Communication-Efficient Distributed Deep Learning: A Comprehensive Survey

Zhenheng Tang*, Shaohuai Shi*, Xiaowen Chu*, Wei Wang†, Bo Li†
*Department of Computer Science, Hong Kong Baptist University
{zhtang, csshshi, chxw}@comp.hkbu.edu.hk
†Department of Computer Science and Engineering, The Hong Kong University of Science and Technology
{weiwa, bli}@cse.ust.hk

***Abstract*—Distributed deep learning becomes very common to reduce the overall training time by exploiting multiple computing devices (e.g., GPUs/TPUs) as the size of deep models and data sets increases. However, data communication between computing devices could be a potential bottleneck to limit the system scalability. How to address the communication problem in distributed deep learning is becoming a hot research topic recently. In this paper, we provide a comprehensive survey of the communication-efficient distributed training algorithms in both system-level and algorithmic-level optimizations. In the system-level, we demystify the system design and implementation to reduce the communication cost. In algorithmic-level, we compare different algorithms with theoretical convergence bounds and communication complexity. Specifically, we first propose the taxonomy of data-parallel distributed training algorithms, which contains four main dimensions: communication synchronization, system architectures, compression techniques, and parallelism of communication and computing. Then we discuss the studies in addressing the problems of the four dimensions to compare the communication cost. We further compare the convergence rates of different algorithms, which enable us to know how fast the algorithms can converge to the solution in terms of iterations. According to the system-level communication cost analysis and theoretical convergence speed comparison, we provide the readers to understand what algorithms are more efficient under specific distributed environments and extrapolate potential directions for further optimizations.**

I. Introduction

Deep learning (DL) has attained profound advances in recent years. Researchers and engineers have applied DL technologies to solve their problems in various fields including computer vision [1], natural language processing [2][3][4], speech recognition [5] and many others.

In DL, increasing the size of the training datasets generally offers to improve the model performance (e.g., accuracy in classification tasks) [6][7][8][7]. However, with the increased data size and model complexity, the training process of DL is very computation-intensive and thus time-consuming. For example, training a state-of-the-art ResNet-50 [1] model (in 90 epochs) on the ImageNet dataset [9] with a latest Nvidia Tesla V100 GPU requires about two days [10]. In general, one should tune the hyper-parameters for some tasks, which requires much more time to achieve acceptable performance. In order to reduce the training time, on the one hand, the computing power of a single accelerator should be fully utilized by using highly optimized software implementation [11][12][13][14][15]. On the other hand, distributed training [16][17][18][19] is becoming much popular to accelerate the training process using multiple processors including CPUs [20], GPUs [21][22][23] and Google TPUs [24]. Intuitively multiple processors collaborating in training one task can reduce the overall training time, but the communication cost between processors generally limits the system scalability [22]. Even worse, when deploying the high-speed computing devices (e.g., Nvidia V100 GPUs) with low-speed interconnect (e.g., 1/10 Gbps Ethernet) to train a deep model (i.e., the computation-to-communication ratio is low), multiple processors could achieve lower performance than that of a single processor [25]. Therefore, to fully utilize the computing power of distributed clusters, data communication should be well optimized in distributed training of deep models.

The mathematical formulation of DL can be defined as the following optimization problem:

$$\min_{\mathbf{x}\in\mathbb{R}^N} f_s(\mathbf{x}) := \mathbb{E}_{\xi_i\sim\mathcal{D}} F(\mathbf{x};\xi_i), \tag{1}$$

where the random variable ξ_i has a probability distribution $\mathcal{D}$, referring to a data sample from a dataset, $\mathbf{x}$ represents all the parameters of the model, N is the number of parameters, and $f_s : \mathbb{R}^N \to \mathbb{R}$ is the objective function.

A. Stochastic Gradient Descent

Gradient-based optimization algorithms are the most widely used in solving the above optimization problem in DL. Due to high computation complexity of second-order gradient descent methods, the first-order gradient descent methods, especially the stochastic gradient descent (SGD) with mini-batch[1] and it variants, are commonly used in DL. The update rule of SGD has the following form.

$$G_t(\mathbf{x}_t) = \nabla F_t(\mathbf{x}_t;\xi_t) \tag{2}$$

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \gamma G_t(\mathbf{x}_t), \tag{3}$$

where $\mathbf{x}_t \in \mathbb{R}^N$ is the N-dimensional model parameter at iteration t, ξ_t is a randomly sampled mini-batch of data, and γ is the learning rate (or step size). SGD is an iterative algorithm and has been proved that it can solve (1) under

[1]To simplify, throughout this paper, we use SGD to denote the training algorithm of SGD with mini-batch which includes the cases of one sample and more than one samples in each training iteration.

the assumptions that $f_s(\mathbf{x})$ is non-convex and is with L-Lipschitzian gradients [26]. The iterative process generally contains several steps: 1) It samples a mini-batch of **data** (i.e., ξ_t). 2) It performs the **feed-forward** computations to calculate the loss value (i.e., $F_t(\mathbf{x}_t;\xi_t)$) of the objective function. 3) It performs the **backward propagation** to calculate the gradients (i.e., $\nabla F_t(\mathbf{x}_t;\xi_t)$) with respect to the model parameters. 4) Finally, it **updates** the model parameters by Eq. (3). It is time-consuming to train a deep model especially for large models or datasets. It becomes common to exploit multiple processors to accelerate the training process using distributed training techniques.

In terms of the parallelism schemes of distributed training, there exist two main types: model parallelism and data parallelism.

B. Model Parallelism

Model parallelism splits model parameters to multiple computing workers [27]. Every computing worker holds different parameters or layers of the model (i.e., a subset of x). There are two main types to split the model in model parallelism, which are shown in Fig. 1(a) and Fig. 1(b) respectively. Due to the high dependency between different neurons in the deep model, each worker should exchange its output results with the other workers before continuing the computation of the next layer. One main advantage of model parallelism is that training huge size models becomes possible since each worker can only hold a subset of the model, and thus the memory requirement is low. However, the model parallel methods have several key issues, such as the unbalance parameter sizes and the high computing dependency in different layers of the deep model [27]. It is non-trivial and even NP-complete [28] to split the learning model into suitable parts and assign them appropriately to different computing nodes. Furthermore, one needs to design an elaborate algorithm to guarantee the robustness of model parallelism, which is difficult due to computing dependency. Because of these issues, it is not trivial to accelerate the training with model parallelism [29].

C. Data Parallelism

The other main type of distributed training is data parallelism, as shown in Fig. 2, in which the model parameters are replicated to all computing workers. In a single iteration, each computing worker processes different mini-batches of data to calculate the local gradient updates which are exchanged with the other workers before updating the model parameters. When using distributed multiple workers in data parallelism, the optimization problem of Eq. (1) is changed into a new form:

$$f^* := \min_{\mathbf{x}\in\mathbb{R}^d} \left[f(\mathbf{x}) := \frac{1}{n}\sum_{i=1}^{n} \underbrace{\mathbb{E}_{\xi_i\sim\mathcal{D}_i} F_i(\mathbf{x};\xi_i)}_{=:f_i(\mathbf{x})} \right] \tag{4}$$

where $f_i : \mathbb{R}^d \to \mathbb{R}$ represents the objective function of worker i, and n denotes the number of workers. This distributed optimization problem aims to minimize the mean of the objective functions at all workers.

D. Distributed SGD

The straightforward method to parallel SGD in multiple workers (say n workers) is the bulk synchronous parallel SGD (BSP-SGD) [30] and the parameter server (PS) architecture [31][32][27][33][34][35]. In this method, each worker (say worker i, and $i = 1, 2, ..., n$) pulls down the whole model from the PS (stores the global model), and then loads different data (i.e., $\xi_{i,t}$) and independently calculates the gradients ($\nabla F(\mathbf{x}_t;\xi_{i,t})$) at the same time. Before updating the model, the gradients are aggregated on the PS and all workers are synchronized by the barrier before beginning the next iteration as shown in Fig. 4. At last the PS average all received local gradients to update the global model. The update rule of BSP-SGD can be formulated as:

$$G_{i,t}(\mathbf{x}_t) = \nabla F_{i,t}(\mathbf{x}_t;\xi_{i,t}) \tag{5}$$

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \gamma\frac{1}{n}\sum_{i=1}^{n} G_{i,t}(\mathbf{x}_t), \tag{6}$$

where $G_{i,t}(\mathbf{x}_t)$ represents the gradient of $F_{i,t}(\mathbf{x}_t)$ of worker i at iteration t.

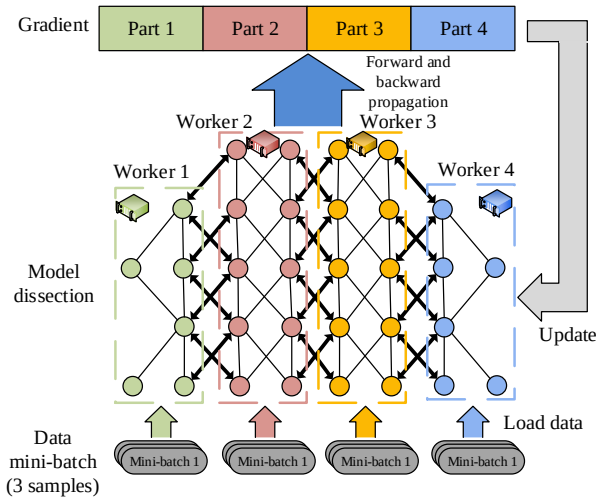
In the above scheme, there exist three key factors affecting the communication cost:

1) Communication synchronization and frequency of the n participating workers;
2) The aggregation algorithm of n vectors in distributed memory;
3) The communication traffic related to the dimension of the model (i.e., N).

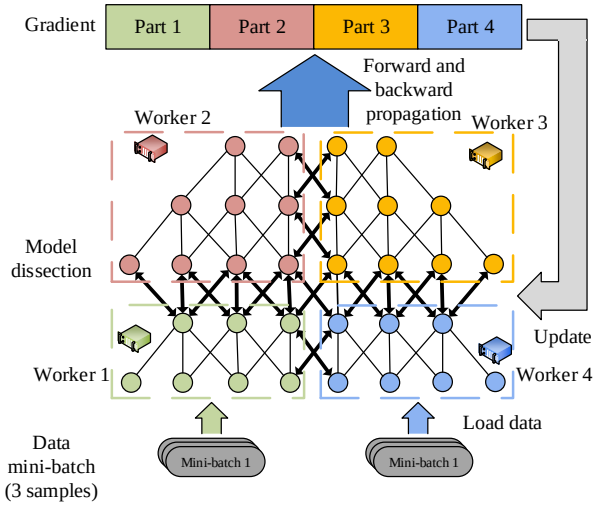
The communication-efficient algorithms mainly address the above three aspects to improve the scalability of distributed training.

1) Communication Synchronization and Frequency: From the view of when to communicate and synchronize, there are two main directions: 1) To relax the synchronization among all workers, asynchronous SGD (ASGD) [36] and stale synchronous parallel SGD (SSP-SGD) [37] are two main directions to make workers continue training without waiting for other workers at each iteration. As shown in Eq. (6), each worker should receive other $n-1$ workers' gradients to do average. 2) The frequency of communication can be reduced by more computation in one iteration [38][39][40][41][42][43][44][45][46].

2) The Aggregation Algorithm: From the view of whom to communicate with, except the PS architecture, there are two other communication schemes: 1) To relax the congestion problem in the PS architecture [31][32][27][33][34][35], the collective algorithms in MPI [47][48][49][21][23] are another kind of system design for aggregating distributed gradients/models; 2) Some researchers explore the gossip architecture [50][51][52][53][54], in which one worker can receive models from several peers or only one peer [51][55] and finally the algorithm converges to the same solution. Not



(a) Type 1.



(b) Type 2.

Fig. 1. Model parallelism. Each color represents one part of the model that is assigned to a worker. The thick lines represent the data communication between different workers.

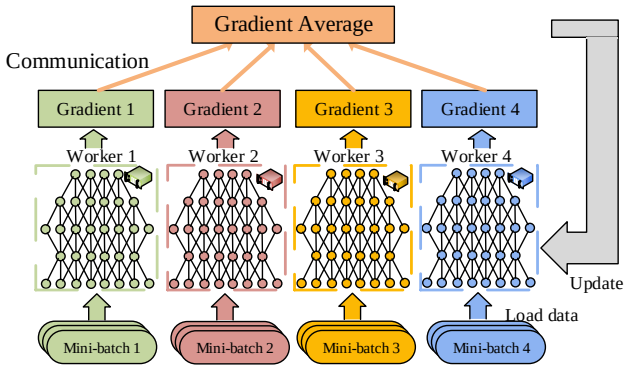


Fig. 2. Data parallelism.

only the congestion can be alleviated, but also the communication peers in one iteration can be reduced by the gossip architecture.

3) Communication Compression: From the view of what to communicate, the exchanged gradients or models can be compressed via quantization or sparsification before transmitting across the network connections to reduce the communication traffic with little impact on the model convergence [56][57][58][59][25][60][61][62].

4) Parallelism of Computations and Communications: From the view of how to make computation and communication running in parallel, many researchers develop different scheduling and pipelining algorithms. Due to the layer-wise structure of deep models, the scheduling algorithms for communication and computation tasks can reduce the waiting time between computation and communication tasks [63][47][64][65][66][67][68][69][70][71].

E. Organization

In this survey, we demystify the details of the above four aspects of communication-efficient distributed training algorithms with data parallelism. The rest of the paper is organized as follows. In Section II, we first illustrate the taxonomy of distributed SGD. We discuss the synchronous and asynchronous frameworks in Section III, followed by the system architectures that support gradient/model aggregation in Section IV. In Section V and Section VI, we introduce the techniques in reducing the communication traffic with gradient/model compression. The scheduling methods are introduced in Section VII. We summarize the theoretical convergence bounds in Section VIII, and some auxiliary tricks in training deep models with communication-efficient algorithms are present in Section IX. We summarize some related surveys in this area in Section X. We finally conclude the paper in Section XI.

II. Taxonomy of Distributed SGD

As discussed in Section I, the original BSP-SGD could easily suffer from the communication bottleneck due to the high communication synchronization frequency and the high model/gradient aggregation cost. Communication-efficient distributed training algorithms aim to address the communication problems while providing convergence guarantees, which can be summarized to four dimensions, including communication synchronization, system architectures, compression techniques, and parallelism of communication and computing, as shown in Table I.

First, methods in communication synchronization consider to relax the strict synchronization of BSP to reduce the impact of synchronization and the number of communications in the same period. Second, different system architectures aim to change the communication topology to avoid the

TABLE I
TAXONOMY OF DISTRIBUTED SGD

Dimension	Method
Communication Synchronization	Synchronous
	Stale-Synchronous
	Asynchronous
	Local SGD
System Architectures	Parameter Server
	All-Reduce
	Gossip
Compression Techniques	Quantization
	Coding
	Sparsification
Parallelism of Communication and Computing	Pipelining
	Scheduling

communication congestion of the PS and workers. Third, the compression techniques intend to compress the communication data to reduce communication traffic and thus reduce the communication time. Last, the parallelism of communication and computing to hide the communication time to achieve shorter iteration time. According to the taxonomy, an overview of the distributed deep learning is shown in Fig. 3. We discuss the each dimension in the following subsections.

A. Communication synchronization

Based on how many workers there are and how frequently each one works with others, data parallel methods can be split into four kinds of frameworks, synchronous, stale-synchronous, asynchronous, and local SGD. We will illustrate these methods in Section III in detail. The communication synchronization decides how frequently all local models are synchronized with others, which will influence not only the communication traffic but also the performance and convergence of model training. So there is a trade-off between the communication traffic and the convergence. Additionally, different synchronization schemes can be combined with different architectures. Here we introduce these four synchronization schemes under PS architecture due to its most extensive range of applications.

1) Synchronous: The synchronization in the data parallel distributed deep learning is that, before the next training, every worker must wait for all workers to finish the transmission of all parameters in the current iteration. Then they can start the next training. The classical algorithm of the synchronous framework is BSP as shown in Fig. 4, which generally requires the following steps: 1) Load the data; 2) Feed-forward computations; 3) Backward propagation computations; 4) Gradient aggregation with a barrier; and 5) Update the model wit aggregated gradients. The synchronization of all workers could make the stragglers in the cluster extremely influence the overall system throughput. The full data aggregation of all workers also introduce very high communication cost to limit the system scalability.

2) Stale-Synchronous: The stale-synchronous parallel (SSP) framework [33] aims to alleviate the straggler problem without losing synchronization. As shown in Fig. 4, it allows the faster workers to do more updates than the slower workers to reduce the waiting time of the faster workers. However, to keep the model consistency to guarantee the convergence, there is a staleness bounded barrier, which limits the iteration gap between the fastest worker and the slowest worker. For a maximum staleness bound s, the update formula of worker i at iteration $t+1$ is changed to

$$\mathbf{x}_{i,t+1} = \mathbf{x}_0 - \gamma \sum_{k=1}^{t} \sum_{j=1}^{n} G_{j,k}(\mathbf{x}_{j,k}) - \gamma \sum_{k=t-s}^{t} G_{i,k}(\mathbf{x}_{i,k}) - \gamma \sum_{(j,k)\in\mathcal{S}_{i,t+1}} G_{j,k}(\mathbf{x}_{j,k}), \quad (7)$$

where $\mathcal{S}_{i,t+1}$ is some subset of the updates from other workers during period $[t-s]$.

3) Asynchronous: Compared with the SSP framework, the asynchronous parallel (ASP) framework [72] completely eliminates the synchronization. As shown in Fig. 4, each work transmits its gradients to the PS after it calculates the gradients. Then the PS updates the global model without waiting for the other workers. Note that ASP is not friendly to MPI collectives. The update formula of asynchronous SGD can be summarized as

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \gamma \sum_{i=1}^{n} G_{i,t-\tau_{i,k}}(\mathbf{x}_{i,t-\tau_{k,i}}), \quad (8)$$

where the $\tau_{k,i}$ is the time delay between the moment when worker i calculates the gradient at the current iteration.

4) Local SGD: Another strict synchronization is Local SGD[73], which makes all workers run several or more iterations itself, and then averages all local models into the newest global model. There is a method named Model Average, which is very similar to Local SGD, locally running lots of iterations and does synchronization. The procedure of Local SGD can be formalized as

$$\mathbf{x}_{i,t+1} = \begin{cases} \mathbf{x}_{i,t} - \gamma G_{i,t}(\mathbf{x}_{i,t}), & \text{if } t+1 \notin \mathcal{I}_T \\ \mathbf{x}_{i,t} - \gamma \frac{1}{n} \sum_{i=1}^{n} G_{i,t}(\mathbf{x}_{i,t}), & \text{if } t+1 \in \mathcal{I}_T \end{cases} \quad (9)$$

where $\mathcal{I}_T$ represents the synchronization timestamps.

B. System Architectures

In terms of how to average the model parameters/gradients among distributed workers, we classify system architectures into three classes: Parameter Server (PS), All-Reduce, and Gossip.

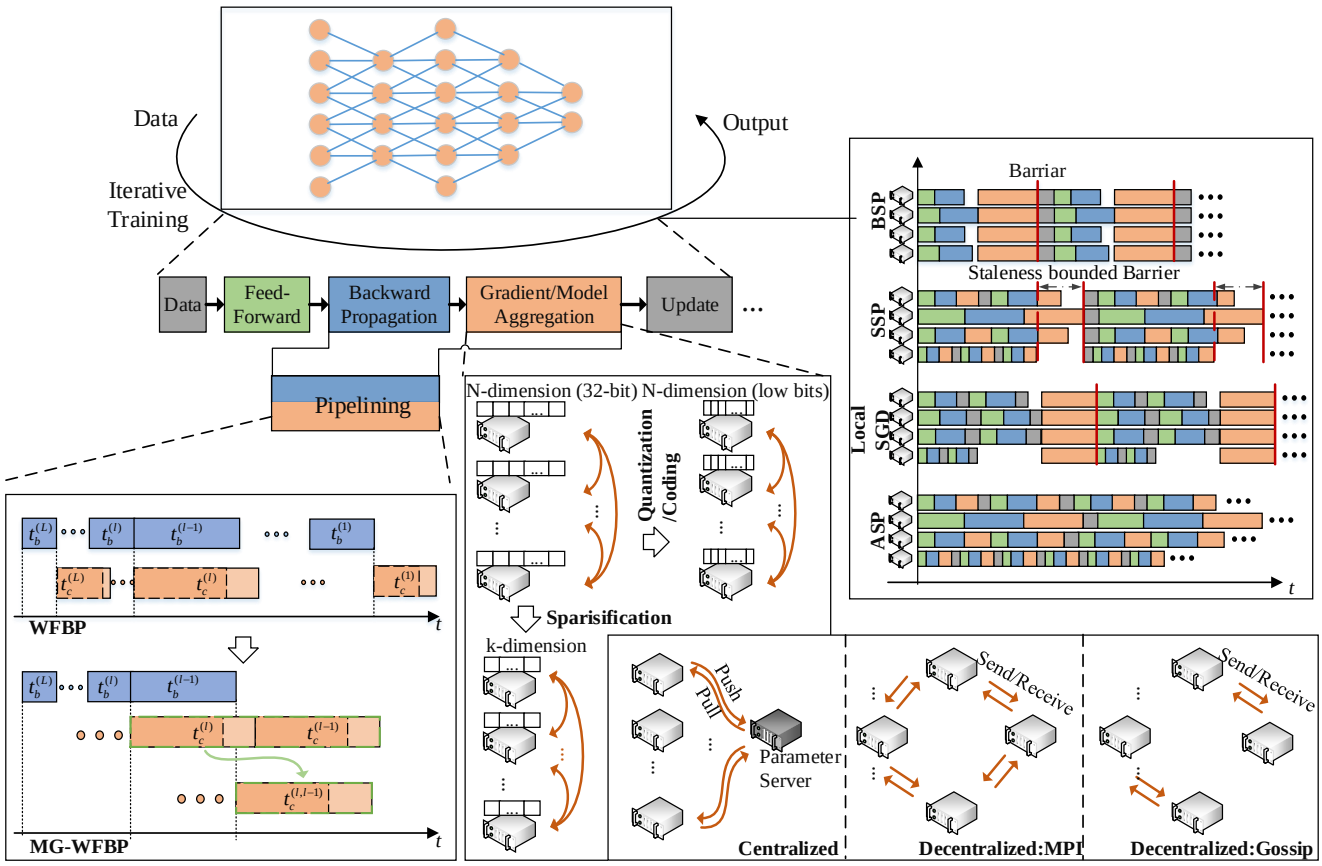


Fig. 3. Overview of distributed deep learning.

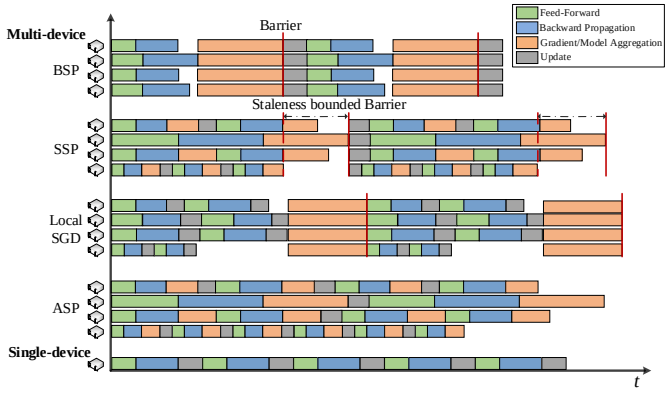


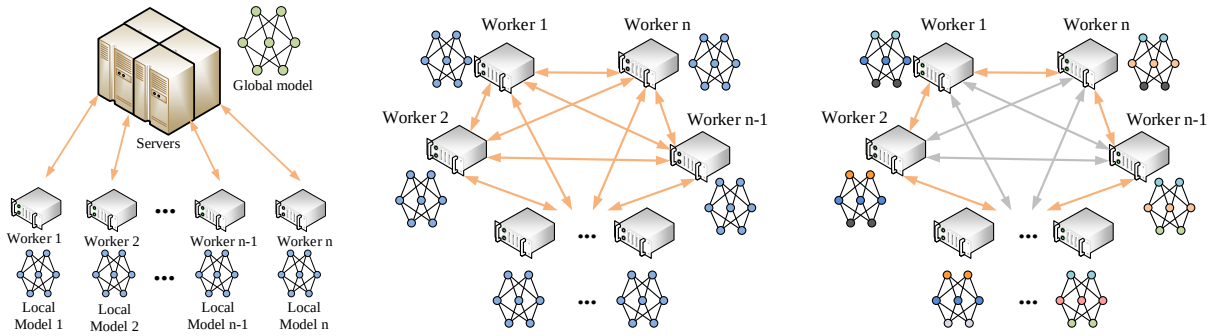
Fig. 4. Comparison of training with single device and multiple devices under different communication synchronization with the PS architecture.

1) Parameter Server: In the Parameter Server (PS) architecture as shown in Fig. 5(a), there are two roles, which are server and worker. Servers are responsible for saving the newest global model, and for collecting the updating information from the workers. Workers independently load the dataset, pull the newest global model from the server, and then compute the updates, which will be transmitted to the servers [74]. In this architecture, the servers are the center of the communication topology, so the PS architecture is generally referred as the centralized framework, which is widely used in distributed training [31][32][27][33][34][35]. The main challenge of the PS architecture is the communication congestion in the server [45][75], due to all workers communicating to the server.

2) All-Reduce: To avoid the communication bottleneck in the PS, researchers prefer to use the All-Reduce architecture [47][48][49][21][23] to implement the gradient aggregation without central servers.

As shown in Fig. 5(b), there is only one role, worker, in this method. All workers communicate without a central node. After the communication, they acquire all gradients of all workers and then update their local model, which is consistent with others. So, All-Reduce is a decentralized communication topology and a model-centralized topology since there is a consistent global model attained by synchronization. This makes there is no change in the updating Eq. (6) using BSP-SGD. Note that this architecture is not suitable for asynchronous communication because the collective communication fashion of All-Reduce. And it is difficult to apply All-Reduce in the asynchronous part in SSP, but it is easy to apply it in the synchronous part in SSP.

3) Gossip: The Gossip architecture is a kind of between-neighbors communication used to solve the distributed average problem [76][77][78][79]. And many re-



(a) PS architecture. (b) All-Reduce architecture. (c) Gossip architecture.

Fig. 5. Three different system architectures for model/gradient aggregation.

searchers exploit it to improve data parallel distributed SGD [80][81][82][83][84][85][50][51][52][53][54].

As shown in Fig. 5(c), the gossip architecture here not only has no parameter servers but also has no global model (represented by different colors of local models). Every worker communicates updates with their neighbors (also named peers). Workers exchange messages via those edges chosen in the current iteration (the blues ones in Fig. 5(c)), which can be described in a communication graph. Communication with only neighbors means that workers need not communicate with everyone else. After one communication during training, the algorithm does not guarantee the consistency of parameters across all workers but does guarantee it at the end of the algorithm. In other words, all local models are different during every iteration. Note that in asynchronous and SSP with the PS architecture, the local models are different, but the PS architecture keeps a global model, while the gossip architecture does not.

C. Compression Techniques

The compression methods consider compressing the gradients or model parameters that need to be transmitted to the workers or servers, to reduce the communication data. Most of the methods are lossy compression, which means that the original gradients or model parameters can not be fully recovered by the receiver. With less information gathered, the convergence could be sacrificed. It has been an active research direction to reduce the communication traffic with little impact on the convergence. Two main compression techniques, quantization and sparsification, are shown in Fig. 6.

1) Quantization: The quantization method is a kind of compression scheme that uses lower bits to represent the data which is originally represented by 32 bits on every dimension of the transmitted gradient. As shown in Fig. 6, the element in the quantized gradient is coded by fewer bits. Therefore, after the quantized communication, the gradients used to optimize the models are of low precision. Quantization of transmitted gradients is closely relevant to the deep learning with low precision. Deep learning with low precision is born in an environment where CPU and GPU need higher-speed calculating and lower memory for training DNNs. If the low precision of gradients can guarantee the convergence of training, the quantized communication is practicable. There have been already many researchers approaching the convergence of deep learning under the low-precision gradients [86][87][88][89][90].

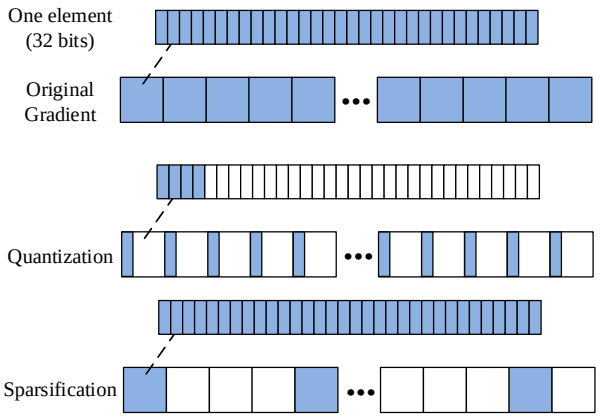


Fig. 6. Comparison of Quantization and Sparsification.

2) Sparsification: Sparsification methods aim to reduce the number of elements that are transmitted at each iteration. The key idea of sparsification is that in the update of SGD, only "significant" gradients are required to update the model parameter to guarantee the convergence of the training [25]. As shown in Fig. 6, a large proportion of the coordinates of the gradient vector are zeroed-out such that the zero-valued elements are no need to transmit. Gradient sparsification is a more aggressive compression technique to reduce the communication traffic than quantization.

D. Parallelism of Communication and Computing

The layer-wise structure of deep models makes it possible to parallels the communication tasks and computing tasks. The parallelism of communication and computing mainly aims to optimize the order of computation and communication such that the communication cost can be minimized [63][64][65][66][67][68][69][70]. For example, the pipelining

technique of wait-free backward propagation (WFBP) [63][47] and merged-gradient WFBP (MG-WFBP) [64] is widely used in BSP-SGD. It is also possible to combine the pipelining with gradient compression so that one can schedule the parallelism between compressed gradients' communications and computations on compression and backward propagation [91][92][93][94].

In summary, the benefit of efficient communication algorithms could not be free. The communication-efficient algorithms generally exchange less information than the naive method while they may sacrifice the convergence performance (e.g., model accuracy). To better demonstrate the different influences by these methods, we summarize the comparison in Table II.

III. Synchronous/asynchronous framework

In this section, we concretely introduce four different communication schemes.

A. Synchronous

No matter what the architecture the algorithm belongs to, the synchronization in the data-parallel distributed deep learning works in the following way: before the next iteration of training, every worker should wait for all workers to finish the transmission of parameters in the current iteration.

To be specific, in the PS architecture, the synchronous barrier exits until all workers finish the transmission of the parameters to parameter servers [95][96][97][98]. Similarly, the synchronization in the All-Reduce architecture is that all workers should wait for the completion of the all-reduce operation, which means that all workers have been updated to the same newest global model [99][100][25]. For the decentralized architecture, synchronization means that all workers will wait for the ending of the communication, but different from All-Reduce, workers in decentralized architecture do not necessarily keep the same model [51][101].

B. Stale-synchronous

Chen et al. [102] revisited the synchronous learning and proposed a method for mitigating the straggler problem, in which a mini-batch gradient is computed with only a subset of worker machines called Backup Workers [103]. The backup workers are the n_e extra workers. The main idea is that the parameter server stops waiting and updates the parameters when the gradients from any n workers arrive. Therefore, the gradients in the slowest n_e workers will be dropped when they arrive.

Ho et al. [33] proposed a stale synchronous parallel model, which relaxes the limitation of the strict synchronization in one iteration. Some faster workers are allowed to compute and update more iterations in advance. To guarantee the convergence, the algorithm sets a threshold of the maximal advance iteration. When the distance between the fastest and the slowest workers reaches the threshold, the fastest worker should wait for the slowest worker.

Another problem in the synchronous framework with Parameter Server is the congestion, especially when the model is large. Chen et al. [104] carefully analyzed this congestion problem when training deep learning models, and proposed a new method named as Round-Robin Synchronization Parallel to address this issue. The algorithm evenly staggers worker updates throughout the training process and coordinates the workers to update the gradients in a fixed round-robin order.

C. Asynchronous

In the asynchronous framework, the PS can update the global model with the updates from several or even one worker instead of all workers [105][106]. In contrast to synchronous algorithms, the asynchronous framework allows more independent updates of the nodes, which cut down the data transmitted during the communication between the current workers and the master. Also, the asynchronous design makes large-scale systems more robust to the failures of workers.

In [27], DistBelief is such an asynchronous framework that can utilize computing clusters with thousands of machines to train large models. Except that, they propose *Downpour SGD*, an asynchronous stochastic gradient descent algorithm for optimization. Multiple workers process data in parallel to calculate their own updates and communicating with the parameters server. For asynchronous parameters server, the steps in each iteration are similar to the asynchronous framework. The difference is that the global model is preserved in the parameters server, but there are some asynchronous algorithms using a decentralized framework.

Mote et al. [107] proposed a communication-efficient distributed algorithm named Distributed Alternating Direction Method of Multipliers (D-ADMM), which is extended from the multi-block ADMM algorithm. The D-ADMM can run on any connected network topology. Wei et al. [108][109] developed a decentralized ADMM algorithm.

Zhang et al. [110] pointed out that some problems exist in decentralized ADMM algorithms. The decentralized ADMM algorithms in [108][109] require the maintenance of a global clock. Because of the decentralized property, each group of workers needs to be aware of each others process. Moreover, decentralized ADMM algorithms heavily depend on the network topology. There should be a central node to keep the global model in decentralized ADMM algorithms. And the central node still needs to wait for all worker nodes to finish their tasks, which is similar to the synchronous framework. Considering these reasons, Zhang et al. [110] proposed an asynchronous distributed ADMM, in which the star topology and the parameter server architecture are used. They used two conditions to control the asynchrony: partial barrier and bounded delay.

In [34], Li et al. proposed an efficient algorithm named Delayed Block Proximal Gradient Method. In this algorithm, only a block of parameters is asynchronously updated per iteration. So only a part of parameters is needed to be transmitted between master and workers without waiting.

TABLE II
INFLUENCES OF DIFFERENT COMBINATIONS OF ARCHITECTURES AND SYNCHRONIZATION SCHEMES

Architecture	Synchronization	Model consistency	Communication Frequency	Communication Congestion	Convergence
PS	BSP	high	high	high	easy
	SSP	normal	high	normal	normal
	ASP	low	high	low	difficult
	Local SGD	normal	low	high	difficult
All-Reduce	BSP	high	high	low	easy
	SSP	-	-	-	-
	ASP	-	-	-	-
	Local SGD	normal	low	low	difficult
Gossip	BSP	low	high	low	normal
	SSP	-	-	-	-
	ASP	low	high	low	difficult
	Local SGD	low	low	low	difficult

Notes: Here the different levels only represent a relative description comparing with the other methods. The model consistency measures how local models are different from others, the communication frequency measures how frequently workers communicate with others, and communication congestion measures how heavy the traffic of the central node is.

Grishchenko et al. [106] proposed an asynchronous distributed algorithm that features a sparsification of upward communications (workers-to-master). They implemented sparsification by a uniformly sampling selection of local update entries, which makes communication more efficient. We will thoroughly discuss the sparsification technology in Section VI.

Without the need to wait for the slow workers, the asynchronous training has been proven to be faster than synchronous ones. However, asynchronous algorithms often result in lower convergence performance. Hence synchronous SGD is still the state-of-the-art in the data center setting [102].

D. Local SGD

In order to handle the problem of the huge communication cost, an extreme method named One-shot Averaging [19] is proposed, which runs m independent workers in parallel for a certain of iterations and then simply averages all the model parameters at the end. We consider the vanilla SGD, which exchanges the parameters at every iteration as another extreme. A balanced method named local SGD can choose to communicate in several iterations [38][39][40][41][42][43][44][45][46].

For local SGD, it is obvious that the less frequency of communications will cause more information to lose. Aggressive reduction of the communication frequency may lead to degraded convergence performance, so one may need more training iterations to achieve the same model accuracy, which could finally slow down the training. So it is crucial to find out the trade-off of the communication period.

Yu et al. [111] combined the distributed momentum SGD and the PR-SGD [46] to improve the performance of local SGD further. Furthermore, they also proved that this new algorithm could provide linear speedup to the training. Jiang et al. [112] exploited the quantization method together with local SGD, which leads to less communication complexity.

Another way to cut down the number of iterations to reduce communication data is by increasing the batch sizes. Yu et al. [113] proposed a Catalyst-like [114][115] algorithm, which dynamically increases the batch sizes after each training iteration, at the same time guaranteeing the same convergence rate of SSP.

Large-batch SGD could make the performance of generalization decrease [116][117][118][119][120]. Lin et al. [121] proposed *post-local SGD*, an efficient training algorithm to address this generalization issue of large-batch training, which allows one to scale the training onto much more parallel computing devices. This algorithm divides the whole training process into two phases, using mini-batch SGD in the first phase and local SGD in the second phase.

那说明其实可以采用momentum来加速

IV. CENTRALIZED/DECENTRALIZED FRAMEWORK

A. Parameter Servers

Early Parameter Server distributed machine learning methods [32][122][123][124][33][31] mainly focus on how to implement distributed machine learning at the system level so as to support a huge mount of model parameters. In the pioneering framework DistBelief [27], Dean et al. firstly successfully train the model with billion of parameters on an extremely large-scale cluster.

For the data parallel distributed machine learning algorithms, a lot of early works focus on CPU clusters [125][126][127][128][129][130]. And later, Cui et al. proposed GeePS [131], a parameter server architecture to scale deep learning models training across a large number of GPUs, which addresses the challenge of limited GPU memory when training large deep models.

Before long, many communication efficient methods were proposed. In [34], Li et al. began to consider communication efficient problems. They mitigated the data communication

volumes by using controllable synchronization and user-definable filters. Later, some methods specialized in efficient communication [89][132][133][134][135][136][137][138][139][140][141][142][25][143] arose, which will be illustrated concretely in Section V and VI.

Wang et al. [144] proposed a new measurement to measure whether the update of one worker is relevant or not. They measure the relevance by comparing the local update with the global update at the last iteration. If the relevance exceeds the threshold, this update will be seen as relevant and transmitted to the server.

B. All-Reduce

All-Reduce collective is an efficient alternative to aggregate the gradients among distributed workers without using the central servers. All-Reduce is an operation that sums or averages the distributed data, which is corresponding to the operation of averaging in Eq. (6). There are many system optimizations for the All-Reduce collective in the long history of the high-performance computing community [145][146][147][148]. Some popular All-Reduce algorithms are summarized in Table III with different latency and bandwidth costs for a N-dimension vector and a n-node cluster. Note that the communication cost of sending/receiving a N-dimension vector is modeled as $\alpha + \beta N$ [149].

TABLE III
COMMUNICATION COST OF ALL-REDUCE ALGORITHMS

Algorithm	Latency	Bandwidth
Binary tree	$2\alpha \log n$	$2\beta(\log n)N$
Recursive doubling	$\alpha \log n$	$\beta(\log n)N$
Ring	$2(n-1)\alpha$	$\frac{2(n-1)}{n}\beta N$

From Table III, the ring-based All-Reduce is bandwidth optimal so that it is widely used in distributed deep learning (e.g., Gloo[2] and earlier versions of NCCL[3].). However, the latency of the ring-based All-Reduce is linear to the number of workers, which could result in high communication costs when scaling to large-scale clusters. Recent updates of NCCL (started from version 2.4)[4] have integrated the double binary trees [148] to perform an all-reduction to achieve full bandwidth and a logarithmic latency. The double binary trees require the whole message to be broken down into multiple blocks and the workers to be formed as a binary tree so that different blocks can be executed in parallel. Therefore, for some small messages or small-scale clusters, recursive doubling or ring-based algorithms would be better.

To further reduce the latency term in All-Reduce while preserving the bandwidth optimality, the hierarchical All-Reduce algorithms [21][23][150] were also proposed, which can reduce the latency cost several times (related to the number of hierarchies). 2D-Torus All-Reduce [151][152] can also massively reduce the communication latency using a 2D-Torus topology network.

Under different topology networks (e.g., BCube [153]), it is also important to carefully design the All-Reduce algorithms to achieve lower latency and higher bandwidth. Wang et al. [153] proposed BML for the BCube topology to achieve efficient communication. Some topology-aware algorithms (e.g., Blink [154] and Plink [155]) were designed to be adaptive to the distributed environments to highly utilize the network bandwidth with low latency.

C. Gossip

There are many decentralized algorithms focusing on gradient based methods [80][81][82][83][84][85][50][51][52][53][54]. For distributed SGD, Lian et al. [51] provided the first theoretical analysis indicating that decentralized algorithms are likely to surpass the centralized algorithms with much less communication cost.

Similar to the All-Reduce architecture, one of the main advantages of the gossip architecture is that there is no master node so that there is no communication problem in the central server [54]. Compared to the All-Reduce architecture, the gossip architecture is relatively fault-tolerant to the stragglers and worker failures.

The key issue in the gossip architecture is how to ensure the attainment of the identical model parameters across all workers. This problem is named as "consensus", having been studied for a long time [156][157][158][159][160][161]. Formally, consensus is a state of all workers attaining a same opinion in common. In the gossip distributed SGD, the consensus is the status that all workers have the identical model parameters.

The gossip-based approaches are constrained to use symmetric communication and thus inherently require deadlock-avoidance, and more synchronization, which makes them slower and more sensitive to stragglers. In [53], Assran et al. combined Stochastic Gradient Push (SGP) [162] with PUSHSUM [163][76]. In SGP, each worker can only send the gradient to its out-neighbors without waiting for the response of these neighbors.

Obtaining the "consensus" incurs the difficulty in designing a method achieving efficient communication, high convergence rate, and consensus at the same time. Considering all these issues, model compression can be combined with decentralized learning algorithms to reduce the communication cost [54][164][165][55].

V. QUANTIZATION METHODS

The communication of gradients in synchronous data-parallel SGD methods actually is a distributed mean estimation problem [166][167]. Suresh et al. [166] and Jakub et al. [167] analyzed the communication-efficient algorithms for distributed mean estimation. They used the Mean Squared Error (MSE) to measure how accurate the quantization methods are.

[2]https://github.com/facebookincubator/gloo
[3]https://developer.nvidia.com/nccl
[4]https://devblogs.nvidia.com/massively-scale-deep-learning-training-nccl-2-4/

And then, they proposed a coding strategy achieving the best MSE for a given communication cost. When one wants to decrease the MSE error by increasing the number of levels of quantization, the communication cost will increase. In order to reduce the communication cost, Suresh et al. [166] proposed two methods, Stochastic Rotated Quantization and Variable Length Coding. In Stochastic Rotated Quantization, all clients and the central server generate a global random rotation matrix and then try to find an orthogonal matrix $\mathbb{R}$ that achieves low MSE. Variable Length Coding uses arithmetic of Huffman Coding corresponding to the number of times each quantized value has appeared.

Sei et al. [132] proposed 1-bit SGD to reduce the transmitted data volumes, and they successfully trained the deep model on traditional speech applications with 10x speedup. The 1-bit SGD reduces each element of the gradient to one bit. It quantifies a gradient $G_{i,t}(\mathbf{x}_t)$, and keeps the quantization error $\delta_{i,t}(\mathbf{x}_t)$ at the same time. The process of 1-bit SGD can be described using the following equations:

$$G_{i,t}^{quant}(\mathbf{x}_t) = Quant(G_{i,t}(\mathbf{x}_t) + \delta_{i,t}(\mathbf{x}_t)) \tag{10}$$

$$\delta_{i,t}(\mathbf{x}_t) = G_{i,t}(\mathbf{x}_t) - Quant^{-1}(G_{i,t}^{quant}(\mathbf{x}_t)) \tag{11}$$

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \gamma \frac{1}{n} \sum_{i=1}^{n} G_{i,t}^{quant}(\mathbf{x}_t) \tag{12}$$

where $Quant(\cdot)$ denotes the quantization function encoding the gradient, and $Quant(\cdot)$ the unquantizer decoding the gradient. Comparing to Eq. (5), the exchanging messages turn into $G_{i,t}^{quant}(\mathbf{x_t})$, which significantly reduces the communication volume.

The idea of 1-bit quantization is also used in [133]. Different from 1-bit SGD, Strom et al. [133] chose a fixed threshold, and then encoded the gradient elements higher than T with the value 1, those less than T with value 0. The absolute value of a gradient element less than T will not be sent, which is similar to the sparsification methods that will be discussed in the next section.

Some researchers further proposed adaptive quantization methods [168][169] with inexact proximal gradients, but these works lack the experiments of deep learning models.

Considering to simultaneously attain the communication efficiency and the good convergence of optimization, Alistarh et al. [134] presented a family of algorithms using quantization rather than just one quantization method. This family of quantization algorithms, named Quantized SGD (QSGD), can trade off the number of bits communicated with the variance of the compressed gradient. Alistarh et al. examined QSGDs performance by training DNNs on ImageNet [9] and CIFAR-10 datasets, and got as high accuracy as the original SGD.

In QSGD, Alistarh et al. exploited a probabilistic approach to quantify a vector. Given any vector $\mathbf{v} \in \mathbb{R}^N$, every j-th element $Quant_s(v_j)$ of quantization gradient $Quant_s(\mathbf{v})$ is quantized by $Quant_s(\cdot)$, corresponding to the element v_j of the original gradient $\mathbf{v}$. The stochastic quantization function is defined as

$$Quant_s(v_j) = \|\mathbf{v}\|_2 \cdot sgn(v_j) \cdot \psi_j(\mathbf{v}, s) \tag{13}$$

where $sgn(v_j) \in \{-1, +1\}$ denotes the sign of v_j, additionally, $sgn(0) = 1$. $\psi_j(\mathbf{v}, s)$ is defined as

$$\psi_j(\mathbf{v}, s) = \begin{cases} l/s & \text{with probability } 1 - p(\frac{|v_j|}{\|\mathbf{v}\|_2}, s); \\ (l+1)/s & \text{otherwise.} \end{cases} \tag{14}$$

in which l is an integer such that $|v_i| / \|\mathbf{v}\|_2 \in [l/s, (l+1)/s]$ s.t. $0 < l < s$, and $p(a, s) = as - l$ for any $a \in [0, 1]$. This is called Standard (random) dithering in [170], and is used in PCM coding [171][172]. Note that if $\mathbf{v} = \mathbf{0}$, the algorithm sets $Quant_s(\mathbf{v}, s) = \mathbf{0}$. The expectation $\psi_j(\mathbf{v}, s)$ satisfies $\mathbb{E}[\psi_j(\mathbf{v}, s)] = |v_i| / \|\mathbf{v}\|_2$. Then it is obvious that QSGD can assure that the quantized gradient is the unbiased estimation of the original vector, i.e., $\mathbb{E}[Quant_s(\mathbf{v})] = \mathbf{v}$, helping to achieve the convergence of training.

Wen et al. [136] developed another compressed communication scheme named TernGrad, which uses ternary gradients to speed up distributed deep learning. In TernGrad, the gradient $G(\mathbf{x})$ is quantized to a ternary level {-1, 0, 1} to reduce the communication data size. The gradient G($\mathbf{x}$) is ternarized as

$$\tilde{Q}_t(G(\mathbf{x}_t)) = ternarize(G(\mathbf{x}_t)) = s_t \cdot sign(G(\mathbf{x}_t)) \circ \mathbf{b}_t \tag{15}$$

where $s_t := max(abs(G(\mathbf{x}_t)))$, and $\circ$ represents the Hadamard product. The $sign(\cdot)$ is the same as $sgn(\cdot)$ in QSGD. Each element of $\mathbf{b}_t$ follows the distribution

$$\begin{aligned} P(b_{t,j} = 1 | G_t(\mathbf{x}_t)) &= |G_{t,j}(\mathbf{x}_t)/s_t| \\ P(b_{t,j} = 0 | G_t(\mathbf{x}_t)) &= 1 - |G_{t,j}(\mathbf{x}_t)/s_t| \end{aligned} \tag{16}$$

where $b_{t,j}$ and $G_{t,j}(\mathbf{x}_t)$ is the j-th element of $\mathbf{b}_t$ and $G_t(\mathbf{x}_t)$ respectively. Different from QSGD, Wen et al. proved the convergence from the perspective of statistic bound on gradients, and used the bound to guide practices. Also, TernGrad has an unbiased estimation of original gradient $G_t(\mathbf{x}_t)$.

A little different from the Parameter Server architecture, in TernGrad, Wen et al. proposed a technique named as Parameter Localization, in which each worker stores a copy of parameters locally. This technique changes the communication of parameters in the floating-point form to the transfer of quantized gradients, which makes the server-to-worker traffic smaller by only pulling the quantized gradients from servers. To minimize the number of levels when all workers are transferring different scalars, Wen et al. presented a new technique called Scalar Sharing, which chooses the maximum one in all scalars and shares it across all workers. In a large deep model, a maximum element could be much larger than most gradients, which could make the approximation weaker. Based on that, Wen et al. proposed layer-wise ternarizing and Gradient Clipping. Layer-wise ternarizing uses the layer-wise scalar in each layer instead of a large global maximum scalar.

Sign-SGD is another kind of quantization method [137]. In Sign-SGD, every worker quantifies the gradient to a binary

value, which is the sign of each coordinate of the gradient vector. Bernstein et al. [173] provided an extensive theoretical analysis of sign-based methods for non-convex optimization. They proved that when gradients are as dense or denser than stochasticity and curvature, Sign-SGD can converge with a theoretical rate. In [173], Bernstein et al. also proposed a new algorithm named Sign-SGD with a majority vote. After workers exchange the sign of their gradient vector to a server, the overall update is decided by a majority vote. They reduce the communication cost to 32x less by this method.

Wang et al. [174] developed a new method named Atomic Sparsification (ATOMO). They demonstrated that gradient sparsification and quantization are parts of a general approach that sparsifies the gradients in some atomic decomposition, such as entry-wise methods like QSGD, singular value decomposition(SVD), Fourier decomposition, etc. ATOMO aims to minimize the variance of the sparsified gradient that is sparse on the atomic basis and maintain it as an unbiased estimator of the original gradient. They illustrate that the 1-bit QSGD and TernGrad are special cases of ATOMO. Furthermore, they improved ATOMO with SVD, named as Spectral-ATOMO. In their experiments, Spectral-ATOMO reduces the training time by a factor of $2\times$ and $3\times$, compared to QSGD and TernGrad, respectively.

Mishchenko et al. [175] introduced a novel method DIANA, which extends the methods of [134][136] by splitting the whole gradient vector into some sub-vectors. And then, they quantified each sub-vector individually.

Sun et al. [176] proposed a communication-efficient algorithm named LAQ. This algorithm first quantifies the gradient difference of current gradient and previously quantized gradient and then skips some gradient communications if the gradient innovation of a worker is not large enough.

Horvth et al. [170] proposed one novel quantization method named Natural Compression. This compression method is defined as below:

$$C_{nat}(t) := \begin{cases} \text{sign}(t) \cdot 2^{\lfloor log_2|t| \rfloor}, & \text{with probability } p(t), \\ \text{sign}(t) \cdot 2^{\lfloor log_2|t| \rfloor}, & \text{with probability } 1 - p(t). \end{cases} \tag{17}$$

in which $p(t) := \frac{2^{\lceil log_2|t| \rceil} - |t|}{2^{\lfloor log_2|t| \rfloor}}$. They proved this compression method could attain a negligible variance, therefore benefits the training convergence. One unique advantage of this idea is that C_{nat} can dispense off the mantissa in the binary representation. And they proposed Natural Dithering which is similar to QSGD. It modifies the l in equation (14) into a binary geometric partition as: $l_1 = 1, l_2 = 2, l_3 = 4, ..., l_s = 2^{s-1}$.

Yu et al. [177] developed a new low-precision algorithm named AsyLPG, which is used in an asynchronous framework and simultaneously quantifies gradients and model parameters. It uses an additional requirement to restrict the quantization level. Then they combined the sparsification method with AsyLPG to further reduce the communication complexity.

VI. Sparsification Methods

The quantization methods can only achieve a maximum compression rate of 32x over the commonly used SGD with single-precision floating-point arithmetic. If one reduces the number of elements that would be transmitted, the compression rate will be further increased. A series of algorithms following this ideology is known as sparsification methods, in which only a part of elements are chosen and transmitted [178][179][180][181].

The origin of sparsification methods can be traced back to [139]. Langford et al. [139] proposed a general method called Truncated Gradient to get a sparse set of weights of online learning algorithms, addressing two problems, memory space constraints, and test-time constraints on computation. To induce sparsity, a natural thought is to round small coordinates of a gradient to zero. Truncated Gradient is a less extreme version, in which the small coordinates will not be directly set at zero, and the large coordinates (exceed a threshold) will be kept with their original values. This Truncated Gradient is the first sparsification technique for large-scale learning.

Sparsification communication methods can be divided into four main categories: coordinate descent, random or deterministic sparsification, and proximal methods.

A. Random Sparsification

Random Sparsification is to select some entries to communicate and update randomly. This ideology is also named as random-k, where k denotes the number of elements selected.

In [140], Konecny et al. used Random Mask, a pre-defined random sparsity pattern, to change the update parameters $\mathbf{H}$ into a sparse matrix $\hat{\mathbf{H}}$. This random pattern is generated afresh in each iteration of training. In addition, it can be initialized in each client independently, or be created by the server and then distributed to the workers. Using the former, each worker should send both indices and values of the non-zero entries of $\mathbf{H}$. For the later, workers only need to send the values of the non-zero entries of $\mathbf{H}$, because all indices of non-zero entries in every worker are the same as others. Konecny et al. proposed another technique named Subsampling. Similar to Random Mask, each worker also randomly chooses a subset of the $\mathbf{H}$. Different from Random Mask, the sparse communication matrix $\hat{\mathbf{H}}$ will be scaled to let the $\mathbf{E}(\hat{\mathbf{H}}) = \mathbf{H}$. So, Subsampling is an unbiased estimator of the true average.

In [141], Wangni et al. proposed to randomly drop out the coordinates with a probability vector $\mathbf{p}$, and then amplify the non-zero coordinates from g_j to g_j/p_j. The whole process of this method can be described as follows:

$$\begin{aligned} \text{original vector } \mathbf{g} &= [g_1, g_2, \cdots, g_N], \\ \text{probability vector } \mathbf{p} &= [p_1, p_2, \cdots, p_N], \\ \text{selection vector } \mathbf{Z} &= [Z_1, Z_2, \cdots, Z_N], \\ \text{sparsified vector } \mathbf{Q}_{spar}(\mathbf{g}) &= \left[Z_1 \frac{g_1}{p_1}, Z_2 \frac{g_2}{p_2}, \cdots, Z_N \frac{g_N}{p_N}\right] \end{aligned} \tag{18}$$

unbiased是一个很重要的事情

Then there is

$$\mathbb{E}\left[\mathbf{Q}_{spar}(\mathbf{g})_i\right] = p_i \times \frac{g_i}{p_i} + (1-p_i) \times 0 = g_i \tag{19}$$

which indicates that the sparsified vector $\mathbf{Q}_{spar}(\mathbf{g})$ is an unbiased estimator of the original vector g, similar to [134], [140].

B. Deterministic Sparsification

Different from Random Sparsification, the sparse properties are guaranteed by Deterministic Sparsification [182][183], in which the most weights of fully connected deep models can be close to zero. Due to the sparse weights, most of the gradients in each iteration are also around zero so that the zeroed gradients are no need for communication to reduce the communication traffic. There are mainly two ways to sparsify the gradients: Fixed Threshold and Adaptive Threshold.

1) Fixed Threshold: Strom et al. [133] introduced a new method to solve the communication bottleneck problem. In his algorithm, those gradient elements with absolute values less than a pre-defined threshold will be discarded. Because not all gradient elements are transmitted, the server must know which gradients to be transmitted and the indices of them. Strom et al. constructed key-value maps where keys are the indices and values are the corresponding gradient elements. The main drawback of the fixed threshold method is that it is non-trivial to choose a suitable threshold for different deep models or different iterations. Even worse, when Error Feedback (introduced in Section IX) techniques are used, the fixed threshold method will result in a large number of gradients being transmitted.

2) Adaptive Threshold: To address the problem of fixed threshold, Top-k sparsification algorithms [184][185] well select the top-k gradients (in terms of absolute values) at each iteration.

Dryden et al. [142] proposed an adaptive threshold, which uses a fixed proportion π to indicate the proportion of gradient elements to transmit. In every iteration, the algorithm will determine a positive threshold and a negative one to satisfy the desired proportion. This method is able to make sure that the compressing ratio will not change during the training process. Despite that this technique requires the sorting of the whole gradient vector elements and costs extra computing time, it still cuts down the wall-lock time a lot.

Aji & Heaeld [57] presented another adaptive threshold method, which only chooses one absolute threshold rather than two. All gradients whose absolute value is smaller than this threshold will be dropped. The parameters and the corresponding gradients may not have comparable scales across different parts of the neural network. If all gradients are compared with one global threshold, some gradients of small scale will be possibly never transmitted. Considering this problem, Aji & Heaeld made use of layer normalization [186] to make a global threshold work.

Depending on local gradient activity, Chen et al. [187] proposed a novel gradient compression scheme, AdaComp, which can self-adapt the compression rate. Chen et al. showed that most of the gradient compression techniques do not work well for convolutional layers. The different types of neural layers, mini-batch sizes, and other factors may affect the compression rate. They proposed AdaComp to determine the sparsification level, adapting to all variations automatically.

Scattler et al. [188] combined sparsification and quantization methods together and proposed a new compression algorithm named Sparse Binary Compression (SBC). The algorithm cuts off the elements of low absolute value, and then averages the positive values and negative values to get the positive mean μ^+ and negative mean μ^- respectively. If the μ^+ is bigger than μ^-, SBC discards all negative elements and sets all positive elements to μ^+, vice versa. At last, it quantizes the non-zero elements, further reducing the communication cost by a factor of $\times 3$.

Sattler et al. [189] further exploited the combination of sparsification and quantization to develop a new method named Sparse Ternary Compression (STC). Specifically, they made use of Top-k sparsification and Ternary quantization. Different from SBC, STC works well for the Federated Learning [190]. In [189], they experimentally demonstrated that sparsification methods achieve better convergence than quantization methods.

In many distributed training algorithms, the workers pull the newest update from the master before the training starts in order to make their models not deviate too much from the global model. In most of the Top-k sparsification methods, the size of the latest update depends on the number of workers. Many indices of chosen elements of a gradient vector are different from work to work. When all gradients being collected together, the elements of the global gradient vector proliferate almost linearly. So the growth of the number of workers leads to an increase in the global gradient elements. This incurs an ineffective sparsity of master-to-workers communication. To address this problem, Shi et al. [191] proposed a novel sparsification method gTop-k. After aggregating all gradients, gTop-k sparsifies the global gradient vector once again, reducing the master-to-workers communication load and attaining convergence simultaneously [192]. By exploiting the properties of gradients that are empirically proved to follow bell-shaped distributions [193], the computing-efficient approximation of top-k can be further exploited. Adaptive selecting the number of gradients or model parameters for communication is also helpful for reducing the overall training time [194].

C. Coordinate Descent

Coordinate descent is a kind of optimization method [195][196][197][198][199][200][201] that splits all variables into multiple blocks, and then updates one block while fixing the remaining blocks. In an iteration, all blocks will be updated one by one [197]. Although the gradient-based methods have achieved great success, they may still suffer from the vanishing gradient problem for training deep neural networks [202]. Gradient-free methods have been recently proposed to ad-

dress the vanishing gradient problem, including BCD methods [200][198]. In distributed deep learning scenarios, BCD can be easily implemented in a distributed and parallel manner [199]. BCD has a property that only a subset of variables would be updated in each round, and it is similar to the sparsification of distributed SGD.

Lau et al. [200] proposed an efficient BCD algorithm and provided its convergence analysis. They clarified three major advantages of BCD: (1) higher per epoch efficiency than SGD at early stage; (2) good scalability; (3) gradient-free. In [203], Zeng et al. provided a general methodology for provable convergence guarantees for using BCD in deep learning.

Mishchenko et al. [204] developed a new algorithm named Independent Block Coordinate Descent (IBCD), which is a variant allowing each worker to sample an independent subset of blocks instead. They proved that the optimal number of blocks to be updated by each of n units in every iteration is equal to m, where m is the total number of blocks. To be more specific, this means that when $n = 100$ parallel units are used, 99% of work is a waste of time.

D. Proximal Methods

Proximal methods include two kinds of sparsity-inducing regularization in learning models and solving the optimization problem with proximal-based algorithms. These methods are used in sparsity learning, which is able to cut down the number of parameters in deep learning models. Also, the communication of distributed deep learning can benefit from the sparsity. $L0$-and $L1$-norms of parameters are most frequently used for these kinds of methods [205][196].

In [106], Grishchenko et al. firstly combined the Proximal method, coordinate descent, and random sparsification together. For the workers, this method computes a local update, including a randomly selected subset of coordinates. For the master, it aggregates and averages all updates from workers that join the computation in the current iteration, then computes the proximity operator of the regularizer at the averaged updates.

Tsuzuku et al. [206] developed a novel method to reduce the communication overhead. When training a batch of samples, some gradient elements are ambiguous [206]. It means that some elements have low-amplitude but high-variance values, which may bring futile updates to the global model. Based on this observation, it only sends elements with small enough variance to get the sparsity of communication. By controlling the hyper-parameters, their algorithm is able to attain the high-rate sparsity.

Ivkin et al. [207] exploited a technique that is widely adopted in distributed systems, Count Sketch [208]. It compresses a gradient vector G into a sketch $S(G)$ of size $O(1/\epsilon \log n)$., which can approximate every coordinate of G and the l_2 norm of the entire gradient. Every worker sends this sketched gradient to the server, and the server recovers the d largest coordinates of the sum of gradients and then performs the update.

VII. Scheduling of Communication and Computing

Due to the layer-wise structure of deep models, the communication and computation tasks during the training process (e.g., one iteration) can be executed in parallel. The parallelism of communication and computing can further hide the communication time to reduce the overall training time. The communication and computation tasks can be formulated to a general directed acyclic graph (DAG) [71] for pipelining or scheduling such that the iteration time is minimal.

In 2017, wait-free backward propagation (WFBP) [63][47] was proposed for pipelining the gradient communication of layer l and the gradient computation of layer $l-1$ due to their independence. WFBP is naively supported in current deep learning frameworks (e.g., TensorFlow, MXNet, Horovod, etc.). However, for small tensors of gradients, the latency term (startup time) easily dominates the communication cost especially on extremely large-scale clusters. To address the problem, merged-gradient (or tensor fusion) techniques (e.g., MG-WFBP [64]) are required to alleviate the negative impact of the startup time. In the layer-wise gradient sparsification, one can pipeline three types of tasks (i.e., gradient computation, gradient sparsification and gradient communication). For the large tensors, on the other hand, it takes a long communication time so that their previous small tensors need to wait. To minimize the waiting time of different tasks, the order of communication and computation tasks can be scheduled. [68][65][66][67] scheduled the communication tasks and computation tasks by changing the order of execution. Peng et al. [67] proposed tensor partitioning to communication scheduling (even feed-forward computations can be paralleled with communications) to further reduce the communication cost. Noticing that multiple communication tasks may deteriorate the training performance, Wang et al. [209] proposed a communication contention aware scheduling of multiple deep learning training jobs on GPU clusters.

VIII. Convergence Analysis

There are some commonly used assumptions for the convergence analysis:

1) **Lipschitzian continuous gradient:** All function $f_i(\cdot)$ have L-Lipschitzian gradients:

$$\|\nabla f_i(\mathbf{x}) - \nabla f_i(\mathbf{y})\| \leq L \|\mathbf{x} - \mathbf{y}\|, \ \forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^n. \quad (20)$$

2) **Unbiased stochastic gradient:** The gradient calculated in every iteration is an unbiased estimation of the gradient of $f_i(\mathbf{x})$:

$$G_i(\mathbf{x}) := \mathbb{E}_{\xi \sim \mathcal{D}_i} \nabla F_i(\mathbf{x}; \xi) = \nabla f_i(\mathbf{x}), \ \forall \mathbf{x} \in \mathbb{R}^n, \quad (21)$$

in which the $\mathcal{D}_i$ is the data distribution on i-th worker.

3) **Bounded variance:** The variance of the stochastic gradient is bounded:

$$\mathbb{E}_{\xi \sim \mathcal{D}_i} \|\nabla F_i(\mathbf{x}; \xi) - \nabla f_i(\mathbf{x})\|^2 \leq \sigma^2, \forall i, \forall \mathbf{x} \in \mathbb{R}^n. \quad (22)$$

4) **Bounded stochastic gradient:** The second moment of the stochastic gradients is bounded:

$$\mathbb{E}_{\xi\sim\mathcal{D}_i}\|\nabla F_i(\mathbf{x};\xi)\|^2 \leq M^2, \forall i, \forall \mathbf{x}\in\mathbb{R}^n. \quad (23)$$

5) **μ-Strongly convex function:**

$$f(\mathbf{x}) \geq f(\mathbf{y}) + \langle \nabla f(\mathbf{y}), \mathbf{x}-\mathbf{y}\rangle + \frac{\mu}{2}\|\mathbf{x}-\mathbf{y}\|^2. \quad (24)$$

For the gossip (peer-to-peer) algorithms, there are some extra assumptions [54]:

1) **Symmetric doubly stochastic matrix:** The weighted matrix W is a real double stochastic matrix that satisfies $W = W^T$ and $W\mathbf{1} = W$. Here the W represents the communication topology of workers.
2) **Spectral gap:** Given the symmetric doubly stochastic matrix W, the spectral gap is defined as $\rho := \max\{\|\lambda_2(W)\|, \|\lambda_n(W)\|\}$, and $\rho < 1$.

For the compression methods, there are some assumptions:

1) **k-contraction [184]:** For a parameter $0 < d < n$, a k-contraction operator is a (possibly randomized) operator $C(\cdot)$: $\mathbb{R}^n \to \mathbb{R}^n$ that satisfies the contraction property:

$$\mathbb{E}\|\mathbf{x} - C(\mathbf{x})\|^2 \leq \left(1 - \frac{d}{n}\right)\|\mathbf{x}\|^2, \forall \mathbf{x}\in\mathbb{R}^n. \quad (25)$$

2) **Unbiased compression [54]:** The stochastic compression operator $C(\cdot)$ is unbiased:

$$\mathbb{E}[C(\mathbf{x})] = \mathbf{x}, \quad (26)$$

for any $\mathbf{x}$ and the compression operators are independent on different workers or at different iterations.

A. PS/All-Reduce Architecture

1) BSP: PS architecture and the All-Reduce architecture have the same iteration equations because the All-Reduce algorithm only changes the way of implementing global synchronization. Then the convergence analysis of the PS architecture is also suitable for the All-Reduce architecture.

For quantization methods, Christopher et al. [210] provided a convergence analysis with martingale-based approach under both convex and non-convex objectives. As for a further quantization method, QSGD, Alistarh et al. [134] not only proposed a family of quantization algorithms, but also provide its convergence analysis. They proved that QSGD can attain convergence both for convex and non-convex objectives. They also proved QSGD satisfies that $\frac{1}{L}\mathbb{E}\left[\|\nabla f(\mathbf{x})\|_2^2\right] \leq O\left(\frac{\sqrt{L(f(\mathbf{x}_1)-f^*)}}{T} + \frac{min(n/s,\sqrt{n}/s)B}{L}\right)$, where L represents Lipschitzian constant, s and B are hyper-parameters.

When applying error accumulation on quantization methods, the variance bound of quantized gradients exceed that in QSGD [135]. Wu et al. [135] presented a convergence analysis of this situation. One limitation of their analysis is the restriction of unbiased gradient compression.

For sparsification methods, Alistarh et al. [185] theoretically proved that the Top-k algorithm can attain a good convergence even with biased estimation and non-convex objectives. This analysis suffers from a deceleration proportional to k.

Extending convergence analysis of Top-k methods to more a general range, random-k or k-sparsification methods, Stich et al. [184] proved that this scheme preserves the same order of convergence rate as the vanilla SGD, i.e. $O\left(\frac{G^2}{\mu T}\right)$.

Shi et al. [192] analyzed the gTop-k sparsification method [191] theoretically. They proved that gTop-k S-SGD provides convergence guarantees for non-convex problems and has the same theoretical convergence rate as the mini-batch SGD.

2) SSP/Asynchronous: Ho et al.[33] established $O(1/\sqrt{T})$ time for SGD under SSP.

Lian et al. [72] proposed an ergodic convergence rate $O(1/\sqrt{T})$ and proved that the linear speedup can be achieved if the number of workers is bounded by $\sqrt{T}$.

Alistarh et al. [211] provided the convergence bounds for lock-free SGD. Moreover, they exhibit a fundamental trade-off between the delay of the system and the convergence rate.

Zhang et al. [212] also provided a convergence rate of Asynchronous SGD under non-convex object function. They also establish a unifying condition for Asynchronous SGD.

3) Local SGD: Stich [73] and Yu et al. [46] provided a concise convergence analysis for local SGD, showing this method has convergence guarantees with reducing communication cost. In [41], Haddadpour et al. strengthened the convergence analysis for local SGD, and showed that local SGD can be far less expensive and applied far more generally than current theory suggests. They proved that for loss functions that satisfy the Polyak-ojasiewicz condition, $O((pT)1/3)$ rounds of communication suffice to achieve a linear speedup, that is, an error of $O(1/pT)$. Patel and Dieuleveut [213] proposed a non-asymptotic error analysis which enables the comparison to one-shot averaging and mini-batch averaging, provided adaptive lower bounds on the communication frequency and showed that the local-SGD could reduce communication by a factor of $O(\frac{\sqrt{T}}{N^{3/2}})$.

B. Gossip Architecture

Different from the centralized architecture, all of the workers own an individual model in the gossip architecture. Therefore, the convergence analysis of the gossip architecture differs from the centralized architecture. In both convergence analysis and algorithm designing of the gossip architecture, consensus should be guaranteed.

1) BSP: Lian et al. [51] firstly provided a theoretical analysis that decentralized SGD algorithms for both convex and non-convex objectives can be faster than centralized SGD with less communication on the busiest node. They proved the convergence rate of decentralized SGD is $O\left(\frac{1}{T} + \frac{1}{\sqrt{nT}}\right)$, in which T and n represents the number of iterations and workers respectively. When T is large enough, the $\frac{1}{\sqrt{nK}}$ term dominate. In this situation, the convergence rate approximately changes into $\frac{1}{\sqrt{nK}}$, which indicates the linear speedup achieved with the number of workers.

Considering communication compression, Tang et al. [54] proposed two algorithms, ECD-PSGD and DCD-PSGD, with

detailed convergence analysis on both convex and non-convex problems. The convergence rate of DCD-PSGD is $O\left(\frac{\sigma}{\sqrt{nT}}+\frac{\zeta^{\frac{2}{3}}}{T^{\frac{2}{3}}}+\frac{1}{T}\right)$, in which σ is the variance of stochastic gradient estimator, ζ is the variance of gradient divergence between single worker and all workers. For ECD-PSGD, they proved its convergence rate is $O\left(\frac{\sigma\left(1+\frac{\tilde{\sigma}^2 logT}{n}\right)}{\sqrt{nT}}+\frac{\zeta^{\frac{2}{3}}\left(1+\frac{\tilde{\sigma}^2 logT}{n}\right)}{T^{\frac{2}{3}}}+\frac{1}{T}+\frac{\tilde{\sigma}^2 logT}{T}\right)$, which is slightly worse than DCD-PSGD, due to the extra terms $O\left(\frac{\sigma\tilde{\sigma}^2 logT}{n\sqrt{nT}}+\frac{\zeta^{\frac{2}{3}}\tilde{\sigma}^2 logT}{nT^{\frac{2}{3}}}+\frac{\tilde{\sigma}^2 logT}{T}\right)$.

The restraint of convergence analysis in ECD-PSGD and DCD-PSGD is that they only support unbiased compression operators. Koloskova et al. [164] proposed the first method CHOCO-SGD that can tolerate biased compression operators. In the condition of convex objectives, their algorithm is able to converge with rate $O\left(1/(nT)+1/(T\delta^2\omega)\right)$, in which δ denotes the eigen gap of the gossip matrix and $\omega \leq 1$ the compression ratio.

Koloskova et al. [214] proved that CHOCO-SGD [164] can converges at the rate $O(1/\sqrt{nT}+n/\left(\rho^4\sigma^2T\right))$ on non-convex smooth functions, where n denotes the number of nodes, T the number of iterations, ρ the spectral gap of the mixing matrix and σ the compression ratio. Additionally, they showed that the CHOCO-SGD can converge with any arbitrary high compression operators and achieve linear speedup.

2) Asynchronous: Jeff et al. [215] provided a sketch of proof of convergence without the convergence rate.

Lian et al. [216] proposed a theoretical analysis of asynchronous gossip SGD with the non-convex objective function, which converges with the $O(1/\sqrt{T})$ rate as SGD and has linear speedup with respect to the number of workers.

Assran et al. [53] provided the theoretical guarantees that another asynchronous gossip SGD algorithm is able to attain a good convergence similar to [216].

A full comparison of different algorithms is shown in Table IV.

IX. Auxiliary Technologies

With the help of some auxiliary technologies, communication compression methods can attain convergence and reduce communication loads simultaneously. Some popular powerful auxiliary technologies are introduced in the following.

A. Error Accumulation

In 1-bit SGD [132], error accumulation adds the current quantization error into the next mini-batch gradient before quantization in next iteration. This error-feedback ensures that all gradients are eventually added up into the model with full accuracy, although being delayed. This process is like equation (10), (11) and (12).

Karimireddy et al. [138] proposed EF-SIGNSGD (SIGNSGD with Error-Feedb). In EF-SIGNSGD, the error is locally stored and added to the next step, also belonging to error accumulation.

With many formulations [217][57][188][187][142][175][135][138][189][133][57][206] of error accumulation, we can summarize the error accumulation as follows:

$$\begin{aligned}
&\text{gradient compression} && C_{i,t} = Sparse(v_{i,t-1}+\nabla_{i,t}),\\
&\text{error accumulation} && v_{i,t} = \nabla_{i,t} - C_{i,t}\\
&\text{update weight} && x_{t+1} = x_t - \gamma\frac{1}{n}\sum_{i=1}^{n} C_{i,t}
\end{aligned}$$

where $C_{i,t}$ represents the updates which are compressed by any compression method $Sparse(\cdot)$, ∇ the gradient, at t-th iteration and in worker i.

In [135], Wu et al. proposed ECQ-SGD (Error Compensated Quantized SGD). They consider not only compression error in the current iteration, but all previous error. Different from the aforementioned methods, ECQ-SGD accumulated all compression error.

B. Momentum Correction

In [25], Lin et al. proposed the Momentum Correction technique to help DGC making use of Momentum SGD. They apply the vanilla momentum SGD [222] and error accumulation to the sparse gradient, the update process is:

$$\begin{aligned}
&\text{momentum accumulation} && u_{i,t} = mu_{i,t-1}+\nabla_{i,t},\\
&\text{error accumulation} && v_{i,t} = v_{i,t-1}+u_{i,t},\\
&\text{update weight} && x_{t+1} = x_t - \gamma\sum_{i=1}^{n} sparse(v_{i,t})
\end{aligned}$$

where m denotes the coefficient of momentum, $u_{i,t}$ denotes the momentum at t-th iteration on worker i. Momentum Correction has also been exploited in [188][175][189].

Shen-Yi Zhao et al. [218] proposed one novel technique named as Global Momentum Compression, of which the update process becomes:

$$\begin{aligned}
&\text{momentum} && u_{i,t} = \nabla_{i,t} - m(x_t - x_{t-1}),\\
&\text{error} && v_{i,t} = v_{i,t-1}+u_{i,t} - sparse(v_{i,t-1}+u_{i,t}),\\
&\text{update weight} && x_{t+1} = x_t - \gamma\sum_{i=1}^{n} sparse(v_{i,t-1}+u_{i,t})
\end{aligned}$$

C. Local Gradient Clipping

Gradient Clipping [223] is a technique widely used in vanilla SGD to avoid the exploding gradient problem. It clips all gradients owning values too big to a user-defined threshold. For the distributed SGD, Yujun Lin et al. [25] modified it into Local Gradient Clipping, which is performed before adding the error accumulation term and the gradient in current iteration. The k-th worker has a threshold $thr\left(G_k\right)$ for its local gradient G_k, and the aggregation of gradients has a threshold $thr\left(G\right)$ for the global gradient $G := \sum_{k=1}^{N} G_k$. Assuming all N workers have IID gradient distribution, with the variance σ^2, then the aggregation of all gradients have the variance $N\sigma^2$. So there are

$$E\left[\|G_k\|_2\right] \approx \sigma,\ E\left[\|G\|_2\right] \approx N^{1/2}\sigma. \tag{27}$$

TABLE IV
SUMMARY OF DISTRIBUTED LEARNING ALGORITHMS

Arch.	Comm.	Compression	Communication Cost in Big O		Convergence in Big O		References of Method
			Server	Workers	convex	non-convex	
PS	BSP	Null	$O(32nNT)$	$O(32NT)$	$O(\frac{1}{T})$[26]	$O(\frac{1}{\sqrt{T}})$[26]	[95][96][97][98]
		Quant.	$O(32nNT)$	$O(bNT)$	$O(\frac{1}{T})$[175] [112]	$O(\frac{1}{\sqrt{T}})$ [175][173] [112][134][138][132]	[217] [140][175][173][136][170] [137] [174][176][204][134][133][142][138]
		Spars.	$O(32nNT)$	$O(k(\log N+32)T)$	$O(\frac{1}{T})$[207][184]	$O(\frac{1}{\sqrt{T}})$ [185][192]	[217][57] [140][207][170][174] [141][25][142][218][191][206]
	SSP	Null	$O(32N\sum_i^n T_i)$	$O(32NT_i)$	-	$O(\frac{1}{\sqrt{T}})$ [33]	[102][33]
		Quant.	-	-	-	-	-
		Spars.	-	-	-	-	-
	ASP	Null	$O(32N\sum_i^n T_i)$	$O(32NT_i)$	$O(\frac{1}{T})$[73][72]	$O(\frac{1}{\sqrt{T}})$ [212]	[73]
		Quant.	$O(32N\sum_i^n T_i)$	$O(bNT_i)$	$O(\frac{1}{T})$[177]	$O(\frac{1}{\sqrt{T}})$[134][132]	[177][134][132][181]
		Spars.	$O(32N\sum_i^n T_i)$	$O(k(\log N+32)T_i)$	$O(\frac{1}{T})$[141][219]	-	[141][106][219][181]
	L-SGD	Null	$O(32N\frac{T}{T_{comm}})$	$O(32N\frac{T}{T_{comm}})$	$O(\frac{1}{T})$[73]	$O(\frac{1}{\sqrt{T}})$ [41][111][113]	[19][73] [41][121][111][113]
		Quant.	$O(32N\frac{T}{T_{comm}})$	$O(bN\frac{T}{T_{comm}})$	-	$O(\frac{1}{\sqrt{T}})$[112]	[189][112][181]
		Spars.	$O(32N\frac{T}{T_{comm}})$	$O(k(\log N+32)\frac{T}{T_{comm}})$	-	-	[189][181]
A.R.	BSP	Null	-	$O(32NT)$	$O(\frac{1}{T})$[26]	$O(\frac{1}{\sqrt{T}})$[26]	[95][96][97][98]
		Quant.	-	$O(bNT)$	$O(\frac{1}{T})$[175] [112]	$O(\frac{1}{\sqrt{T}})$ [175][173] [112][134][138][132]	[217] [140][175][173][136][170] [137] [174][176][204][134][133][142][138]
		Spars.	-	$O(kn(\log N+32)T)$	$O(\frac{1}{T})$[207][184]	$O(\frac{1}{\sqrt{T}})$ [185][192]	[217][57] [140][207][170][174] [141][25][142][218][191][206]
	L-SGD	Null	-	$O(32N\frac{T}{T_{comm}})$	$O(\frac{1}{T})$[73]	$O(\frac{1}{\sqrt{T}})$ [41][111][113]	[19][73] [41][111][121][111][113]
		Quant.	-	$O(bN\frac{T}{T_{comm}})$	-	$O(\frac{1}{\sqrt{T}})$[112]	[189][112][181]
		Spars.	-	$O(kn(\log N+32)\frac{T}{T_{comm}})$	-	-	[189][181]
Gossip	BSP	Null	-	$O(32Nn_{peers}T)$		$O(\frac{1}{\sqrt{T}})$[51]	[51]
		Quant.	-	$O(nbN_{peers}T)$	-	$O(\frac{1}{\sqrt{T}})$[135]	[54][164][168][165][135][220]
		Spars.	-	$O(k(\log N+32)n_{peers}T)$	-	-	[164][55]
	ASP	Null	-	$O(32Nn_{peers}T_i)$	-	$O(\frac{1}{\sqrt{T}})$ [53]	[53]
		Quant.	-	-	-	-	-
		Spars.	-	-	-	-	-
	L-SGD	Null	-	$O(32Nn_{peers}\frac{T}{T_{comm}})$	$O(\frac{1}{T})$ [221]	$O(\frac{1}{\sqrt{T}})$ [221]	[221]
		Quant.	-	-	-	-	-
		Spars.	-	-	-	-	-
Pipelining			[63][47][64][92][93][94]				
Scheduling			[64][65][66][67][68][69][70]				

Notes: 1) The "Arch." indicates the architecture supported in the original paper, "A.R." represents All-Reduce, "Quant." quantization and "Spars." sparsification. The "Comm." column indicates the communication scheme which includes "BSP" (Bulk Synchronous Parallel), "SSP" (Stale Synchronous Parallel), "ASP" (ASynchronous Parallel), and "L-SGD"(Local SGD). 2) Some methods use both compression techniques and both Asyc and Local SGD. 3) Some methods also optimize download communication, we have listed them together with upload communication. 4) The communication complexity and convergence rate of some paper maybe different, we just list out the common ones. 5) Pipelining and scheduling can be used in many methods so we only list methods that uses pipeline without classifying it into any category.

Scaling the threshold by $N^{-1/2}$, we have

$$thr\left(G_k\right) = N^{-1/2} \cdot thr\left(G\right). \tag{28}$$

By adjusting the threshold of clipping with respect to the numbers of workers, Local Gradient Clipping can restore the original variance of the aggregation models.

D. Warm-up Training

When the neural network just begins to train for several epochs, the gradient values are so big that the network varies rapidly. Compressing these early gradients restricts the evolving ability of the neural network in the first several epochs. In order to elude this problem, Yujun Lin et al. [25] make use of Warm-up training [224] in DGC. This technique splits the training process into two periods, warm-up period and the normal training period. Firstly, the algorithm trains the model in the warm-up period, using a less aggressive learning rate and also less aggressive gradient sparsity, which can help to reduce the number of extreme gradients being delayed. And then, the gradient sparsity increases exponentially from a small value to the final value. Then the algorithm trains the model in a high sparsity and decreasing the learning rate as vanilla SGD.

X. Related Work

Two surveys [225][226] provide a general introduction of distributed machine learning algorithms for dealing with big data. Xing et al. [227] discuss distributed machine learning on big data, mainly covering different synchronization schemes, scheduling, and balancing workloads and communication typologies. Ben-Nun et al. [228] provide a review of distributed deep learning, mostly focusing on DNN operators and some main approaches for parallelism for both training and inference. Guo [229] gives a thorough review of different quantized neural networks, which offers some insights to quantized SGD optimization. Zhang et al. [230] provide a quick survey on large-scale distributed deep learning systems, which concisely introduces parallelisms, parameter server architectures, synchronization schemes, related applications, and platforms.

Different from the two surveys [228][230] that are mostly related to ours, we mainly focus on communication-efficient distributed deep learning training. The two surveys [228][230] lack of detailed discussions on communication compression techniques, which are introduced explicitly in this survey. Moreover, we give a quick review of some auxiliary technologies and also provide the convergence bound comparisons.

XI. Conclusion and Future Directions

In this survey, we provided a thorough introduction to the communication-efficient distributed deep learning algorithms. We summarized the taxonomy of distributed deep learning and classified the communication-efficient distributed training algorithms in four main dimensions: 1) synchronous scheme, 2) system architecture, 3) compression techniques, and 4) parallelism of communication and computing. For each dimension, we provided a comprehensive introduction to related techniques in addressing the communication problems. Furthermore, we provided a review of convergence bounds of different algorithms and some auxiliary techniques in helping accelerate the training speed.

We summarize some challenges and future directions below.

1) **More combinations of different communication schemes.** 1) Can local SGD be combined with the gossip architecture? Both local SGD and the gossip architecture will make model consistency low. 2) Can compression techniques be combined with local SGD or the gossip architecture? The combinations of different communication reduction methods can further reduce the communication cost, while they could make the deep model not convergent.
2) **Higher compression level.** Can a higher compression level be implemented without losing the training performance? The current quantization method can reduce data size $\times 32$, and sparsification $\times 1000$. How to make the compression ratio higher while preserving the model accuracy is a more difficult question.
3) **Adaptive Compression.** Gradient/model compression can reduce the communication size (thus the communication time). However, with a very high compression ratio, it generally requires a larger number of iterations to achieve the the target optimization error. Therefore, it is also challenging to balance the compression ratio and the convergence speed. For example, is it possible to set different compression ratios for different layers/tensors in a deep model or for different peers with various network bandwidth to achieve optimal system throughput?
4) **Fault-tolerant algorithms.** In a stable computing cluster, the algorithm is able to run without interference. But when one wants to apply a large number of heterogeneous devices to train the deep models, one may have lots of uncertainty, such as the severe straggler, network congestion, worker failure. To develop more fault-tolerant algorithms is an important direction to make the training more reliable.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016, pp. 770–778.

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17. USA: Curran Associates Inc., 2017, pp. 6000–6010.

[3] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2018.

[4] S. Yang, Y. Wang, and X. Chu, "A survey of deep learning techniques for neural machine translation," *arXiv preprint arXiv:2002.07526*, 2020.

[5] D. Amodei, S. Ananthanarayanan, R. Anubhai *et al.*, "Deep speech 2: End-to-end speech recognition in english and mandarin," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ser. ICML'16. JMLR.org, 2016, pp. 173–182.

[6] T. Reinartz, *Focusing Solutions for Data Mining: Analytical Studies and Experimental Results in Real-world Domains*. Berlin, Heidelberg: Springer-Verlag, 1999.

[7] C. Sun, A. Shrivastava, S. Singh, and A. Gupta, "Revisiting unreasonable effectiveness of data in deep learning era," in *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 843–852.

[8] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "Imagenet large scale visual recognition challenge," *Int. J. Comput. Vision*, vol. 115, no. 3, pp. 211–252, Dec. 2015.

[9] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, June 2009, pp. 248–255.

[10] Y. Wang, Q. Wang, S. Shi, X. He, Z. Tang, K. Zhao, and X. Chu, "Benchmarking the performance and power of AI accelerators for AI training," *arXiv preprint arXiv:1909.06842*, 2019.

[11] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "cuDNN: Efficient primitives for deep learning," *arXiv preprint arXiv:1410.0759*, 2014.

[12] S. Shi, Q. Wang, P. Xu, and X. Chu, "Benchmarking state-of-the-art deep learning software tools," in *2016 7th International Conference on Cloud Computing and Big Data (CCBD)*. IEEE, 2016, pp. 99–104.

[13] P. Xu, S. Shi, and X. Chu, "Performance evaluation of deep learning tools in docker containers," in *2017 3rd International Conference on Big Data Computing and Communications (BIGCOM)*. IEEE, 2017, pp. 395–403.

[14] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[15] D. Yan, W. Wang, and X. Chu, "Optimizing batched winograd convolution on GPUs," in *Proceedings of the 25th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2020, pp. 32–44.

[16] T. Chilimbi, Y. Suzue, J. Apacible, and K. Kalyanaraman, "Project adam: Building an efficient and scalable deep learning training system," in *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'14. Berkeley, CA, USA: USENIX Association, 2014, pp. 571–582.

[17] E. P. Xing, Q. Ho, W. Dai, J. K. Kim, J. Wei, S. Lee, X. Zheng, P. Xie, A. Kumar, and Y. Yu, "Petuum: A new platform for distributed machine learning on big data," *IEEE Transactions on Big Data*, vol. 1, no. 2, pp. 49–67, June 2015.

[18] P. Moritz, R. Nishihara, I. Stoica, and M. I. Jordan, "Sparknet: Training deep networks in spark," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[19] M. A. Zinkevich, M. Weimer, A. Smola, and L. Li, "Parallelized stochastic gradient descent," in *Proceedings of the 23rd International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS'10. USA: Curran Associates Inc., 2010, pp. 2595–2603.

[20] Y. You, Z. Zhang, C.-J. Hsieh, J. Demmel, and K. Keutzer, "Imagenet training in minutes," in *Proceedings of the 47th International Conference on Parallel Processing*. ACM, 2018, p. 1.

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

[22] S. Shi, Q. Wang, and X. Chu, "Performance modeling and evaluation of distributed deep learning frameworks on gpus," in *2018 IEEE 4th Intl Conf on Big Data Intelligence and Computing*. IEEE, 2018, pp. 949–957.

[23] X. Jia, S. Song, S. Shi, W. He, Y. Wang, H. Rong, F. Zhou, L. Xie, Z. Guo, Y. Yang, L. Yu, T. Chen, G. Hu, and X. Chu, "Highly scalable deep learning training system with mixed-precision: Training ImageNet in four minutes," in *Proc. of Workshop on Systems for ML and Open Source Software, collocated with NeurIPS 2018*, 2018.

[24] Y. You, J. Li, S. Reddi, J. Hseu, S. Kumar, S. Bhojanapalli, X. Song, J. Demmel, K. Keutzer, and C.-J. Hsieh, "Large batch optimization for deep learning: Training BERT in 76 minutes," in *International Conference on Learning Representations*, 2020.

[25] Y. Lin, S. Han, H. Mao, Y. Wang, and B. Dally, "Deep gradient compression: Reducing the communication bandwidth for distributed training," in *International Conference on Learning Representations*, 2018.

[26] L. Bottou, F. E. Curtis, and J. Nocedal, "Optimization methods for large-scale machine learning," *SIAM Review*, vol. 60, pp. 223–311, 2016.

[27] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang *et al.*, "Large scale distributed deep networks," in *Advances in neural information processing systems*, 2012, pp. 1223–1231.

[28] R. Mayer, C. Mayer, and L. Laich, "The tensorflow partitioning and scheduling problem: Its the critical path!" in *Proceedings of the 1st Workshop on Distributed Infrastructures for Deep Learning*, ser. DIDL 17. New York, NY, USA: Association for Computing Machinery, 2017, p. 16.

[29] A. Mirhoseini, H. Pham, Q. V. Le, B. Steiner, R. Larsen, Y. Zhou, N. Kumar, M. Norouzi, S. Bengio, and J. Dean, "Device placement optimization with reinforcement learning," in *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ser. ICML'17. JMLR.org, 2017, pp. 2430–2439.

[30] L. G. Valiant, "A bridging model for parallel computation," *Commun. ACM*, vol. 33, no. 8, p. 103111, Aug. 1990.

[31] M. Li, L. Zhou, Z. Yang, A. Q. Li, F. Xia, D. G. Andersen, and A. J. Smola, "Parameter server for distributed machine learning," in *In Big Learning NIPS Workshop*, 2013.

[32] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, "Scaling distributed machine learning with the parameter server," in *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'14. Berkeley, CA, USA: USENIX Association, 2014, pp. 583–598.

[33] Q. Ho, J. Cipar, H. Cui, J. K. Kim, S. Lee, P. B. Gibbons, G. A. Gibson, G. R. Ganger, and E. P. Xing, "More effective distributed ml via a stale synchronous parallel parameter server," in *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'13. USA: Curran Associates Inc., 2013, pp. 1223–1231.

[34] M. Li, D. G. Andersen, A. Smola, and K. Yu, "Communication efficient distributed machine learning with the parameter server," in *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'14. Cambridge, MA, USA: MIT Press, 2014, pp. 19–27.

[35] B. C. Ooi, K.-L. Tan, S. Wang, W. Wang, Q. Cai, G. Chen, J. Gao, Z. Luo, A. K. Tung, Y. Wang, Z. Xie, M. Zhang, and K. Zheng, "Singa: A distributed deep learning platform," in *Proceedings of the 23rd ACM International Conference on Multimedia*, ser. MM '15. New York, NY, USA: ACM, 2015, pp. 685–688.

[36] M. Zinkevich, M. Weimer, L. Li, and A. J. Smola, "Parallelized stochastic gradient descent," in *Advances in neural information processing systems*, 2010, pp. 2595–2603.

[37] Q. Ho, J. Cipar, H. Cui, S. Lee, J. K. Kim, P. B. Gibbons, G. A. Gibson, G. Ganger, and E. P. Xing, "More effective distributed ML via a stale synchronous parallel parameter server," in *Advances in neural information processing systems*, 2013, pp. 1223–1231.

[38] C. Zhang and C. Ré, "Dimmwitted: A study of main-memory statistical analytics," *Proc. VLDB Endow.*, vol. 7, no. 12, pp. 1283–1294, Aug. 2014.

[39] A. S. Bijral, A. D. Sarwate, and N. Srebro, "On data dependence in distributed stochastic optimization," 2016.

[40] J. Zhang, C. D. Sa, I. Mitliagkas, and C. Ré, "Parallel sgd: When does averaging help?" *ArXiv*, vol. abs/1606.07365, 2016.

[41] F. Haddadpour, M. M. Kamani, M. Mahdavi, and V. Cadambe, "Local sgd with periodic averaging: Tighter analysis and adaptive synchronization," in *Advances in Neural Information Processing Systems 32*, 2019, pp. 11 080–11 092.

[42] R. T. McDonald, K. B. Hall, and G. Mann, "Distributed training strategies for the structured perceptron," in *HLT-NAACL*, 2010.

[43] R. Mcdonald, M. Mohri, N. Silberman, D. Walker, and G. S. Mann, "Efficient large-scale distributed training of conditional maximum entropy models," in *Advances in Neural Information Processing Systems 22*, Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams, and A. Culotta, Eds., 2009, pp. 1231–1239.

[44] X. Zhang, J. Trmal, D. Povey, and S. Khudanpur, "Improving deep neural network acoustic models using generalized maxout networks," in *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2014, pp. 215–219.

[45] S. Zhang, A. Choromanska, and Y. LeCun, "Deep learning with elastic averaging sgd," in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'15. Cambridge, MA, USA: MIT Press, 2015, pp. 685–693.

[46] H. Yu, S. X. Yang, and S. Zhu, "Parallel restarted sgd with faster convergence and less communication: Demystifying why model averaging works for deep learning," in *AAAI*, 2018.

[47] A. A. Awan, K. Hamidouche, J. M. Hashmi, and D. K. Panda, "S-caffe: Co-designing mpi runtimes and caffe for scalable deep learning on modern GPU clusters," in *Acm Sigplan Notices*, vol. 52, no. 8. ACM, 2017, pp. 193–205.

[48] C.-H. Chu, X. Lu, A. A. Awan, H. Subramoni, J. Hashmi, B. Elton, and D. K. Panda, "Efficient and scalable multi-source streaming broadcast on GPU clusters for deep learning," in *2017 46th International Conference on Parallel Processing (ICPP)*. IEEE, 2017, pp. 161–170.

[49] S. Wang, D. Li, J. Geng, Y. Gu, and Y. Cheng, "Impact of network topology on the performance of dml: Theoretical analysis and practical factors," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 1729–1737.

[50] G. Lan, S. Lee, and Y. Zhou, "Communication-efficient algorithms for decentralized and stochastic optimization," *CoRR*, vol. abs/1701.03961, 2017.

[51] X. Lian, C. Zhang, H. Zhang, C.-J. Hsieh, W. Zhang, and J. Liu, "Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17. USA: Curran Associates Inc., 2017, pp. 5336–5346.

[52] H. Tang, X. Lian, M. Yan, C. Zhang, and J. Liu, "Decentralized training over decentralized data," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80, 10–15 Jul 2018, pp. 4848–4856.

[53] M. Assran, N. Loizou, N. Ballas, and M. Rabbat, "Stochastic gradient push for distributed deep learning," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. Long Beach, California, USA: PMLR, 09–15 Jun 2019, pp. 344–353.

[54] H. Tang, S. Gan, C. Zhang, T. Zhang, and J. Liu, "Communication compression for decentralized training," in *Proceedings of the 32Nd International Conference on Neural Information Processing Systems*, ser. NIPS'18. USA: Curran Associates Inc., 2018, pp. 7663–7673.

[55] Z. Tang, S. Shi, and X. Chu, "Communication-efficient decentralized learning with sparsification and adaptive peer selection," *arXiv preprint arXiv:2002.09692*, 2020.

[56] F. Seide, H. Fu, J. Droppo, G. Li, and D. Yu, "1-bit stochastic gradient descent and its application to data-parallel distributed training of speech dnns," in *Fifteenth Annual Conference of the International Speech Communication Association*, 2014.

[57] A. F. Aji and K. Heafield, "Sparse communication for distributed gradient descent," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Copenhagen, Denmark: Association for Computational Linguistics, Sep. 2017, pp. 440–445.

[58] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic, "QSGD: Communication-efficient SGD via gradient quantization and encoding," in *Advances in Neural Information Processing Systems*, 2017, pp. 1709–1720.

[59] C.-Y. Chen, J. Choi, D. Brand, A. Agrawal, W. Zhang, and K. Gopalakrishnan, "AdaComp: Adaptive residual gradient compression for data-parallel distributed training," in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[60] W. Wen, C. Xu, F. Yan, C. Wu, Y. Wang, Y. Chen, and H. Li, "Terngrad: Ternary gradients to reduce communication in distributed deep learning," in *Advances in neural information processing systems*, 2017, pp. 1509–1519.

[61] J. Wu, W. Huang, J. Huang, and T. Zhang, "Error compensated quantized SGD and its applications to large-scale distributed optimization," *International Conference on Machine Learning*, 2018.

[62] F. Haddadpour, M. M. Kamani, M. Mahdavi, and V. Cadambe, "Trading redundancy for communication: Speeding up distributed SGD for non-convex optimization," in *International Conference on Machine Learning*, 2019, pp. 2545–2554.

[63] H. Zhang, Z. Zheng, S. Xu, W. Dai, Q. Ho, X. Liang, Z. Hu, J. Wei, P. Xie, and E. P. Xing, "Poseidon: An efficient communication architecture for distributed deep learning on GPU clusters," in *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, 2017, pp. 181–193.

[64] S. Shi, X. Chu, and B. Li, "MG-WFBP: Efficient data communication for distributed synchronous SGD algorithms," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 172–180.

[65] S. H. Hashemi, S. A. Jyothi, and R. H. Campbell, "Tictac: Accelerating distributed deep learning with communication scheduling," 2018.

[66] A. Harlap, D. Narayanan, A. Phanishayee, V. Seshadri, N. Devanur, G. Ganger, and P. Gibbons, "Pipedream: Fast and efficient pipeline parallel DNN training," *arXiv preprint arXiv:1806.03377*, 2018.

[67] Y. Peng, Y. Zhu, Y. Chen, Y. Bao, B. Yi, C. Lan, C. Wu, and C. Guo, "A generic communication scheduler for distributed DNN training acceleration," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*. ACM, 2019, pp. 16–29.

[68] A. Jayarajan, J. Wei, G. Gibson, A. Fedorova, and G. Pekhimenko, "Priority-based parameter propagation for distributed dnn training," in *In Proceedings ofSystems and Machine Learning (SysML)*, 2018.

[69] J. Wang and G. Joshi, "Adaptive communication strategies to achieve the best error-runtime trade-off in local-update SGD," *Proc. of Workshop on Systems for ML and Open Source Software, collocated with NeurIPS 2018*, vol. abs/1810.08313, 2018.

[70] W. Lee, Y. Lee, J. S. Jeong, G. Yu, J. Y. Kim, H. J. Park, B. Jeon, W. Song, G. Kim, M. Weimer, B. Cho, and B. Chun, "Automating system configuration of distributed machine learning," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, July 2019, pp. 2057–2067.

[71] S. Shi, Q. Wang, X. Chu, and B. Li, "A DAG model of synchronous stochastic gradient descent in distributed deep learning," in *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 2018, pp. 425–432.

[72] X. Lian, Y. Huang, Y. Li, and J. Liu, "Asynchronous parallel stochastic gradient for nonconvex optimization," in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS15. Cambridge, MA, USA: MIT Press, 2015, p. 27372745.

[73] S. U. Stich, "Local SGD converges fast and communicates little," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

[74] A. Smola and S. Narayanamurthy, "An architecture for parallel topic models," *Proc. VLDB Endow.*, vol. 3, no. 1-2, pp. 703–710, Sep. 2010.

[75] F. N. I. P. H. Jin, Q. Yuan and K. Keutzer., "How to scale distributed deep learning?" in *ML Systems Workshop at NIPS*, 2016.

[76] D. Kempe, A. Dobra, and J. Gehrke, "Gossip-based computation of aggregate information," in *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.*, Oct 2003, pp. 482–491.

[77] Lin Xiao and S. Boyd, "Fast linear iterations for distributed averaging," in *42nd IEEE International Conference on Decision and Control (IEEE Cat. No.03CH37475)*, vol. 5, Dec 2003, pp. 4997–5002 Vol.5.

[78] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Randomized gossip algorithms," *IEEE/ACM Trans. Netw.*, vol. 14, no. SI, pp. 2508–2530, Jun. 2006.

[79] I. Colin, A. Bellet, J. Salmon, and S. Clémençon, "Gossip dual averaging for decentralized optimization of pairwise functions," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ser. ICML'16. JMLR.org, 2016, pp. 1388–1396.

[80] A. Nedic and A. Ozdaglar, "Distributed subgradient methods for multi-agent optimization," *IEEE Transactions on Automatic Control*, vol. 54, no. 1, pp. 48–61, Jan 2009.

[81] F. Iutzeler, P. Bianchi, P. Ciblat, and W. Hachem, "Asynchronous distributed optimization using a randomized alternating direction method of multipliers," in *52nd IEEE Conference on Decision and Control*, Dec 2013, pp. 3671–3676.

[82] K. Seaman, F. Bach, S. Bubeck, Y. T. Lee, and L. Massoulié, "Optimal algorithms for smooth and strongly convex distributed optimization in networks," in *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ser. ICML'17. JMLR.org, 2017, pp. 3027–3036.

[83] K. Scaman, F. Bach, S. Bubeck, Y. T. Lee, and L. Massoulié, "Optimal algorithms for non-smooth distributed optimization in networks," in *Proceedings of the 32Nd International Conference on Neural Information Processing Systems*, ser. NIPS'18. USA: Curran Associates Inc., 2018, pp. 2745–2754.

[84] C. A. Uribe, S. Lee, A. Gasnikov, and A. Nedic, "A dual approach for optimal algorithms in distributed optimization over networks," *CoRR*, vol. abs/1809.00710, 2018.

[85] M. Rabbat, "Multi-agent mirror descent for decentralized stochastic optimization," in *2015 IEEE 6th International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP)*, Dec 2015, pp. 517–520.

[86] V. Vanhoucke, A. Senior, and M. Z. Mao, "Improving the speed of neural networks on cpus," in *Deep Learning and Unsupervised Feature Learning Workshop, NIPS 2011*, 2011.

[87] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *Proceedings of the 32nd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, F. Bach and D. Blei, Eds., vol. 37. Lille, France: PMLR, 07–09 Jul 2015, pp. 1737–1746.

[88] M. Höhfeld and S. E. Fahlman, "Probabilistic rounding in neural network learning with limited precision," *Neurocomputing*, vol. 4, pp. 291–299, 1992.

[89] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients," *arXiv preprint arXiv:1606.06160*, 2016.

[90] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 6869–6898, Jan. 2017.

[91] S. Zheng, Z. Huang, and J. Kwok, "Communication-efficient distributed blockwise momentum SGD with error-feedback," in *NIPS*, 2019, pp. 11 446–11 456.

[92] S. Shi, Z. Tang, Q. Wang, K. Zhao, and X. Chu, "Layer-wise adaptive gradient sparsification for distributed deep learning with convergence guarantees," in *Proc. of The 24th European Conference on Artificial Intelligence*, 2020.

[93] S. Shi, Q. Wang, X. Chu, B. Li, Y. Qin, R. Liu, and X. Zhao, "Communication-efficient distributed deep learning with merged gradient sparsification on gpus," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, 2020.

[94] A. Dutta, E. H. Bergou, A. M. Abdelmoniem, C.-Y. Ho, A. N. Sahu, M. Canini, and P. Kalnis, "On the discrepancy between the theoretical analysis and practical implementations of compressed communication for distributed deep learning," in *AAAI*, 2020.

[95] A. Krizhevsky, "One weird trick for parallelizing convolutional neural networks," *ArXiv*, vol. abs/1404.5997, 2014.

[96] J. K. Bradley, A. Kyrola, D. Bickson, and C. Guestrin, "Parallel coordinate descent for l1-regularized loss minimization," in *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ser. ICML'11. USA: Omnipress, 2011, pp. 321–328.

[97] R. Krizanc and A. Saarimaki, "Bulk synchronous parallel: practical experience with a model for parallel computing," in *Proceedings of the 1996 Conference on Parallel Architectures and Compilation Technique*, Oct 1996, pp. 208–217.

[98] T. Cheatham, A. Fahmy, D. C. Stefanescu, and L. G. Valiant, "Bulk synchronous parallel computing-a paradigm for transportable software," in *Proceedings of the Twenty-Eighth Annual Hawaii International Conference on System Sciences*, vol. 2, Jan 1995, pp. 268–275 vol.2.

[99] Y. Zou, X. Jin, Y. Li, Z. Guo, E. Wang, and B. Xiao, "Mariana: Tencent deep learning platform and its applications," *PVLDB*, vol. 7, pp. 1772–1777, 2014.

[100] J. Bruck, Ching-Tien Ho, S. Kipnis, E. Upfal, and D. Weathersby, "Efficient algorithms for all-to-all communications in multiport message-passing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 8, no. 11, pp. 1143–1156, Nov 1997.

[101] S. S. Ram, A. Nedic, and V. V. Veeravalli, "Distributed stochastic subgradient projection algorithms for convex optimization," *Journal of Optimization Theory and Applications*, vol. 147, pp. 516–545, 2008.

[102] J. Chen, R. Monga, S. Bengio, and R. Jozefowicz, "Revisiting distributed synchronous sgd," in *International Conference on Learning Representations Workshop Track*, 2016.

[103] J. Dean and L. A. Barroso, "The tail at scale," *Commun. ACM*, vol. 56, no. 2, pp. 74–80, Feb. 2013.

[104] C. Chen, W. Wang, and B. Li, "Round-robin synchronization: Mitigating communication bottlenecks in parameter servers," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, April 2019, pp. 532–540.

[105] D. G. A. M. Li and A. Smola., "Distributed delayed proximal gradient methods," in *In NIPS Workshop on Optimization for Machine Learning*, Lake Tahoe, CA, 2013.

[106] D. Grishchenko, F. Iutzeler, J. Malick, and M.-R. Amini, "Asynchronous distributed learning with sparse communications and identification," *ArXiv*, vol. abs/1812.03871, 2018.

[107] J. F. C. Mota, J. M. F. Xavier, P. M. Q. Aguiar, and M. Pschel, "D-admm: A communication-efficient distributed algorithm for separable optimization," *IEEE Transactions on Signal Processing*, vol. 61, no. 10, pp. 2718–2723, May 2013.

[108] E. Wei and A. Ozdaglar, "Distributed alternating direction method of multipliers," in *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, Dec 2012, pp. 5445–5450.

[109] E. Wei and A. E. Ozdaglar, "On the o(1=k) convergence of asynchronous distributed alternating direction method of multipliers," in *2013 IEEE Global Conference on Signal and Information Processing*, Dec 2013, pp. 551–554.

[110] R. Zhang and J. T. Kwok, "Asynchronous distributed admm for consensus optimization," in *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ser. ICML'14. JMLR.org, 2014, pp. II–1701–II–1709.

[111] H. Yu, R. Jin, and S. Yang, "On the linear speedup analysis of communication efficient momentum SGD for distributed non-convex optimization," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. Long Beach, California, USA: PMLR, 09–15 Jun 2019, pp. 7184–7193.

[112] P. Jiang and G. Agrawal, "A linear speedup analysis of distributed deep learning with sparse and quantized communication," in *Proceedings of the 32Nd International Conference on Neural Information Processing Systems*, ser. NIPS'18. USA: Curran Associates Inc., 2018, pp. 2530–2541.

[113] H. Yu and R. Jin, "On the computation and communication complexity of parallel SGD with dynamic batch sizes for stochastic non-convex optimization," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. Long Beach, California, USA: PMLR, 09–15 Jun 2019, pp. 7174–7183.

[114] H. Lin, J. Mairal, and Z. Harchaoui, "A universal catalyst for first-order optimization," in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS'15. Cambridge, MA, USA: MIT Press, 2015, pp. 3384–3392.

[115] C. Paquette, H. Lin, D. Drusvyatskiy, J. Mairal, and Z. Harchaoui, "Catalyst for gradient-based nonconvex optimization," in *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, A. Storkey and F. Perez-Cruz, Eds., vol. 84. Playa Blanca, Lanzarote, Canary Islands: PMLR, 09–11 Apr 2018, pp. 613–622.

[116] K. Chen and Q. Huo, "Scalable training of deep learning machines by incremental block training with intra-block parallel optimization and blockwise model-update filtering," in *ICASSP-2016*, March 2016.

[117] A. Defazio and L. Bottou, "On the ineffectiveness of variance reduced optimization for deep learning," in *Advances in Neural Information Processing Systems 32*, 2019, pp. 1753–1763.

[118] E. Hoffer, I. Hubara, and D. Soudry, "Train longer, generalize better: Closing the generalization gap in large batch training of neural networks," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17. USA: Curran Associates Inc., 2017, pp. 1729–1739.

[119] 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 *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

[120] C. J. Shallue, J. Lee, J. M. Antognini, J. Sohl-Dickstein, R. Frostig, and G. E. Dahl, "Measuring the effects of data parallelism on neural network training," *CoRR*, vol. abs/1811.03600, 2018.

[121] T. Lin, S. U. Stich, and M. Jaggi, "Don't use large mini-batches, use local sgd," *ArXiv*, vol. abs/1808.07217, 2018.

[122] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein, "Distributed graphlab: A framework for machine learning

and data mining in the cloud," *Proc. VLDB Endow.*, vol. 5, no. 8, pp. 716–727, Apr. 2012.

[123] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016, pp. 265–283.

[124] A. Smola and S. Narayanamurthy, "An architecture for parallel topic models," in *Proc. VLDB Endow.*, vol. 3, no. 1-2, Sep. 2010, pp. 703–710.

[125] A. Ahmed, M. Aly, J. Gonzalez, S. Narayanamurthy, and A. J. Smola, "Scalable inference in latent variable models," in *Proceedings of the Fifth ACM International Conference on Web Search and Data Mining*, ser. WSDM '12. New York, NY, USA: ACM, 2012, pp. 123–132.

[126] S. Ahn, B. Shahbaba, and M. Welling, "Distributed stochastic gradient mcmc," in *Proceedings of the 31st International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, E. P. Xing and T. Jebara, Eds., vol. 32, no. 2, Bejing, China, 22–24 Jun 2014, pp. 1044–1052.

[127] H. Cui, J. Cipar, Q. Ho, J. K. Kim, S. Lee, A. Kumar, J. Wei, W. Dai, G. R. Ganger, P. B. Gibbons, G. A. Gibson, and E. P. Xing, "Exploiting bounded staleness to speed up big data analytics," in *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference*, ser. USENIX ATC'14. Berkeley, CA, USA: USENIX Association, 2014, pp. 37–48.

[128] H. Cui, A. Tumanov, J. Wei, L. Xu, W. Dai, J. Haber-Kucharsky, Q. Ho, G. R. Ganger, P. B. Gibbons, G. A. Gibson, and E. P. Xing, "Exploiting iterative-ness for parallel ml computations," in *Proceedings of the ACM Symposium on Cloud Computing*, ser. SOCC '14. New York, NY, USA: ACM, 2014, pp. 5:1–5:14.

[129] R. Power and J. Li, "Piccolo: Building fast, distributed programs with partitioned tables," in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 293–306.

[130] N. Sukhija, M. Tatineni, N. Brown, M. V. Moer, P. Rodriguez, and S. Callicott, "Topic modeling and visualization for big data in social sciences," in *2016 Intl IEEE Conferences on Ubiquitous Intelligence Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld)*, July 2016, pp. 1198–1205.

[131] H. Cui, H. Zhang, G. R. Ganger, P. B. Gibbons, and E. P. Xing, "Geeps: Scalable deep learning on distributed gpus with a gpu-specialized parameter server," in *Proceedings of the Eleventh European Conference on Computer Systems*, ser. EuroSys '16. New York, NY, USA: ACM, 2016, pp. 4:1–4:16.

[132] F. Seide, H. Fu, J. Droppo, G. Li, and D. Yu, "1-bit stochastic gradient descent and its application to data-parallel distributed training of speech DNNs," in *INTERSPEECH*, 2014.

[133] N. Strom, "Scalable distributed DNN training using commodity GPU cloud computing," in *INTERSPEECH*, 2015.

[134] D. Alistarh, J. Li, R. Tomioka, and M. Vojnovic, "Qsgd: Randomized quantization for communication-optimal stochastic gradient descent," *ArXiv*, vol. abs/1610.02132, 2016.

[135] J. Wu, W. Huang, J. Huang, and T. Zhang, "Error compensated quantized SGD and its applications to large-scale distributed optimization," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. Stockholmsmssan, Stockholm Sweden: PMLR, 10–15 Jul 2018, pp. 5325–5333.

[136] W. Wen, C. Xu, F. Yan, C. Wu, Y. Wang, Y. Chen, and H. Li, "Terngrad: Ternary gradients to reduce communication in distributed deep learning," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17. USA: Curran Associates Inc., 2017, pp. 1508–1518.

[137] J. Bernstein, Y. Wang, K. Azizzadenesheli, and A. Anandkumar, "SIGNSGD: compressed optimisation for non-convex problems," in *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, 2018, pp. 559–568.

[138] S. P. Karimireddy, Q. Rebjock, S. Stich, and M. Jaggi, "Error feedback fixes SignSGD and other gradient compression schemes," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. Long Beach, California, USA: PMLR, 09–15 Jun 2019, pp. 3252–3261.

[139] J. Langford, L. Li, and T. Zhang, "Sparse online learning via truncated gradient," *J. Mach. Learn. Res.*, vol. 10, pp. 777–801, Jun. 2009.

[140] J. Konen, H. B. McMahan, F. X. Yu, P. Richtarik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," in *NIPS Workshop on Private Multi-Party Machine Learning*, 2016.

[141] J. Wangni, J. Wang, J. Liu, and T. Zhang, "Gradient sparsification for communication-efficient distributed optimization," in *Proceedings of the 32Nd International Conference on Neural Information Processing Systems*, ser. NIPS'18. USA: Curran Associates Inc., 2018, pp. 1306–1316.

[142] N. Dryden, S. A. Jacobs, T. Moon, and B. Van Essen, "Communication quantization for data-parallel training of deep neural networks," in *Proceedings of the Workshop on Machine Learning in High Performance Computing Environments*, ser. MLHPC '16. Piscataway, NJ, USA: IEEE Press, 2016, pp. 1–8.

[143] H. Zhang, Z. Zheng, S. Xu, W. Dai, Q. Ho, X. Liang, Z. Hu, J. Wei, P. Xie, and E. P. Xing, "Poseidon: An efficient communication architecture for distributed deep learning on GPU clusters," in *Proceedings of the 2017 USENIX Conference on Usenix Annual Technical Conference*, ser. USENIX ATC '17. Berkeley, CA, USA: USENIX Association, 2017, pp. 181–193.

[144] L. Wang, W. Wang, and B. Li, "CMFL: mitigating communication overhead for federated learning," in *39th IEEE International Conference on Distributed Computing Systems, ICDCS 2019, Dallas, TX, USA, July 7-10, 2019*, 2019, pp. 954–964.

[145] R. Rabenseifner, "Optimization of collective reduction operations," in *Computational Science - ICCS 2004*, M. Bubak, G. D. van Albada, P. M. A. Sloot, and J. Dongarra, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 1–9.

[146] R. Thakur, R. Rabenseifner, and W. Gropp, "Optimization of collective communication operations in mpich," *Int. J. High Perform. Comput. Appl.*, vol. 19, no. 1, pp. 49–66, Feb. 2005.

[147] T. Hoefler, W. Gropp, R. Thakur, and J. L. Träff, "Toward performance models of mpi implementations for understanding application scaling issues," in *Proceedings of the 17th European MPI Users' Group Meeting Conference on Recent Advances in the Message Passing Interface*, ser. EuroMPI'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 21–30.

[148] P. Sanders, J. Speck, and J. L. Träff, "Two-tree algorithms for full bandwidth broadcast, reduction and scan," *Parallel Computing*, vol. 35, no. 12, pp. 581–594, 2009.

[149] S. Sarvotham, R. Riedi, and R. Baraniuk, "Connection-level analysis and modeling of network traffic," in *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*. ACM, 2001, pp. 99–103.

[150] Y. Ueno and R. Yokota, "Exhaustive study of hierarchical allreduce patterns for large messages between gpus," in *Proceedings of the 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, 2019.

[151] H. Mikami, H. Suganuma, Y. Tanaka, Y. Kageyama *et al.*, "Massively distributed SGD: ImageNet/ResNet-50 training in a flash," *arXiv preprint arXiv:1811.05233*, 2018.

[152] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, 2017, pp. 1–12.

[153] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, "BCube: a high performance, server-centric network architecture for modular data centers," in *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, 2009, pp. 63–74.

[154] G. Wang, S. Venkataraman, A. Phanishayee, J. Thelin, N. Devanur, and I. Stoica, "Blink: Fast and generic collectives for distributed ML," in *Proceedings of the 3rd MLSys Conference*, 2020.

[155] L. Luo, P. West, J. Nelson, A. Krishnamurthy, and L. Ceze, "PLink: Efficient cloud-based training with topology-aware dynamic hierarchical aggregation," in *Proceedings of the 3rd MLSys Conference*, 2020.

[156] N. S. Aybat, Z. Wang, T. Lin, and S. Ma, "Distributed linearized alternating direction method of multipliers for composite convex consensus

optimization," *IEEE Transactions on Automatic Control*, vol. 63, no. 1, pp. 5–20, Jan 2018.

[157] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Gossip algorithms: design, analysis and applications," in *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.*, vol. 3, March 2005, pp. 1653–1664 vol. 3.

[158] R. Carli, F. Fagnani, P. Frasca, and S. Zampieri, "Gossip consensus algorithms via quantized communication," *Automatica*, vol. 46, no. 1, pp. 70–80, Jan. 2010.

[159] F. Fagnani and S. Zampieri, "Randomized consensus algorithms over large scale networks," *IEEE Journal on Selected Areas in Communications*, vol. 26, no. 4, pp. 634–649, May 2008.

[160] R. Olfati-Saber, J. A. Fax, and R. M. Murray, "Consensus and cooperation in networked multi-agent systems," *Proceedings of the IEEE*, vol. 95, no. 1, pp. 215–233, Jan 2007.

[161] L. Schenato and G. Gamba, "A distributed consensus protocol for clock synchronization in wireless sensor network," in *2007 46th IEEE Conference on Decision and Control*, Dec 2007, pp. 2289–2294.

[162] A. Nedi and A. Olshevsky, "Stochastic gradient-push for strongly convex functions on time-varying directed graphs," *IEEE Transactions on Automatic Control*, vol. 61, no. 12, pp. 3936–3947, Dec 2016.

[163] A. Nedi, A. Olshevsky, and M. G. Rabbat, "Network topology and communication-computation tradeoffs in decentralized optimization," *Proceedings of the IEEE*, vol. 106, no. 5, pp. 953–976, May 2018.

[164] A. Koloskova, S. Stich, and M. Jaggi, "Decentralized stochastic optimization and gossip algorithms with compressed communication," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. Long Beach, California, USA: PMLR, 09–15 Jun 2019, pp. 3478–3487.

[165] L. He, A. Bian, and M. Jaggi, "Cola: Decentralized linear learning," in *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., 2018, pp. 4536–4546.

[166] A. T. Suresh, F. X. Yu, S. Kumar, and H. B. McMahan, "Distributed mean estimation with limited communication," in *Proceedings of the 34th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, D. Precup and Y. W. Teh, Eds., vol. 70. International Convention Centre, Sydney, Australia: PMLR, 06–11 Aug 2017, pp. 3329–3337.

[167] J. Konen and P. Richtrik, "Randomized distributed mean estimation: Accuracy vs. communication," *Frontiers in Applied Mathematics and Statistics*, vol. 4, p. 62, 2018.

[168] Y. Pu, M. N. Zeilinger, and C. N. Jones, "Quantization design for distributed optimization," *IEEE Transactions on Automatic Control*, vol. 62, no. 5, pp. 2107–2120, May 2017.

[169] N. McGlohon and S. Patterson, "Distributed semi-stochastic optimization with quantization refinement," in *2016 American Control Conference (ACC)*, July 2016, pp. 7159–7164.

[170] S. Horvath, C.-Y. Ho, L. Horvath, A. N. Sahu, M. Canini, and P. Richtárik, "Natural compression for distributed deep learning," *ArXiv*, vol. abs/1905.10988, 2019.

[171] W. M. Goodall, "Television by pulse code modulation," *The Bell System Technical Journal*, vol. 30, no. 1, pp. 33–49, Jan 1951.

[172] L. Roberts, "Picture coding using pseudo-random noise," *IRE Transactions on Information Theory*, vol. 8, no. 2, pp. 145–154, February 1962.

[173] J. Bernstein, J. Zhao, K. Azizzadenesheli, and A. Anandkumar, "signsgd with majority vote is communication efficient and byzantine fault tolerant," *ArXiv*, vol. abs/1810.05291, 2018.

[174] H. Wang, S. Sievert, Z. Charles, S. Liu, S. Wright, and D. Papailiopoulos, "Atomo: Communication-efficient learning via atomic sparsification," in *Proceedings of the 32Nd International Conference on Neural Information Processing Systems*, ser. NIPS'18. USA: Curran Associates Inc., 2018, pp. 9872–9883.

[175] K. Mishchenko, E. A. Gorbunov, M. Takác, and P. Richtárik, "Distributed learning with compressed gradient differences," *ArXiv*, vol. abs/1901.09269, 2019.

[176] J. Sun, T. Chen, G. Giannakis, and Z. Yang, "Communication-efficient distributed learning via lazily aggregated quantized gradients," in *Advances in Neural Information Processing Systems 32*, 2019, pp. 3365–3375.

[177] Y. Yu, J. Wu, and L. Huang, "Double quantization for communication-efficient distributed optimization," in *Advances in Neural Information Processing Systems 32*, 2019, pp. 4440–4451.

[178] X. Sun, X. Ren, S. Ma, and H. Wang, "meProp: Sparsified back propagation for accelerated deep learning with reduced overfitting," in *Proceedings of the 34th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, D. Precup and Y. W. Teh, Eds., vol. 70. International Convention Centre, Sydney, Australia: PMLR, 06–11 Aug 2017, pp. 3299–3308.

[179] H. Zhang, J. Li, K. Kara, D. Alistarh, J. Liu, and C. Zhang, "ZipML: Training linear models with end-to-end low precision, and a little bit of deep learning," in *Proceedings of the 34th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, D. Precup and Y. W. Teh, Eds., vol. 70. International Convention Centre, Sydney, Australia: PMLR, 06–11 Aug 2017, pp. 4035–4043.

[180] C. Dünner, T. P. Parnell, and M. Jaggi, "Efficient use of limited-memory accelerators for linear learning on heterogeneous systems," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, 2017, pp. 4258–4267.

[181] D. Basu, D. Data, C. Karakus, and S. Diggavi, "Qsparse-local-SGD: Distributed SGD with quantization, sparsification and local computations," in *Advances in Neural Information Processing Systems 32*, 2019, pp. 14 668–14 679.

[182] N. Strom, "A tonotopic artificial neural network architecture for phoneme probability estimation," in *1997 IEEE Workshop on Automatic Speech Recognition and Understanding Proceedings*, Dec 1997, pp. 156–163.

[183] N. Strm, "Sparse connection and pruning in large dynamic artificial neural networks," in *Proc. EUROSPEECH' 97*, 1997, pp. 2807–2810.

[184] S. U. Stich, J.-B. Cordonnier, and M. Jaggi, "Sparsified SGD with memory," in *Proceedings of the 32Nd International Conference on Neural Information Processing Systems*, ser. NIPS'18. USA: Curran Associates Inc., 2018, pp. 4452–4463.

[185] D. Alistarh, T. Hoefler, M. Johansson, S. Khirirat, N. Konstantinov, and C. Renggli, "The convergence of sparsified gradient methods," in *Proceedings of the 32Nd International Conference on Neural Information Processing Systems*, ser. NIPS'18. USA: Curran Associates Inc., 2018, pp. 5977–5987.

[186] J. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *ArXiv*, vol. abs/1607.06450, 2016.

[187] C. Chen, J. Choi, D. Brand, A. Agrawal, W. Zhang, and K. Gopalakrishnan, "Adacomp : Adaptive residual gradient compression for data-parallel distributed training," in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18)*, 2018, pp. 2827–2835.

[188] F. Sattler, S. Wiedemann, K.-R. Müller, and W. Samek, "Sparse binary compression: Towards distributed deep learning with minimal communication," *2019 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, 2018.

[189] F. Sattler, S. Wiedemann, K. Müller, and W. Samek, "Robust and communication-efficient federated learning from non-iid data," *CoRR*, vol. abs/1903.02891, 2019.

[190] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, 2017, pp. 1273–1282.

[191] S. Shi, Q. Wang, K. Zhao, Z. Tang, Y. Wang, X. Huang, and X. Chu, "A distributed synchronous SGD algorithm with global top-k sparsification for low bandwidth networks," in *39th IEEE International Conference on Distributed Computing Systems, ICDCS 2019*, 2019, pp. 2238–2247.

[192] S. Shi, K. Zhao, Q. Wang, Z. Tang, and X. Chu, "A convergence analysis of distributed SGD with communication-efficient gradient sparsification," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, 7 2019, pp. 3411–3417.

[193] S. Shi, X. Chu, K. C. Cheung, and S. See, "Understanding top-k sparsification in distributed deep learning," *arXiv preprint arXiv:1911.08772*, 2019.

[194] P. Han, S. Wang, and K. K. Leung, "Adaptive gradient sparsification for efficient federated learning: An online learning approach," *arXiv preprint arXiv:2001.04756*, 2020.

[195] C.-J. Hsieh, H.-F. Yu, and I. S. Dhillon, "Passcode: Parallel asynchronous stochastic dual co-ordinate descent," in *Proceedings of the*

32Nd International Conference on International Conference on Machine Learning - Volume 37, ser. ICML'15. JMLR.org, 2015, pp. 2370–2379.

[196] T. Sun, R. Hannah, and W. Yin, "Asynchronous coordinate descent under more realistic assumption," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17. USA: Curran Associates Inc., 2017, pp. 6183–6191.

[197] Y. Xu and W. Yin, "A block coordinate descent method for regularized multiconvex optimization with applications to nonnegative tensor factorization and completion," *SIAM J. Imaging Sciences*, vol. 6, pp. 1758–1789, 2013.

[198] Z. Zhang and M. Brand, "Convergent block coordinate descent for training tikhonov regularized deep neural networks," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17. USA: Curran Associates Inc., 2017, pp. 1719–1728.

[199] D. Mahajan, S. S. Keerthi, and S. Sundararajan, "A distributed block coordinate descent method for training l1regularized linear classifiers," *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 3167–3201, Jan. 2017.

[200] T. T. Lau, J. Zeng, B. Wu, and Y. Yao, "A proximal block coordinate descent algorithm for deep neural network training," in *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Workshop Track Proceedings*, 2018.

[201] Z. Qu, P. Richtárik, and T. Zhang, "Quartz: Randomized dual coordinate ascent with arbitrary sampling," in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'15. Cambridge, MA, USA: MIT Press, 2015, pp. 865–873.

[202] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. The MIT Press, 2016.

[203] J. Zeng, T. T.-K. Lau, S. Lin, and Y. Yao, "Global convergence of block coordinate descent in deep learning," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. Long Beach, California, USA: PMLR, 09–15 Jun 2019, pp. 7313–7323.

[204] K. Mishchenko, F. Hanzely, and P. Richtárik, "99% of parallel optimization is inevitably a waste of time," *ArXiv*, vol. abs/1901.09437, 2019.

[205] J. Ren, X. Li, and J. Haupt, "Communication-efficient distributed optimization for sparse learning via two-way truncation," in *2017 IEEE 7th International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP)*, Dec 2017, pp. 1–5.

[206] Y. Tsuzuku, H. Imachi, and T. Akiba, "Variance-based gradient compression for efficient distributed deep learning," in *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Workshop Track Proceedings*, 2018.

[207] N. Ivkin, D. Rothchild, E. Ullah, V. braverman, I. Stoica, and R. Arora, "Communication-efficient distributed sgd with sketching," in *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 13 144–13 154.

[208] M. Charikar, K. Chen, and M. Farach-Colton, "Finding frequent items in data streams," in *Proceedings of the 29th International Colloquium on Automata, Languages and Programming*, ser. ICALP '02. Berlin, Heidelberg: Springer-Verlag, 2002, pp. 693–703.

[209] Q. Wang, S. Shi, C. Wang, and X. Chu, "Communication contention aware scheduling of multiple deep learning training jobs," *arXiv preprint arXiv:2002.10105*, 2020.

[210] C. D. Sa, C. Zhang, K. Olukotun, and C. Ré, "Taming the wild: A unified analysis of hogwild-style algorithms," *Advances in neural information processing systems*, vol. 28, pp. 2656–2664, 2015.

[211] D. Alistarh, C. D. Sa, and N. Konstantinov, "The convergence of stochastic gradient descent in asynchronous shared memory," in *PODC '18*, 2018.

[212] X. Zhang, J. Liu, and Z. Zhu, "Taming convergence for asynchronous stochastic gradient descent with unbounded delay in non-convex learning," *arXiv preprint arXiv:1805.09470*, 2018.

[213] A. Dieuleveut and K. K. Patel, "Communication trade-offs for local-sgd with large step size," in *Advances in Neural Information Processing Systems 32*, 2019, pp. 13 579–13 590.

[214] A. Koloskova, T. Lin, S. U. Stich, and M. Jaggi, "Decentralized deep learning with arbitrary communication compression," *ArXiv*, vol. abs/1907.09356, 2019.

[215] J. Daily, A. Vishnu, C. Siegel, T. Warfel, and V. Amatya, "Gossipgrad: Scalable deep learning using gossip communication based asynchronous gradient descent," *CoRR*, vol. abs/1803.05880, 2018.

[216] X. Lian, W. Zhang, C. Zhang, and J. Liu, "Asynchronous decentralized parallel stochastic gradient descent," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. Stockholmsmssan, Stockholm Sweden: PMLR, 10–15 Jul 2018, pp. 3043–3052.

[217] H. Lim, D. G. Andersen, and M. Kaminsky, "3lc: Lightweight and effective traffic compression for distributed machine learning," *ArXiv*, vol. abs/1802.07389, 2019.

[218] S.-Y. Zhao, Y. Xie, H. Gao, and W.-J. Li, "Global momentum compression for sparse communication in distributed sgd," *ArXiv*, vol. abs/1905.12948, 2019.

[219] Q. Meng, W. Chen, J. Yu, T. Wang, Z.-M. Ma, and T.-Y. Liu, "Asynchronous accelerated stochastic gradient descent," in *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, ser. IJCAI'16, 2016, pp. 1853–1859.

[220] A. Reisizadeh, H. Taheri, A. Mokhtari, H. Hassani, and R. Pedarsani, "Robust and communication-efficient collaborative learning," in *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8388–8399.

[221] J. Wang and G. Joshi, "Cooperative SGD: A unified framework for the design and analysis of communication-efficient SGD algorithms," *CoRR*, vol. abs/1808.07576, 2018.

[222] N. Qian, "On the momentum term in gradient descent learning algorithms," *Neural Netw.*, vol. 12, no. 1, pp. 145–151, Jan. 1999.

[223] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, March 1994.

[224] P. Goyal, P. Dollár, R. B. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, large minibatch sgd: Training imagenet in 1 hour," *ArXiv*, vol. abs/1706.02677, 2017.

[225] D. Peteiro-Barral and B. Guijarro-Berdiñas., "A survey of methods for distributed machine learning," *Progress in Artificial Intelligence*, vol. 2, pp. 1–11, 2013.

[226] M. Gheisari, G. Wang, and M. Z. A. Bhuiyan, "A survey on deep learning in big data," in *2017 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC)*, vol. 2, July 2017, pp. 173–180.

[227] E. P. Xing, Q. Ho, P. Xie, and D. Wei, "Strategies and principles of distributed machine learning on big data," *Engineering*, vol. 2, no. 2, pp. 179 – 195, 2016.

[228] T. Ben-Nun and T. Hoefler, "Demystifying parallel and distributed deep learning: An in-depth concurrency analysis," *ACM Comput. Surv.*, vol. 52, no. 4, Aug. 2019.

[229] Y. Guo, "A survey on methods and theories of quantized neural networks," *CoRR*, vol. abs/1808.04752, 2018.

[230] Z. Zhang, L. Yin, Y. Peng, and D. Li, "A quick survey on large scale distributed deep learning systems," in *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*, Dec 2018, pp. 1052–1056.