuming. With the expansion of data and model scale, it may take a few days or weeks to train a deep model by using mini-batch SGD on a single machine/GPU. To accelerate the training process, distributed optimization provides an effective tool for deep net training by allocating the computation to multiple computing devices (CPUs or GPUs).

When variants of mini-batch SGD are applied to a distributed system, communication between computing devices will be incurred to keep the same convergence rate as mini-batch SGD. As a matter of fact, the extra communication cost in a distributed system is the main factor which prevents a distributed optimization algorithm from achieving the linear time speedup, although the computation load is the same as its single machine version. In addition, the communication cost, which is often linearly proportional to the number of workers, can be extremely expensive when the number of workers is huge. Therefore, it is critical to reduce the communication bottleneck to make better use of the hardware resources.

Given that the total amount of communication bits equals the number of communications multiplied by the number of bits per communication, several works are proposed to accelerate training by reducing the communication frequency [Zhou and Cong, 2018; Yu *et al.*, 2018; Stich, 2018] or communication bits [Lin *et al.*, 2017; Stich *et al.*, 2018; Wen *et al.*, 2017; Alistarh *et al.*, 2017]. However, even when the communication frequency or the number of bits per communication is reduced, hardware resources are not fully exploited in traditional synchronous distributed algorithms because of the following two reasons: (1) only partial resources can be used when workers are communicating with each other and (2) the computation and the communication are interdependent in each iteration. Specifically, the computation of the t-th step relies on the communication of the $(t-1)$-th step and the communication in the t-th step follows the computation of the t-th step. To address that, another line of research aims to run computation and communication in parallel by using stale gradients [Li *et al.*, 2018], but the communication complexity is still high as all gradients need to be communicated.

To tackle this dilemma, we propose computation and communication decoupled stochastic gradient descent (CoCoD-SGD) for distributed training. Instead of waiting for the completion of communication as in traditional synchronous SGD, workers in CoCoD-SGD continue to calculate stochastic gradients and update models locally after the communication started. After the communication finishes, each worker updates its local model with the result of communication and the difference of the local models. As a result, CoCoD-SGD can make full use of resources by running computation and communication in parallel. In the meantime, workers in CoCoD-SGD communicate with each other periodically instead of in each iteration. As a benefit, the communication complexity of CoCoD-SGD is lower. By running computation and communication in parallel while reducing the communication complexity, faster distributed training is achieved. In addition, CoCoD-SGD is suitable for both homogeneous and heterogeneous environments. Further, it is theoretically justified with a linear iteration speedup with respect to the total computa-

*The corresponding author.

tion capability of hardware devices.

Contributions of this paper are summarized as follows:

- We propose CoCoD-SGD, a computation and communication decoupled distributed algorithm, to make full use of hardware resources and reduce the communication complexity.
- From the theoretical perspective, we prove that CoCoD-SGD has a linear iteration speedup with respect to the number of workers in a homogeneous environment. Besides, it has a theoretically justified linear iteration speedup with respect to the total computation capability in a heterogeneous environment.
- When training deep neural networks with multiple GPUs, CoCoD-SGD achieves a better time speedup comparing with existing distributed algorithms.
- In both homogeneous and heterogeneous environments, experimental results verify the effectiveness of the proposed algorithm.

2 Related Work

A conventional framework for distributed training is the centralized parameter server architecture [Li *et al.*, 2014], which is supported by most existing systems such as Tensorflow [Abadi *et al.*, 2016], Pytorch [Paszke *et al.*, 2017] and Mxnet [Chen *et al.*, 2015]. In each iteration, the parameter server, which holds the global model, needs to communicate with $O(N)$ workers. This becomes a bottleneck which slows down the convergence when N is large. Therefore, the decentralized ring architecture using the Ring-AllReduce algorithm became popular in recent years. In Ring-AllReduce, each worker only transports $O(1)$ gradients to its neighbors to get the average model over all workers.

The Ring-AllReduce algorithm can be directly combined with synchronous stochastic gradient descent (S-SGD), which has a theoretically justified convergence rate $O(\frac{1}{\sqrt{TM}})$ for both general convex [Dekel *et al.*, 2012] and non-convex problems [Ghadimi and Lan, 2013], where T is the number of iterations and M is the mini-batch size. Such rate shows its linear iteration speedup with respect to the number of workers since increasing the number of workers is equivalent to increasing the mini-batch size. Although only $O(1)$ gradients are needed to be communicated per worker in each iteration for Ring-AllReduce, $O(N)$ handshakes are needed for each worker. Therefore, the communication cost grows as the number of the workers increases.

There have been many attempts to reduce the communication bottleneck of S-SGD while maintaining its linear iteration speedup property. Among them, Local-SGD is a variant of S-SGD with low communication frequency, in which workers update their models locally and communicate with each other every k iterations. It has been proved to have linear iteration speedup for both strongly convex [Stich, 2018] and non-convex [Zhou and Cong, 2018; Yu *et al.*, 2018] problems. QSGD [Alistarh *et al.*, 2017] and TernGrad [Wen *et al.*, 2017] compress the gradients from 32-bit float to lower bit representations. Sparse SGD [Aji and Heafield, 2017] is another method that communicates part of the gradients in each iteration and is proved to have the same convergence rate as SGD [Stich *et al.*, 2018; Alistarh *et al.*, 2018]. Although the communication complexity is reduced in the above methods, the hardware resources are not fully utilized because only a part of them is used for communication.

To fully utilize hardware resources, asynchronous stochastic gradient descent (A-SGD), a distributed variant of SGD based on the parameter server architecture, is proposed. After one communication started, a worker uses a stale model to compute the next stochastic gradient without waiting for the completion of the communication. A linear iteration speedup is proved for both convex [Agarwal and Duchi, 2011] and non-convex [Lian *et al.*, 2015] problems. However, stale gradients may slow down the training and make it converge to a poor solution especially when the number of workers is large, which implies a large delay [Chen *et al.*, 2016]. As a remedy, Pipe-SGD [Li *et al.*, 2018] is proposed to integrate the advantages of A-SGD and Ring-AllReduce. Specifically, it employs Ring-AllReduce to get the average gradient and updates the model with a stale gradient. Pipe-SGD can control the delay as a constant because of the efficiency of the Ring-AllReduce algorithm. Nevertheless, one disadvantage of Pipe-SGD is that its communication complexity is $O(T)$, which is higher than Local-SGD and QSGD. In comparison, the proposed algorithm which makes full use of hardware resources and has a lower communication complexity, achieves the advantages of both Pipe-SGD and Local-SGD.

3 Algorithm

In this section, we introduce the CoCoD-SGD algorithm with the following three techniques: (1) computation and communication decoupling, (2) periodically communicating and (3) proportionally sampling.

3.1 Preliminary

Problem Definition

We focus on data-parallel distributed training, in which each worker can access only part of the data. We use $\mathcal{D}$ to denote the full training dataset and $\mathcal{D}_i$ to denote the local dataset stored in the i-th worker. We have $\mathcal{D} = \mathcal{D}_1 \cup \cdots \cup \mathcal{D}_N$ and $\mathcal{D}_i \cap \mathcal{D}_j = \varnothing, \forall i \neq j$. The objective can be written as

$$\min_{x \in \mathbb{R}^d} f(x) := \frac{1}{|\mathcal{D}|} \sum_{\xi \in \mathcal{D}} f(x, \xi) = \sum_{i=1}^{N} p_i f_i(x), \qquad (1)$$

where $f_i(x) := \frac{1}{|\mathcal{D}_i|} \sum_{\xi_i \in \mathcal{D}_i} f(x, \xi_i)$ is the local loss of the i-th worker and p_i's define the partition of data among all workers. Specifically, p_i is proportional to size of the local dataset on the i-th worker: $p_i = \frac{|\mathcal{D}_i|}{|\mathcal{D}|}$ and we have $\sum_{i=1}^{N} p_i = 1$. We use $\mathcal{P}$ to denote the distribution of p_i's.

Notations

- $\|\cdot\|$ indicates the ℓ_2 norm of a vector.
- f^* represents the optimal value of (1).
- $\mathbb{E}$ indicates a full expectation with respect to all the randomness, which includes the random indexes sampled to calculate stochastic gradients in all iterations.
- $\mathcal{C}_i$ represents the computation capability, which indicates the computing speed, of the i-th worker.

Algorithm 1 CoCoD-SGD

Input: The number of workers N, the number of iterations T, communication period k, mini-batch sizes $M_1, \cdots, M_N$ and initial point $\hat{x}_0 \in \mathbb{R}^d$.
Initialize: $t = 0, x_i^0 = \hat{x}_0, i = 1, 2, \cdots, N$.

1: **while** $t < T$ **do**
2: Worker W_i does:
3: Run **Step I** and **Step II** in parallel.
4: **Step I** : Store x_t^i in the memory and communicate with other workers to get the weighted mean of all local models: $\hat{x}_t = \sum_{i=1}^N \frac{M_i}{\sum_{j=1}^N M_j} x_t^i$.
5: **Step II** :
for $\tau = t$ to $t + k - 1$ **do**
Compute a mini-batch stochastic gradient $G_\tau^i = \frac{1}{M_i} \sum_{j=1}^{M_i} \nabla f_i(x_\tau^i, \xi_\tau^{i,j}), \xi_\tau^{i,j} \in \mathcal{D}_i$, and update locally: $x_{\tau+1}^i = x_\tau^i - \gamma G_\tau^i$
end for
6: $x_{t+k}^i = \hat{x}_t + (x_{t+k}^i - x_t^i)$
7: $t = t + k$
8: **end while**

3.2 Computation and Communication Decoupled Stochastic Gradient Descent

The complete procedure of CoCoD-SGD is summarized in Algorithm 1. On one hand, the goal of CoCoD-SGD is two-fold: (1) running computation and communication in parallel and (2) reducing the communication complexity. To achieve that, we propose the following two techniques:

- **Computation and communication decoupling:** Different from Pipe-SGD which uses stale gradients to decouple the dependency of computation and communication, CoCoD-SGD continues to update models in workers locally after the communication starts. If one communication starts after the t-th update, workers need to communicate with each other to get the average model $\hat{x}_t$. After the communication starts, the i-th worker continues to update its local model x_t^i by mini-batch SGD (line 4-5). As a result, computation and communication can be executed simultaneously.
- **Periodically communicating:** In practice, when the number of workers is large, the communication time may exceed the time of one update, which will cause idle time of the computing devices. Therefore, we let workers keep on updating the local models k times instead of waiting for the completion of the communication after only one local update (line 5). In this way, the communication complexity can be also reduced.

On the other hand, CoCoD-SGD is designed to be suitable for heterogeneous distributed system, where workers may have different computation capabilities. Intuitively, when all workers use the same batch size to compute stochastic gradients, the faster workers need to wait for the slower ones after finishing their updates. Therefore, we use proportionally sampling in CoCoD-SGD:

- **Proportionally sampling:** Workers in CoCoD-SGD use different mini-batch sizes M_i's to compute stochastic gradients. The batch sizes are proportional to the computation capabilities of workers, i.e., $\frac{M_i}{M_j} = \frac{\mathcal{C}_i}{\mathcal{C}_j}$. Under this setting, all workers will finish k updates at the same time. Meanwhile, to let all workers finish one epoch simultaneously, we proportionally divide the dataset among workers, i.e., $\frac{|\mathcal{D}_i|}{|\mathcal{D}_j|} = \frac{\mathcal{C}_i}{\mathcal{C}_j}$. In addition, we define the average model $\hat{x}_t$ as the weighted mean of all local models: $\hat{x}_t = \sum_{i=1}^N \frac{M_i}{\sum_{j=1}^N M_j} x_t^i$ (line 4).

Besides, different from QSGD [Alistarh *et al.*, 2017] and Sparse-SGD [Stich *et al.*, 2018], which are based on the parameter server architecture, CoCoD-SGD can communicate with the Ring-AllReduce algorithm by the definition of $\hat{x}^t$.

After finishing local iterations and the communication, the i-th worker updates its local model with the average model and the difference of the local models by $x_{t+k}^i = \hat{x}_t + (x_{t+k}^i - x_t^i)$ (line 6).

4 Theoretical Analysis

In this section, we provide the theoretical analysis for CoCoD-SGD and show that CoCoD-SGD has the same convergence rate as S-SGD. In addition, we show that CoCoD-SGD has lower communication complexity and better time speedup. Due to the space limit, all proofs are deferred to the supplemental material[†]. In the subsequence analysis, we will use the following definitions.

Definition 1 *We denote the total number of iterations and the total time used to converge when using N workers as T_N and $\mathcal{T}_N$, respectively. Then the iteration speedup (IS) and the time speedup (TS) are respectively defined as*

$$\mathrm{IS}^N = \frac{T_1}{T_N}, \tag{2}$$

$$\mathrm{TS}^N = \frac{\mathcal{T}_1}{\mathcal{T}_N}. \tag{3}$$

4.1 Main Results

Before establishing our main results, we introduce the following assumptions, all of which are commonly used in the analysis of distributed algorithms [Lian *et al.*, 2015; Aji and Heafield, 2017; Yu *et al.*, 2018].

Assumption 1

(1) ***Lipschitz gradient****: All local functions f_i's have L-Lipschitz gradients*

$$\|\nabla f_i(x) - \nabla f_i(y)\| \leq L\|x - y\|, \forall i, \forall x, y \in \mathbb{R}^d. \tag{4}$$

(2) ***Unbiased estimation****:*

$$\mathbb{E}_{\xi \sim \mathcal{D}_i} \nabla f_i(x, \xi) = \nabla f_i(x), \forall i, \tag{5}$$

$$\mathbb{E}_{i \sim \mathcal{P}} \mathbb{E}_{\xi \sim \mathcal{D}_i} \nabla f_i(x, \xi) = \nabla f(x). \tag{6}$$

[†] It can be downloaded from the anonymous link: https://github.com/IJCAI19-CoCoD-SGD/Supplemental-Material

(3) ***Bounded variance****: There exist constants σ and ζ such that*

$$\mathbb{E}_{\xi\sim\mathcal{D}_i}\|\nabla f_i(x,\xi)-\nabla f_i(x)\|^2\le\sigma^2,\ \ \forall x\in\mathbb{R}^d,\forall i, \quad (7)$$

$$\mathbb{E}_{i\sim\mathcal{P}}\|\nabla f_i(x)-\nabla f(x)\|^2\le\zeta^2,\ \ \forall x\in\mathbb{R}^d. \quad (8)$$

(4) ***Dependence of random variables****: $\xi_t^{i,j}$'s are independent random variables, where $t\in\{0,1,\cdots,T-1\}$, $i\in\{1,2,\cdots,N\}$, and $j\in\{1,2,\cdots,M_i\}$.*

To evaluate the convergence rate, the metric in nonconvex optimization is to bound the weighted average of the ℓ_2 norm of all gradients [Ghadimi and Lan, 2013; Lian *et al.*, 2015; Yu *et al.*, 2018].

Theorem 1 *Under Assumption 1, if the learning rate satisfies $\gamma\le\frac{1}{L}$, we have the following convergence result for Algorithm 1:*

$$\frac{1}{T}\sum_{t=0}^{T-1}D_1\mathbb{E}\|\nabla f(\hat{x}_t)\|^2$$
$$\le\ \frac{2(f(\hat{x}_0)-f^*)}{T\gamma}+D_2\Big(\frac{N\sigma^2}{\sum_{i=1}^N M_i}+2k\zeta^2\Big)+\frac{\gamma L\sigma^2}{\sum_{i=1}^N M_i}, \quad (9)$$

where k is the communication period and

$$D_1=1-2kD_2,\quad D_2=\frac{8\gamma^2L^2k}{1-16\gamma^2k^2L^2}. \quad (10)$$

Choosing the learning rate γ appropriately, we have the following corollary.

Corollary 1 *Under Assumption 1, when the learning rate is set as $\gamma=\frac{1}{\sigma\sqrt{\frac{T}{\sum_{i=1}^N M_i}}}$ and the total number of iterations satisfies*

$$T\ \ge\ \max\Bigg\{\frac{L^2(\sum_{i=1}^N M_i)}{\sigma^2},\frac{48(\sum_{i=1}^N M_i)L^2k^2}{\sigma^2},$$
$$\frac{144(\sum_{i=1}^N M_i)^3}{\sigma^6}L^2k^2\left(\frac{N\sigma^2}{\sum_{i=1}^N M_i}+2k\zeta^2\right)^2\Bigg\}, \quad (11)$$

we have the following convergence result for Algorithm 1:

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\|\nabla f(\hat{x}_t)\|^2\le\frac{4\sigma(f(\hat{x}_0)-f^*+L)}{\sqrt{T\sum_{i=1}^N M_i}}. \quad (12)$$

Corollary 1 indicates that the convergence rate of the weighted average model is $O\left(1/\sqrt{\sum_{i=1}^N M_iT}\right)$, which is consistent with S-SGD [Ghadimi and Lan, 2013]. Next, we establish the linear iteration speedup in both homogeneous and heterogeneous environments, and show the communication complexity of CoCoD-SGD.

Remark 1 (linear iteration speedup in the homogeneous environment). *For CoCoD-SGD in a homogeneous environment, all workers use the same mini-batch size: $M_1=M_2=\cdots=M_N=M$. According to (12), CoCoD-SGD converges at the rate $O\left(1/\sqrt{NMT}\right)$. Consequently, to achieve the ϵ-approximation solution, $O(1/(NM\epsilon^2)$ iterations are needed, which means CoCoD-SGD has a linear iteration speedup with respect to the number of workers according to the definition of IS in (2).*

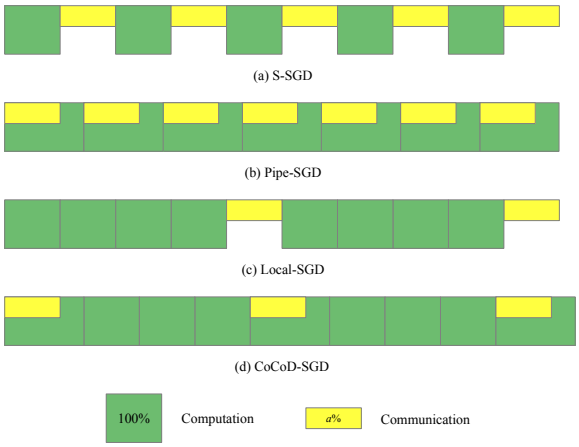


Figure 1: Comparison of S-SGD, Pipe-SGD, Local-SGD and CoCoD-SGD. A green block indicates computation which can make full use of the hardware resources, and a yellow block corresponds to communication which only uses $a\%$ of the resources. S-SGD and Local-SGD run computation and communication sequentially, while Pipe-SGD and CoCoD-SGD run them in parallel. Both Local-SGD and CoCoD-SGD communicate every k iterations.

Remark 2 (linear iteration speedup in the heterogeneous environment). *With Proportionally Sampling, we have $M_i/M_j=\mathcal{C}_i/\mathcal{C}_j$. According to (12), to achieve the ϵ-approximation solution, the number of iterations required is $O\left(1/(\sum_{i=1}^N M_i\cdot\epsilon^2)\right)$, that is $O\left(\mathcal{C}_1/\left(\sum_{i=1}^N\mathcal{C}_i\cdot M_1\epsilon^2\right)\right)$, which means CoCoD-SGD has a linear iteration speedup with respect to the total computation capability of all workers according to (2).*

Remark 3 (communication complexity). *From (11), we have $T\ge\max\{O(N),O(Nk^2),O(N^3k^4)\}$. Therefore, when the communication period is bounded by $O\left(T^{\frac{1}{4}}/N^{\frac{3}{4}}\right)$, the convergence rate in Corollary 1 is achievable. As a result, the total communication complexity of CoCoD-SGD is $O\left(T/\left(T^{\frac{1}{4}}/N^{\frac{3}{4}}\right)\right)$, that is $O\left(T^{\frac{3}{4}}N^{\frac{3}{4}}\right)$.*

Remark 4 (choice of learning rate). *When we run CoCoD-SGD for a fixed number of epochs, $\sum_{i=1}^N M_iT$ will be a constant. The learning rate suggested in Corollary 1 can be written as $\gamma=\frac{\sum_{i=1}^N M_i}{\sigma\sqrt{\sum_{i=1}^N M_iT}}$. Thus, for CoCoD-SGD with N workers, the learning rate should be set as $\gamma_N=N\cdot\gamma_1$ in the homogeneous environment and $\gamma_N=\sum_{i=1}^N\mathcal{C}_i/\mathcal{C}_1\cdot\gamma_1$ in the heterogeneous environment.*

4.2 Time Speedup Analysis

Next, we compare and analyse the time speedup of CoCoD-SGD and other distributed algorithms when they are applied in practice.

When multiple nodes with GPUs are used to train a deep neural network, GPUDirect communication, where all computation threads and communication threads are executed in GPUs, is widely adopted [Paszke *et al.*, 2017; Chen *et*

al., 2015; Sergeev and Balso, 2018] and verified to be efficient [Potluri *et al.*, 2013]. The computation threads can make full use of GPU resources especially when the size of mini-batch is large, while the communication threads can only use partial of the GPU resources, which is assumed to be $a\%$ for ease of analysis. We further denote the time of one computation as $\mathcal{T}_{\text{comp}}$ and the time of communication when using N workers as $\mathcal{T}_{\text{comm}}^N$.

A comparison of the running processes of CoCoD-SGD and other standard distributed algorithms is shown in Figure 1. All algorithms have provable linear iteration speedup. Thus, the time speedup equals $(N \cdot \mathcal{T}_1^{\text{av}})/\mathcal{T}_N^{\text{av}}$ according to (3), where $\mathcal{T}_N^{\text{av}}$ is denoted as the average time required to finish one iteration among N workers. For S-SGD, computation and communication are sequential. Therefore, its time speedup is

$$\text{TS}_{\text{S-SGD}}^N = \frac{N \cdot \mathcal{T}_{\text{comp}}}{\mathcal{T}_{\text{comp}} + \mathcal{T}_{\text{comm}}^N}. \tag{13}$$

On the other hand, Pipe-SGD decouples the dependence of computation and communication by using a stale gradient, but the communication complexity is still $O(T)$ as it needs to communicate all gradients. Accordingly, its time speedup is

$$\text{TS}_{\text{Pipe-SGD}}^N = \frac{N \cdot \mathcal{T}_{\text{comp}}}{\mathcal{T}_{\text{comp}} + \mathcal{T}_{\text{comm}}^N \cdot a\%}. \tag{14}$$

According to Corollary 1 in [Yu *et al.*, 2018] and Remark 3, Local-SGD and CoCoD-SGD have the same convergence rate as S-SGD when the communication period is $O(T^{\frac{1}{4}}/N^{\frac{3}{4}})$. The difference between Local-SGD and CoCoD-SGD is that CoCoD-SGD runs computation and communication in parallel while Local-SGD runs them sequentially. As a result, their time speedups are

$$\text{TS}_{\text{Local-SGD}}^N = \frac{N \cdot \mathcal{T}_{\text{comp}}}{\mathcal{T}_{\text{comp}} + \mathcal{T}_{\text{comm}}^N/k}, \tag{15}$$

and

$$\text{TS}_{\text{CoCoD-SGD}}^N = \frac{N \cdot \mathcal{T}_{\text{comp}}}{\mathcal{T}_{\text{comp}} + (\mathcal{T}_{\text{comm}}^N \cdot a\%)/k}, \tag{16}$$

respectively, where $k = O(T^{\frac{1}{4}}/N^{\frac{3}{4}})$. As (16) is bigger than (13), (14), and (15), we can verify that CoCoD-SGD achieves the best time speedup.

5 Experiments

In this section, we validate the performance of CoCoD-SGD in both homogeneous and heterogeneous environments.

5.1 Experimental Settings

Hardware. We evaluate CoCoD-SGD on a cluster where each node has 3 Nvidia Geforce GTX 1080Ti GPUs, 2 Xeon(R) E5-2620 cores and 64 GB memory. The cluster has 6 nodes, which are connected with a 56Gbps InfiniBand network. Each GPU is viewed as one worker in our experiments.

	CIFAR10		CIFAR100	
	ResNet18	VGG16	ResNet18	VGG16
S-SGD	94.00%	93.25%	75.08%	70.81%
Pipe-SGD	93.94%	93.09%	75.13%	70.59%
Local-SGD	94.35%	93.30%	75.42%	71.13%
CoCoD-SGD	94.41%	93.38%	75.67%	72.24%

Table 1: Final best test accuracy for all tasks in a homogeneous environment. 16 workers in total.

Software. We use Pytorch 0.4.1 [Paszke *et al.*, 2017] to implement the algorithms in our experiments, and use Horovod 0.15.2 [Sergeev and Balso, 2018], OpenMPI 3.1.2[‡], and NCCL 2.3.7[§] to conduct the GPUDirect communication with the Ring-AllReduce algorithm.

Datasets. We use two datasets for image classification.

- CIFAR10 [Krizhevsky and Hinton, 2009]: it consists of a training set of 50, 000 images from 10 classes, and a test set of 10, 000 images.
- CIFAR100 [Krizhevsky and Hinton, 2009]: it is similar to CIFAR10 but has 100 classes.

Tasks. We train ResNet18 [He *et al.*, 2016] and VGG16 [Simonyan and Zisserman, 2014] on the two datasets.

Baselines. We compare CoCoD-SGD with S-SGD, Pipe-SGD [Li *et al.*, 2018] and Local-SGD [Stich, 2018]. All of them support Ring-AllReduce communication.

Hyper-parameters. We use the following hyper-parameters.

- Basic batch size: 32 for both ResNet18 and VGG16.
- Basic learning rate: For both networks we start the learning rate from 0.01 and decay it by a factor of 10 at the beginning of the 81st epoch.
- Momentum: 0.9.
- Weight decay: 10^{-4}.
- Communication period and gradient staleness: Since the variance of stochastic gradients is higher in the beginning, we set the communication period to 1 for the first 10 epochs and 5 for the subsequential epochs. The staleness of gradients in Pipe-SGD is set to 1 as suggested in [Li *et al.*, 2018].

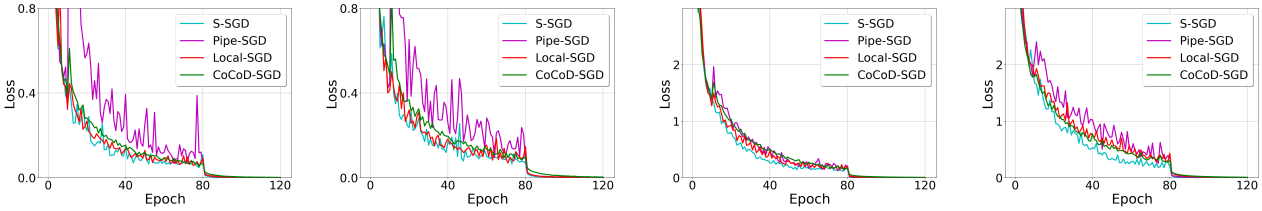
5.2 Homogeneous Environment

In a homogeneous environment, all workers have the same computation speed. So for N workers, we set $\gamma_N = N\gamma$ as suggested in Remark 4. And the learning rate warm-up scheme proposed in [Goyal *et al.*, 2017] is adopted.

Comparison of convergence rate. Figure 2 shows the training loss with regard to epochs of ResNet18 and VGG16 on 16 GPUs. All algorithms have similar convergence speed, which validates the theoretical results claimed in Section 4.1.

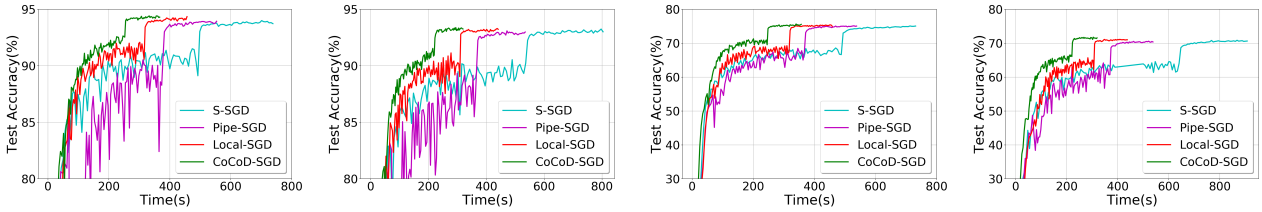
[‡]https://openmp.org

[§]N Luehr. Fast multi-gpu collectives with nccl, 2016.



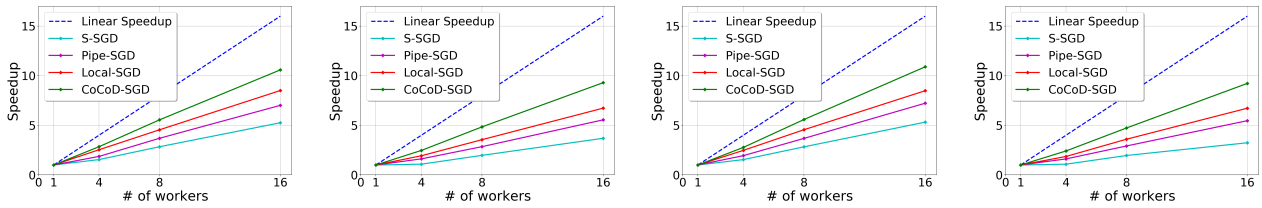
(a) CIFAR10-ResNet-Epoch-Loss (b) CIFAR10-VGG-Epoch-Loss (c) CIFAR100-ResNet-Epoch-Loss (d) CIFAR100-VGG-Epoch-Loss

Figure 2: Training loss for ResNet18 and VGG16 on CIFAR10 and CIFAR100 w.r.t epochs in a homogeneous environment. All algorithms have a similar convergence rate.



(a) CIFAR10-ResNet-Time-Accuracy (b) CIFAR10-VGG-Time-Accuracy (c) CIFAR100-ResNet-Time-Accuracy (d) CIFAR100-VGG-Time-Accuracy

Figure 3: Test accuracy for ResNet18 and VGG16 on CIFAR10 and CIFAR100 w.r.t time in a homogeneous environment. CoCoD-SGD achieves the fastest convergence and the results are consistent with the analysis in Section 4.2.



(a) CIFAR10-ResNet-Speedup (b) CIFAR10-VGG-Speedup (c) CIFAR100-ResNet-Speedup (d) CIFAR100-VGG-Speedup

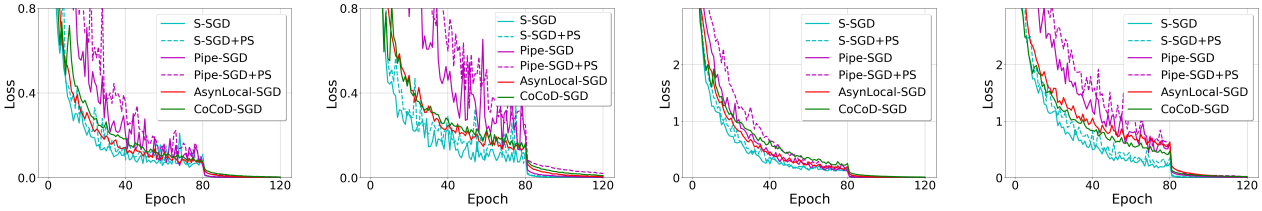
Figure 4: The comparison of time speedup for ResNet18 and VGG16 on CIFAR10 and CIFAR100 in a homogeneous environment. CoCoD-SGD achieves the best time speedup.

Comparison of convergence speed. Figure 3 shows the test accuracy regarding time on 16 GPUs and Table 1 shows the best test accuracies of all algorithms on the two datasets. We evaluate the accuracy on the test set during training, but we only accumulate the time used for training. As shown in Figure 3, CoCoD-SGD achieves almost $2\times$ and $2.5\times$ speedup against S-SGD for ResNet18 and VGG16 respectively. Although Pipe-SGD can run computation and communication in parallel, it is still slower than Local-SGD and CoCoD-SGD since its communication complexity is higher. CoCoD-SGD converges faster than others since it not only runs computation and communication simultaneously but also has a lower communication complexity. The results verify our theoretical results claimed in Section 4.2. In the meanwhile, we can observe from Table 1 that CoCoD-SGD does not sacrifice the test accuracy on both datasets and may get better results than S-SGD.

Comparison of time speedup. Figure 4 shows the time speedup for ResNet18 and VGG16 when the number of workers increases from 1 to 16. We run the experiments for 120 epochs on 1 GPU and multiple GPUs. The speedup for ResNet18 is better due to its smaller model size. On both tasks, CoCoD-SGD achieves the fastest convergence and the best time speedup, which validates our time speedup analysis.

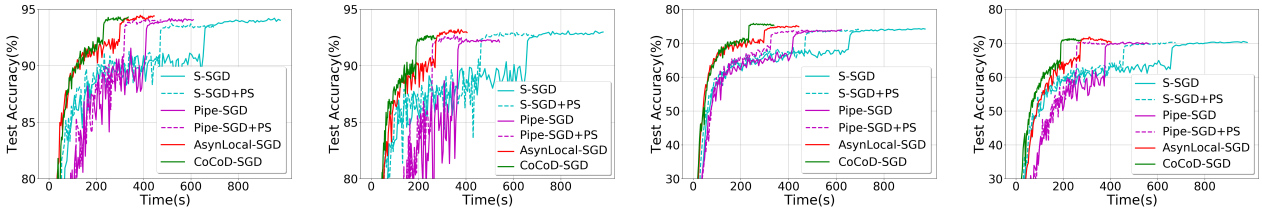
5.3 Heterogeneous Environment

To simulate a heterogeneous environment, we use 16 workers in our experiments and reduce the computation speed of 8 workers by half. For S-SGD and Pipe-SGD, *Proportionally Sampling* proposed in Section 3.2 can be also applied in the heterogeneous environment, which is equivalent to increasing the mini-batch size. And we denote corresponding algorithms as S-SGD+PS and Pipe-SGD+PS respectively. When one algorithm employs *Proportionally Sampling*, the batch size is set to 32 for the slower workers and 64 for the faster work-



(a) Heter-CIFAR10-ResNet-Epoch-Loss (b) Heter-CIFAR10-VGG-Epoch-Loss (c) Heter-CIFAR100-ResNet-Epoch-Loss (d) Heter-CIFAR100-VGG-Epoch-Loss

Figure 5: Training loss for ResNet18 and VGG16 on CIFAR10 and CIFAR100 w.r.t epochs in a heterogeneous environment. All algorithms have a similar convergence rate.



(a) Heter-CIFAR10-ResNet-Time-Accuracy (b) Heter-CIFAR10-VGG-Time-Accuracy (c) Heter-CIFAR100-ResNet-Time-Accuracy (d) Heter-CIFAR100-VGG-Time-Accuracy

Figure 6: Test accuracy for ResNet18 and VGG16 on CIFAR10 and CIFAR100 w.r.t time in a heterogeneous environment. CoCoD-SGD converges fastest and does not sacrifice the test accuracy.

ers. In the heterogeneous environment with 16 workers, we set $\gamma_{16}^{\text{ps}} = 0.01*(8+8*2)$ for algorithms with *Proportionally Sampling* and $\gamma_{16} = 0.01 * 16$ for others as suggested in Remark 4. In addition, the asynchronous version of Local-SGD (AsynLocal-SGD), which is proposed for heterogeneous environments in [Yu *et al.*, 2018], is included in comparison with CoCoD-SGD in our experiments.

Comparison of convergence rate. Figure 5 presents the training loss regarding epochs of ResNet18 and VGG16 on 16 GPUs, which exhibits similar convergence rate for all the algorithms in comparison. Besides, we can see from Table 2 that CoCoD-SGD does not lose the test accuracy.

Comparison of convergence speed. Figure 6 shows the curves of the test accuracy regarding time. As we can see, CoCoD-SGD converges fastest and achieves almost 2.5× and 3× speedup against S-SGD for ResNet18 and VGG16 respectively. Comparing with the results in Figure 3, S-SGD converges slower due to the heterogeneous computation, while CoCoD-SGD is robust. When equipped with *Proportionally Sampling*, S-SGD and Pipe-SGD converge faster since the hardware utilization is improved.

	CIFAR10		CIFAR100	
	ResNet18	VGG16	ResNet18	VGG16
S-SGD	94.18%	93.09%	74.30%	70.51%
S-SGD+PS	93.73%	92.96%	74.16%	70.32%
Pipe-SGD	94.17%	92.36%	73.95%	70.26%
Pipe-SGD+PS	94.10%	92.71%	73.84%	70.15%
AsynLocal-SGD	94.47%	93.24%	75.21%	71.80%
CoCoD-SGD	94.33%	92.69%	75.74%	71.33%

Table 2: Final best test accuracy for all tasks in a heterogeneous environment. 16 workers in total.

6 Conclusion

In this paper, we propose a computation and communication decoupled stochastic gradient descent (CoCoD-SGD) for distributed optimization. In comparison with existing distributed algorithms, CoCoD-SGD can run computation and communication simultaneously to make full use of hardware resources and has lower communication complexity. CoCoD-SGD is also theoretically justified to have the same convergence rate as S-SGD and obtains linear iteration speedup in both homogeneous and heterogeneous environments. In addition, CoCoD-SGD achieves faster distributed training with superior time speedup when comparing with others. Experimental results demonstrate the efficiency of the proposed algorithm.

Acknowledgments

This research was supported by the National Natural Science Foundation of China (No. 61673364, No. 91746301) and the Fundamental Research Funds for the Central Universities (WK2150110008). We also gratefully acknowledge the support of Cheng Li and Youhui Bai from USTC for providing the experimental environment.

References

[Abadi *et al.*, 2016] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard,

et al. Tensorflow: a system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

[Agarwal and Duchi, 2011] Alekh Agarwal and John C Duchi. Distributed delayed stochastic optimization. In *Advances in Neural Information Processing Systems*, pages 873–881, 2011.

[Aji and Heafield, 2017] Alham Fikri Aji and Kenneth Heafield. Sparse communication for distributed gradient descent. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 440–445, 2017.

[Alistarh *et al.*, 2017] Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. Qsgd: Communication-efficient sgd via gradient quantization and encoding. In *Advances in Neural Information Processing Systems*, pages 1709–1720, 2017.

[Alistarh *et al.*, 2018] Dan Alistarh, Torsten Hoefler, Mikael Johansson, Nikola Konstantinov, Sarit Khirirat, and Cédric Renggli. The convergence of sparsified gradient methods. In *Advances in Neural Information Processing Systems*, pages 5977–5987, 2018.

[Chen *et al.*, 2015] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274*, 2015.

[Chen *et al.*, 2016] Jianmin Chen, Xinghao Pan, Rajat Monga, Samy Bengio, and Rafal Jozefowicz. Revisiting distributed synchronous sgd. *arXiv preprint arXiv:1604.00981*, 2016.

[Dekel *et al.*, 2012] Ofer Dekel, Ran Gilad-Bachrach, Ohad Shamir, and Lin Xiao. Optimal distributed online prediction using mini-batches. *Journal of Machine Learning Research*, 13(Jan):165–202, 2012.

[Ghadimi and Lan, 2013] Saeed Ghadimi and Guanghui Lan. Stochastic first- and zeroth-order methods for nonconvex stochastic programming. *SIAM Journal on Optimization*, 23(4):2341–2368, 2013.

[Goyal *et al.*, 2017] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

[He *et al.*, 2016] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[Krizhevsky and Hinton, 2009] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[Li *et al.*, 2014] Mu Li, David G Andersen, Alexander J Smola, and Kai Yu. Communication efficient distributed machine learning with the parameter server. In *Advances in Neural Information Processing Systems*, pages 19–27, 2014.

[Li *et al.*, 2018] Youjie Li, Mingchao Yu, Songze Li, Salman Avestimehr, Nam Sung Kim, and Alexander Schwing. Pipe-sgd: A decentralized pipelined sgd framework for distributed deep net training. In *Advances in Neural Information Processing Systems*, pages 8056–8067, 2018.

[Lian *et al.*, 2015] Xiangru Lian, Yijun Huang, Yuncheng Li, and Ji Liu. Asynchronous parallel stochastic gradient for nonconvex optimization. In *Advances in Neural Information Processing Systems*, pages 2737–2745, 2015.

[Lin *et al.*, 2017] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William J Dally. Deep gradient compression: Reducing the communication bandwidth for distributed training. *arXiv preprint arXiv:1712.01887*, 2017.

[Paszke *et al.*, 2017] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[Potluri *et al.*, 2013] Sreeram Potluri, Khaled Hamidouche, Akshay Venkatesh, Devendar Bureddy, and Dhabaleswar K Panda. Efficient inter-node mpi communication using gpudirect rdma for infiniband clusters with nvidia gpus. In *Parallel Processing (ICPP), 2013 42nd International Conference on*, pages 80–89. IEEE, 2013.

[Sergeev and Balso, 2018] Alexander Sergeev and Mike Del Balso. Horovod: fast and easy distributed deep learning in TensorFlow. *arXiv preprint arXiv:1802.05799*, 2018.

[Simonyan and Zisserman, 2014] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[Stich *et al.*, 2018] Sebastian U Stich, Jean-Baptiste Cordonnier, and Martin Jaggi. Sparsified sgd with memory. In *Advances in Neural Information Processing Systems*, pages 4452–4463, 2018.

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

[Wen *et al.*, 2017] Wei Wen, Cong Xu, Feng Yan, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Terngrad: Ternary gradients to reduce communication in distributed deep learning. In *Advances in neural information processing systems*, pages 1509–1519, 2017.

[Yu *et al.*, 2018] Hao Yu, Sen Yang, and Shenghuo Zhu. Parallel restarted sgd for non-convex optimization with faster convergence and less communication. *arXiv preprint arXiv:1807.06629*, 2018.

[Zhou and Cong, 2018] Fan Zhou and Guojing Cong. On the convergence properties of a k-step averaging stochastic gradient descent algorithm for nonconvex optimization. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 3219–3227. International Joint Conferences on Artificial Intelligence Organization, 7 2018.