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 [11], 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][9]. 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 [11] 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:

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

where the random variable $\xi_t$ has a probability distribution $D$, referring to a data sample from a dataset, $x$ represents all the parameters of the model, $N$ is the number of parameters, and $f_s : \mathbb{R}^N \rightarrow \mathbb{R}$ 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 and it variants, are commonly used in DL. The update rule of SGD has the following form.

$$G_t(x_t) = \nabla F_t(x_t; \xi_t)$$  (2)

$$x_{t+1} = x_t - \gamma G_t(x_t),$$  (3)

where $x_t \in \mathbb{R}^N$ is the $N$-dimensional model parameter at iteration $t$, $\xi_t$ is a randomly sampled mini-batch of data, and $\gamma$ is the learning rate (or step size). SGD is an iterative algorithm and has been proved that it can solve problem under...
the assumptions that $f_i(x)$ 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., $\xi_i$). 2) It performs the feed-forward computations to calculate the loss value (i.e., $F_i(x_i; \xi_i)$) of the objective function. 3) It performs the backward propagation to calculate the gradients (i.e., $\nabla F_i(x_i; \xi_i)$) 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:

$$f^* := \min_{x \in \mathbb{R}^d} \left[ f(x) := \frac{1}{n} \sum_{i=1}^{n} \mathbb{E}_{\xi_i \sim D_i} F_i(x; \xi_i) \right]$$

(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 ($\nabla F_i(x_i; \xi_{i,t})$) 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:

$$G_{i,t}(x_t) = \nabla F_{i,t}(x_t; \xi_{i,t})$$

$$x_{t+1} = x_t - \gamma \frac{1}{n} \sum_{i=1}^{n} G_{i,t}(x_t),$$

(5)

(6)

where $G_{i,t}(x_t)$ represents the gradient of $F_{i,t}(x_t)$ 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) [35] 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
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.

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.

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.
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 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 with the 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 \cite{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

$$x_{i,t+1} = x_0 - \gamma \sum_{k=1}^{t} \sum_{j=1}^{n} G_{j,k}(x_{j,k}) - \gamma \sum_{(j,k) \in S_{i,t+1}} G_{j,k}(x_{j,k}),$$

where $S_{i,t+1}$ 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 \cite{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

$$x_{i,t+1} = x_t - \gamma \sum_{k=1}^{t} \sum_{j=1}^{n} G_{i,t-k,i}(x_{i,t-k,i}),$$

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 \cite{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

$$x_{i,t+1} = \begin{cases} x_{i,t} - \gamma G_{i,t}(x_{i,t}), & \text{if } t + 1 \notin T_T \\ x_{i,t} - \frac{1}{n} \sum_{i=1}^{n} G_{i,t}(x_{i,t}), & \text{if } t + 1 \in T_T \end{cases},$$

where $T_T$ 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.

<table>
<thead>
<tr>
<th>TABLE I</th>
<th>TAXONOMY OF DISTRIBUTED SGD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension</td>
<td>Method</td>
</tr>
<tr>
<td>Communication Synchronization</td>
<td>Synchronous, Stale-Synchronous, Asynchronous, Local SGD</td>
</tr>
<tr>
<td>System Architectures</td>
<td>Parameter Server, All-Reduce, Gossip</td>
</tr>
<tr>
<td>Compression Techniques</td>
<td>Quantization, Coding, Sparsification</td>
</tr>
<tr>
<td>Parallelism of Communication and Computing</td>
<td>Pipelining, Scheduling</td>
</tr>
</tbody>
</table>
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][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-
searchers exploit it to improve data parallel distributed SGD \cite{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 \cite{86,87,88,89,90}.

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 \cite{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 \cite{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 I.

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 [25][96][27][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 [29][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 $n_c$ 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 $n_c$ 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 staggered 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**

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Synchronization</th>
<th>Model consistency</th>
<th>Communication Frequency</th>
<th>Communication Congestion</th>
<th>Convergence</th>
</tr>
</thead>
<tbody>
<tr>
<td>PS</td>
<td>BSP</td>
<td>high</td>
<td>high</td>
<td>high</td>
<td>easy</td>
</tr>
<tr>
<td></td>
<td>SSP</td>
<td>normal</td>
<td>high</td>
<td>normal</td>
<td>normal</td>
</tr>
<tr>
<td></td>
<td>ASP</td>
<td>low</td>
<td>high</td>
<td>low</td>
<td>difficult</td>
</tr>
<tr>
<td></td>
<td>Local SGD</td>
<td>normal</td>
<td>low</td>
<td>high</td>
<td>difficult</td>
</tr>
<tr>
<td>All-Reduce</td>
<td>BSP</td>
<td>high</td>
<td>high</td>
<td>low</td>
<td>easy</td>
</tr>
<tr>
<td></td>
<td>SSP</td>
<td>low</td>
<td>low</td>
<td>normal</td>
<td>normal</td>
</tr>
<tr>
<td></td>
<td>ASP</td>
<td>low</td>
<td>low</td>
<td>low</td>
<td>difficult</td>
</tr>
<tr>
<td></td>
<td>Local SGD</td>
<td>normal</td>
<td>low</td>
<td>low</td>
<td>difficult</td>
</tr>
<tr>
<td>Gossip</td>
<td>BSP</td>
<td>low</td>
<td>high</td>
<td>low</td>
<td>normal</td>
</tr>
<tr>
<td></td>
<td>SSP</td>
<td>low</td>
<td>low</td>
<td>normal</td>
<td>normal</td>
</tr>
<tr>
<td></td>
<td>ASP</td>
<td>low</td>
<td>high</td>
<td>low</td>
<td>difficult</td>
</tr>
<tr>
<td></td>
<td>Local SGD</td>
<td>low</td>
<td>low</td>
<td>low</td>
<td>difficult</td>
</tr>
</tbody>
</table>

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, 51, 52, 53, 41, 44, 55, 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. [115] proposed a Catalyst-like [114, 113] 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.

**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 have integrated the double and earlier versions of NCCL. 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) have integrated the double binary trees 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 were also proposed, which can reduce the latency cost several times (related to the number of hierarchies). 2D-Torus All-Reduce can also massively reduce the communication latency using a 2D-Torus topology network.

Wang et al. proposed BML for the BCube topology to achieve efficient communication. Some topology-aware algorithms (e.g., Blink and Plink) 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. For distributed SGD, Lian et al. 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. 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. 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, Assran et al. combined Stochastic Gradient Push (SGP) with PUSHSUM. 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.

V. Quantization methods

The communication of gradients in synchronous data-parallel SGD methods actually is a distributed mean estimation problem. Suresh et al. and Jakub et al. analyzed the communication-efficient algorithms for distributed mean estimation. They used the Mean Squared Error (MSE) to measure how accurate the quantization methods are.
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. \[168\] 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 \( G_{i,t}(x_t) \), and keeps the quantization element \( \delta_{i,t}(x_t) \) at the same time. The process of 1-bit SGD can be described using the following equations:

\[
G_{i,t}^{\text{quant}}(x_t) = \text{Quant}(G_{i,t}(x_t) + \delta_{i,t}(x_t))
\]

\[
\delta_{i,t}(x_t) = G_{i,t}(x_t) - \text{Quant}^{-1}(G_{i,t}^{\text{quant}}(x_t))
\]

\[
x_{t+1} = x_t - \frac{1}{N} \sum_{i=1}^{n} G_{i,t}^{\text{quant}}(x_t)
\]

where \( \text{Quant}() \) denotes the quantization function encoding the gradient, and \( \text{Quant}() \) the unquantizer decoding the gradient. Comparing to Eq. 5, the exchanging messages turn into \( G_{i,t}^{\text{quant}}(x_t) \), 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 \[168\] 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 \[19\] 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 \( v \in \mathbb{R}^N \), every \( j \)-th element \( \text{Quant}_s(v_j) \) of quantization gradient \( \text{Quant}_s(v) \) is quantized by \( \text{Quant}_s(\cdot) \), corresponding to the element \( v_j \) of the original gradient \( v \). The stochastic quantization function is defined as

\[
\text{Quant}_s(v_j) = \|v\|_2 \cdot \text{sgn}(v_j) \cdot \psi_j(v, s)
\]

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

\[
\psi_j(v, s) = \begin{cases} 
1/s & \text{with probability } 1 - p(v_j), s = 1/s \text{;} \\
(l + 1)/s & \text{otherwise.}
\end{cases}
\]

in which \( l \) is an integer such that \( |v_i| / \|v\|_2 \in [l/s, (l + 1)/s] \) 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 \( v = 0 \), the algorithm sets \( \text{Quant}_s(v, s) = 0 \). The expectation \( \psi_j(v, s) \) satisfies \( E[\psi_j(v, s)] = |v_i| / \|v\|_2 \). Then it is obvious that QSGD can assure that the quantized gradient is the unbiased estimation of the original vector, i.e., \( E[\text{Quant}_s(v)] = v \), 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 \( G(x) \) is quantized to a ternary level \{-1, 0, 1\} to reduce the communication data size. The gradient \( G(x) \) is ternarized as

\[
\tilde{Q}_t(G(x_i)) = \text{ternarize}(G(x_i)) = s_t \cdot \text{sgn}(G(x_i)) \odot b_t
\]

where \( s_t := \max(\text{abs}(G(x_i))) \), and \( \odot \) represents the Hadamard product. The \( \text{sgn}(\cdot) \) is the same as \( \text{sgn}(\cdot) \) in QSGD. Each element of \( b_t \) follows the distribution

\[
P(b_{t,j} = 1 | G_t(x_i)) = |G_{t,j}(x_i)/s_t|
\]

\[
P(b_{t,j} = 0 | G_t(x_i)) = 1 - |G_{t,j}(x_i)/s_t|
\]

where \( b_{t,j} \) and \( G_{t,j}(x_i) \) is the \( j \)-th element of \( b_t \) and \( G_t(x_i) \) 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 \( G_t(x_i) \).

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 they proposed Natural Dithering which is similar to QSGD. It modifies the $\ell$ in equation (14) into a binary geometric partition as: $l_1 = 1, l_2 = 2, l_3 = 4, \ldots, l_s = 2^{s-1}$.

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 $H$ into a sparse matrix $\hat{H}$. 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 $H$. For the later, workers only need to send the values of the non-zero entries of $H$, 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 $H$. Different from Random Mask, the sparse communication matrix $\hat{H}$ will be scaled to let $E(\hat{H}) = H$. 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 $p$, 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:

$$
\text{original vector } g = [g_1, g_2, \cdots, g_N],
$$
$$
\text{probability vector } p = [p_1, p_2, \cdots, p_N],
$$
$$
\text{selection vector } Z = [Z_1, Z_2, \cdots, Z_N],
$$
$$
\text{sparsified vector } Q_{spar}(g) = \left[ Z_1 \frac{g_1}{p_1}, Z_2 \frac{g_2}{p_2}, \cdots, Z_N \frac{g_N}{p_N} \right].
$$
Then there is
\[
\mathbb{E}[Q_{spar}(g)_i] = p_i \times \frac{g_i}{p_i} + (1 - p_i) \times g_i = 0 \quad (19)
\]
which indicates that the sparsified vector \(Q_{spar}(g)\) is an unbiased estimator of the original vector \(g\), similar to [134].

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 \(\pi\) 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-clock 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 \(\mu^+\) and negative mean \(\mu^-\) respectively. If the \(\mu^+\) is bigger than \(\mu^-\), SBC discards all negative elements and sets all positive elements to \(\mu^+\), 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 \(g\)Top-\(k\). After aggregating all gradients, \(g\)Top-\(k\) 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 \cite{198,198}. In distributed deep learning scenarios, BCD can be easily implemented in a distributed and parallel manner \cite{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. \cite{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 \cite{203}, Zeng et al. provided a general methodology for provable convergence guarantees for using BCD in deep learning.

Mishchenko et al. \cite{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 \cite{203,196}.

In \cite{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. \cite{205} developed a novel method to reduce the communication overhead. When training a batch of samples, some gradient elements are ambiguous \cite{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. \cite{207} exploited a technique that is widely adopted in distributed systems, Count Sketch \cite{208}. It compresses a gradient vector \( G \) into a sketch \( S(G) \) of size \( O(1/e \log n) \), 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) \cite{21} for pipelining or scheduling such that the iteration time is minimal.

In 2017, wait-free backward propagation (WFBP) \cite{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 \cite{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. \cite{68,65,69,67} scheduled the communication tasks and computation tasks by changing the order of execution. Peng et al. \cite{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. \cite{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:

\[
\| \nabla f_i(x) - \nabla f_i(y) \| \leq L \| x - y \| , \forall x, y \in \mathbb{R}^n. \tag{20}
\]

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

\[
g_i(x) := \mathbb{E}_{\xi \sim D_i} \nabla F_i(x; \xi) = \nabla f_i(x), \forall x \in \mathbb{R}^n, \tag{21}
\]

in which the \( D_i \) is the data distribution on \( i \)-th worker.

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

\[
\mathbb{E}_{\xi \sim D_i} \| \nabla F_i(x; \xi) - \nabla f_i(x) \|^2 \leq \sigma^2 , \forall i, \forall x \in \mathbb{R}^n. \tag{22}
\]
4) **Bounded stochastic gradient:** The second moment of the stochastic gradients is bounded:

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

5) \( \mu \)-**Strongly convex function:**

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

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 doubly stochastic matrix that satisfies \( W = W^T \) and \( W1 = W. \) 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** [134]: For a parameter \( 0 < d < n \), a \( k \)-contraction operator is a (possibly randomized) operator \( C(\cdot): \mathbb{R}^n \rightarrow \mathbb{R}^n \) that satisfies the contraction property

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

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

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

for any \( x \) 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

\[ \frac{1}{T} \mathbb{E} \left[ \| \nabla f(x) \|^2 \right] \leq O \left( \sqrt{\frac{L(f(x_0) - f^*)}{T}} + \frac{\min(n/s, \sqrt{n/s})B}{L} \right), \]

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. \( O \left( \frac{\sigma^2}{\mu k} \right) \).

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. [53] 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-ogasiewicz condition, \( O((\mu T)^{1/3}) \) rounds of communication suffice to achieve a linear speedup, that is, an error of \( O(1/\mu T) \). Patel and Dieuloue [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 \( O(\sqrt{T/N}) \).

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 \( \frac{1}{\sqrt{nT}} \) term dominate. In this situation, the convergence rate approximately changes into \( \frac{1}{\sqrt{nT}} \), 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 
\[ O \left( \frac{\sigma^2 + \sqrt{nT}}{\sqrt{nT}} + \frac{\xi^2 + \sqrt{nT}}{nT} + \frac{1}{T} + \frac{\delta^3\log T}{T} \right), \]
in which \( \sigma \) is the variance of stochastic gradient estimator, \( \xi \) is the variance of gradient divergence between single worker and all workers. For ECD-PSGD, they proved its convergence rate is 
\[ O \left( \frac{(1 + \frac{\sigma^2\log n}{nT})}{\sqrt{nT}} + \frac{\xi^2 (1 + \frac{\sigma^2\log n}{nT})}{T} + \frac{1}{T} + \frac{\delta^3\log T}{T} \right), \]
which is slightly worse than DCD-PSGD, due to the extra terms.

In the condition of convex objectives, their algorithm is able to converge with rate 
\[ O \left( \frac{1}{(nT)} + \frac{1}{(T\delta^2\omega)} \right), \]
in which \( \delta \) denotes the eigen gap of the gossip matrix and \( \omega \leq 1 \) the compression ratio.

Koloskova et al. [164] proved that CHOCO-SGD can converge at the rate \( O(1/\sqrt{nT} + n/(\rho^4\sigma^2 T)) \) on non-convex smooth functions, where \( n \) denotes the number of nodes, \( T \) the number of iterations, \( \rho \) the spectral gap of the mixing matrix and \( \sigma \) 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.

Asrran 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, 139, 138, 189, 133, 57, 206] of error accumulation, we can summarize the error accumulation as follows:

gradient compression: \( C_{i,t} = \text{Sparse}(v_{i,t-1} + \nabla v_{i,t}) \),
error accumulation: \( v_{i,t} = \nabla v_{i,t} - C_{i,t} \),
update weight: \( x_{t+1} = x_t - \gamma \frac{1}{n} \sum_{i=1}^{n} C_{i,t} \)
where \( C_{i,t} \) represents the updates which are compressed by any compression method \( \text{Sparse}() \), \( \nabla \) 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 [23], 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:

momentum accumulation: \( u_{i,t} = m u_{i,t-1} + \nabla v_{i,t} \),
error accumulation: \( v_{i,t} = v_{i,t-1} + u_{i,t} \),
update weight: \( x_{t+1} = x_t - \gamma \sum_{i=1}^{n} \text{sparse}(v_{i,t}) \)
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 Correction, of which the update process becomes:

momentum: \( u_{i,t} = \nabla v_{i,t} - m(x_t - x_{t-1}) \),
error: \( v_{i,t} = v_{i,t-1} + u_{i,t} - \text{sparse}(v_{i,t-1} + u_{i,t}) \),
update weight: \( x_{t+1} = x_t - \gamma \sum_{i=1}^{n} \text{sparse}(v_{i,t-1} + u_{i,t}) \)

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 \( \text{thr} \) for its local gradient \( G_k \), and the aggregation of gradients has a threshold \( \text{thr} \) for the global gradient \( G := \sum_{k=1}^{N} G_k \). Assuming all \( N \) workers have IID gradient distribution, with the variance \( \sigma^2 \), then the aggregation of all gradients have the variance \( N\sigma^2 \). So there are

\[
E[\|G\|_2] \approx \sigma, \quad E[\|G\|_2] \approx N^{1/2}\sigma.
\] (27)
### TABLE IV

**Summary of distributed learning algorithms**

<table>
<thead>
<tr>
<th>Arch.</th>
<th>Comm.</th>
<th>Compression</th>
<th>Communication Cost in Big O</th>
<th>Convergence in Big O</th>
<th>References of Method</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Server</td>
<td>Workers</td>
<td>convex</td>
</tr>
<tr>
<td><strong>BSP</strong></td>
<td>Null</td>
<td>$O(32nNT)$</td>
<td>$O(32NT)$</td>
<td>$O\left(\frac{1}{T}\right)$</td>
<td>56</td>
</tr>
<tr>
<td></td>
<td>Quant.</td>
<td>$O(32nNT)$</td>
<td>$O(bNT)$</td>
<td>$O\left(\frac{1}{T}\right)$</td>
<td>[175, 172]</td>
</tr>
<tr>
<td></td>
<td>Spars.</td>
<td>$O(32nNT)$</td>
<td>$O(k\log N + 32T)$</td>
<td>$O\left(\frac{1}{\sqrt{T}}\right)$</td>
<td>[207, 184]</td>
</tr>
<tr>
<td><strong>SSP</strong></td>
<td>Null</td>
<td>$O(32N\sum_{i} T_i)$</td>
<td>$O(32NT_i)$</td>
<td>-</td>
<td>$O\left(\frac{1}{\sqrt{T}}\right)$</td>
</tr>
<tr>
<td></td>
<td>Quant.</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Spars.</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td><strong>ASP</strong></td>
<td>Null</td>
<td>$O(32N\sum_{i} T_i)$</td>
<td>$O(32NT_i)$</td>
<td>$O\left(\frac{1}{T}\right)$</td>
<td>[73]</td>
</tr>
<tr>
<td></td>
<td>Quant.</td>
<td>$O(32N\sum_{i} T_i)$</td>
<td>$O(bNT_i)$</td>
<td>$O\left(\frac{1}{T}\right)$</td>
<td>[122]</td>
</tr>
<tr>
<td></td>
<td>Spars.</td>
<td>$O(32N\sum_{i} T_i)$</td>
<td>$O(k\log N + 32T_i)$</td>
<td>$O\left(\frac{1}{\sqrt{T}}\right)$</td>
<td>[141, 219]</td>
</tr>
<tr>
<td><strong>L-SGD</strong></td>
<td>Null</td>
<td>$O(32N\sum_{i} \frac{T_{comm}}{T_{comm}})$</td>
<td>$O(32N+\frac{T_{comm}}{T_{comm}})$</td>
<td>$O\left(\frac{1}{\sqrt{T}}\right)$</td>
<td>[73]</td>
</tr>
<tr>
<td></td>
<td>Quant.</td>
<td>$O(32N\sum_{i} \frac{T_{comm}}{T_{comm}})$</td>
<td>$O(bN+\frac{T_{comm}}{T_{comm}})$</td>
<td>$O\left(\frac{1}{T}\right)$</td>
<td>[122]</td>
</tr>
<tr>
<td></td>
<td>Spars.</td>
<td>$O(32N\sum_{i} \frac{T_{comm}}{T_{comm}})$</td>
<td>$O(k\log N + 32T_i)$</td>
<td>$O\left(\frac{1}{\sqrt{T}}\right)$</td>
<td>[141, 219]</td>
</tr>
</tbody>
</table>

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” (Systolic 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

$$\text{thr} (G_k) = N^{-1/2} \cdot \text{thr} (G).$$  \hspace{1cm} (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 \[228\],\[229\] 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 topologies. 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 × 32, and sparsification × 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 threshold 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


Deep Learning


