2021 IEEE International Conference on Cluster Computing (CLUSTER) | 978-1-7281-9666-4/21/$31.00 ©2021 IEEE | DOI: 10.1109/Cluster48925.2021.00054

$\mathcal{O}(1)$ Communication for Distributed SGD through Two-Level Gradient Averaging

Subhadeep Bhattacharya† Weikuan Yu† Fahim Tahmid Chowdhury† Kathryn Mohror‡
†Florida State University ‡Lawrence Livermore National Lab
{bhattach, yuw, fchowdhu}@cs.fsu.edu mohror1@llnl.gov

***Abstract*—Large neural network models present a hefty communication challenge to distributed Stochastic Gradient Descent (SGD), with a per-iteration communication complexity of $\mathcal{O}(n)$ per worker for a model of n parameters. Many sparsification and quantization techniques have been proposed to compress the gradients, some reducing the per-iteration communication complexity to $\mathcal{O}(k)$, where $k \ll n$. In this paper, we introduce a strategy called two-level gradient averaging (A2SGD) to consolidate all gradients down to merely two local averages per worker before the computation of two global averages for an updated model. A2SGD also retains local errors to maintain the variance for fast convergence. Our analysis shows that A2SGD converges similar to the default distributed SGD algorithm. Our evaluation validates the conclusion and demonstrates that A2SGD significantly reduces the communication traffic per worker, and improves the overall training time of LSTM-PTB by $3.2\times$ and $23.2\times$, compared to Top-K and QSGD, respectively. We evaluate the effectiveness of our approach using two kinds of optimizers, SGD and Adam. Also, our evaluation with various communication options demonstrates the strength of our approach both in terms of communication reduction and convergence. To the best of our knowledge, A2SGD is the first to achieve $\mathcal{O}(1)$ communication complexity per worker without incurring a significant accuracy degradation of DNN models while communicating only two scalars representing gradients per worker for distributed SGD.**

I. Introduction

Deep learning has found great success in image classification, speech recognition, and language processing [1], [2], etc. The demand for more powerful and accurate Deep Neural Networks (DNNs) leads to large and complex models with more than 1 Billion parameters, such as GPT-2 (1.5B) [3], Transformer (6B) [4], Turing-NLG (17B) [5], GPT-3 (175B) [6] and recently Switch-C (1.6T) [7]. Such large-scale models require distributed Stochastic Gradient Descent (SGD) algorithms for training. Distributed SGD typically adopts data parallelism, in which P workers hold the same model $w \in \mathbb{R}^n$ of n parameters and train it in parallel through many iterations. At the t-th iteration, weight w is updated as follows based on the learning rate η_t and the gradients g:

$$w_{t+1} = w_t - \eta_t \frac{1}{P} \sum_{p=1}^{P} g_t^p, \tag{1}$$

where a worker computes local gradients g_t^p (of the same size n) for the model using its fraction of a mini-batch, and exchanges the gradients across all workers for an updated global model. Such a global exchange and synchronization problem imposes a hefty requirement on both the latency and bandwidth of distributed systems, and hampers the scalability of distributed SGD [8]–[12]. Various strategies have been proposed to tackle this problem by increasing the mini-batch sizes [13]–[15], reducing the rounds of communication [16]–[18], or pruning the neural networks [19]–[22].

Particularly, there exists a fundamental bottleneck, i.e., the need to transfer $\mathcal{O}(n)$ local gradients for each worker. Many studies have proposed to compress the gradients through quantization [9], [18], [23]–[26] and/or sparsification [27]–[30]. Quantization enables lossy compression of gradients by reducing the precision of their representation to a varying degree, from 1BitSGD [9], [31] with only a sign bit, TernGrad [23] with three numerical levels {-1, 0, 1}, to QSGD [24] that supports multiple quantization levels. These quantization techniques can reduce the magnitude of each gradient during communication by at most 32 times per iteration, assuming gradients are single-precision floating-point numbers.

Gradient sparsification can achieve higher compression by selecting only k out of n gradients to reduce the communication traffic per worker [28]–[30]. Usually, k is defined as $x * d$, where x represents some fraction of gradient density (d). The selection criteria of k can be based on a user-defined threshold (Top-K) [29], a gaussian-estimated threshold (Gaussian-K) [28], or simple randomization (Rand-K) [30]. Prior results [27], [30] have shown that, theoretically, sparsified SGD can converge within the same upper bound as the original distributed SGD (dense SGD) algorithm, which exchanges full gradients. In practice, they have different convergence behaviors, for which Shi et al. [28] have performed a theoretical analysis to distinguish them.

In this paper, we propose a novel algorithm different from both sparsification and quantization. Our algorithm *two-level gradient averaging (A2SGD)* consolidates all local gradients down to merely two local means and achieves *a per iteration communication complexity of $\mathcal{O}(1)$ per worker*. It then aggregates the local means into two global means across all workers for an updated model. The key idea behind A2SGD is not to drop or quantize any gradient but to average all the positive and negative local gradients layer-wise while recording the difference between the gradients and the resulting means locally at each worker. In doing so, A2SGD retains local errors to maintain the same variance across gradients as dense SGD,

978-1-7281-9666-4/21/$31.00 ©2021 IEEE
DOI 10.1109/Cluster48925.2021.00054

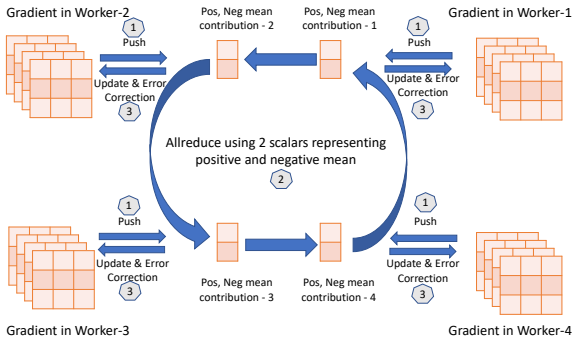
avoiding any potential variance blowup or any increase in the number of iterations. A2SGD does not require complex sampling or sorting of gradients but only simple calculations for the two means and their differences with the gradients. Our analysis shows that A2SGD converges similarly to dense SGD. Our evaluation validates the conclusion and demonstrates that A2SGD significantly improves the execution time per iteration and the overall training time, by $3.2\times$ and $23.2\times$ compared to Top-K and QSGD, respectively for LSTM-PTB, a big model with around 66 million parameters. Compared to the default dense SGD algorithm, A2SGD improves the overall training time of LSTM-PTB by $1.72\times$. Besides, A2SGD achieves the best overall performance in terms of convergence accuracy, execution time, and scaling efficiency in comparison to other techniques such as Top-K, Gaussian-K and QSGD.

Our Contributions. In summary, by examining the scalability challenge of gradient synchronization in distributed SGD and analyzing its computation and communication complexities, we have proposed a two-level gradient averaging algorithm, A2SGD, for distributed workers to exchange only two means globally. Our analysis and experimental results have confirmed the convergence of A2SGD, and demonstrated that it achieves an overall improvement compared to other sparsification and quantization algorithms [24], [28], [30]. Our results also show that A2SGD achieves fast computation complexity as discussed in §IV-C2. To the best of our knowledge, A2SGD is the first to achieve $\mathcal{O}(1)$ per iteration communication complexity per worker for distributed SGD. Moreover, Our contributions can be summarized as follows:

- We study different quantization and sparsification approaches and analyze that the additional computation introduced by such approach can diminish the benefit of communication reduction.
- We describe and implement A2SGD using Horovod [32], PyTorch [33] and MPI [34], which significantly reduces the communication traffic by exchanging only the positive and negative mean contribution from all nodes during allreduce.
- While A2SGD avoids gradient synchronization across all workers at the end of each iteration, it uses error correction mechanism and ensure consistent parameter and gradient across all workers after fixed number of epochs.
- We empirically show that our method achieves significant communication reduction during allreduce while maintaining quality of the model in terms of Top-1 accuracy and validation loss.
- We evaluate the benefit of our method using different communication options both using Remote Direct Memory Access (RDMA), TCP and Hierarchical collectives (HCOLL) [35] while utilizing MPI for Allreduce.
- We analyze the effectiveness of our method in further detail by using both SGD and Adam optimizers. It shows that our method converges within fixed epoch without significant degradation of the model while comparing with Dense SGD.

Our proposed A2SGD algorithm will have broader impact to the field of high-performance deep learning through distributed SGD, particularly on the use of big neural network models and the deployment of large-scale computer systems and high-speed networks. With its dramatic reduction on the communication complexity of gradient synchronization, A2SGD will facilitate the adoption of big DNN models for a wide variety of image classification, speech recognition and natural language processing applications.

II. Related Work



(a) Allreduce operation in A2SGD

Fig. 1: Design of A2SGD

A. Gradient Quantization.

Gradient quantization takes advantage of the fact that distributed SGD can still converge with low-precision gradients instead of 32-bit floating-point representations. A wide variety of quantization techniques have tried to represent gradients in 16 bits [36]–[38], 8 bits [39], 2.8 bits [24], 2 bits [40], or even 1 bit [9], [31]. In addition, Wen et al. [23] have quantized gradients from workers to the server using ternary values {-1, 0, 1}. Furthermore, some studies have provided theoretical analysis on the convergence guarantees of quantization techniques [18], [24], [25], [41]–[43]. Notwithstanding the compulsory cost for quantizing the gradients, quantization is inherently limited by its optimization scope, i.e., the number of bits representing the gradients. Thus it can reduce the network traffic by at most 32x compared to 32-bit numbers while using the quantization technique alone without any additional compression. The overall improvement of the time per iteration or the total training time is further limited for training large-scale models using distributed SGD.

B. Gradient Sparsification.

Unlike gradient quantization, sparsification examines the total number (n) of gradients and selectively transfers only a small number (k) of them while still allowing DNN models to converge. Because k can be several orders of magnitude smaller than n, sparsification techniques [9], [18], [27], [29], [30], [44]–[46] are proved to be much more effective than quantization in

reducing the communication traffic. Several studies [9], [29] have differentiated gradient values by magnitude and purged the small ones under a threshold. [44] adopted a number of optimizations to achieve very high sparsity in the exchanged gradients and carefully tuned the hyperparameters of DNN models to avoid any loss of accuracy.

Various recent studies [27], [30], [45] theoretically analyzed the performance of sparsification and established various bounds on the convergence rate. Nonetheless, it is imperative for these techniques to process all gradients, at certain computation costs, to reach their desired sparsity levels. For example, the selection procedure of top k number of elements from a given gradient vector involves additional computation overhead. As an alternative, Shi et. al. [28] have recently proposed a technique to remove the additional computation and take advantage of the gaussian distribution property. They statistically pinpoint a threshold to select the top k gradients at low computation latency by thresholding the values using GPU in parallel.

III. Design of Distributed SGD with Two-Level Gradient Averaging

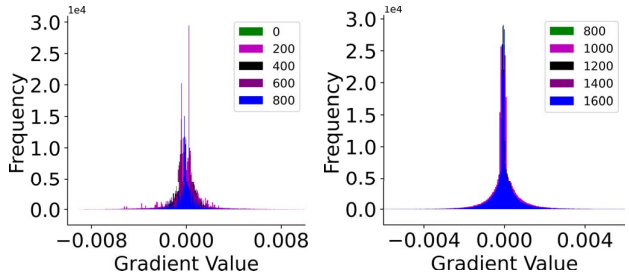
As mentioned in §I, gradient synchronization imposes a fundamental scalability challenge for data-parallel distributed SGD due to the requirement for all workers to exchange their gradients. While the sparsity and quantization levels are important to the per-iteration communication complexity of gradient synchronization, the computation efficiency of sparsification and quantization can be critical to its scalability as well. [28] reported that, while Top-K sparsification reduces the communication traffic, its computation overhead can offset the overall benefit, resulting in a suboptimal improvement on the execution time per iteration. On systems with high-bandwidth communication networks at 100 Gbps or higher, the computation costs from Top-K sparsification can overshadow its gains on communication efficiency, as we have observed in our experimental evaluation (§IV). The same tradeoff happens to quantization techniques such as QSGD. Shi et. al. [28] proposed Gaussian-K to avoid costly sorting and selection of top k elements across all gradient values. Gaussian-K assumes a gaussian distribution of gradient values and estimates a statistical threshold for the selection of gradient values. It has demonstrated the importance of low computation for sparsification. However, selection of k poses another challenge on maintaining the desired quality of model which sometimes requires manual adjustment on density values. Dynamic densities and careful hyperparameter tuning can become a primary concern for achieving the desired accuracy while reducing the communication load in sparsification based methods. Moreover, Sparsification and quantization can also be combined and generalized as compression techniques for the improvement of gradient synchronization [18], [25], [47]. All these studies have mitigated the computation costs of gradient while allowing the models to converge. Even so, all of them require the workers in distributed SGD to exchange some fraction of their gradients.

We propose an alternative to sparsification and quantization techniques. Instead of selecting a top fraction of gradient values, we can exchange the mean across the distributed workers. To avoid over simplification caused by a unified mean, we arrange the gradient values of each layer into two groups: positive (≥ 0) and negative (< 0), and compute their absolute means accordingly. Then all workers can exchange these two means for the synchronized global mean values. A global negative mean is computed by averaging the negative means from all workers; and a global positive mean by averaging the positive means. We refer to our algorithm as *Two-level Gradient Averaging (A2SGD)*. It effectively reduces the communication traffic down to two values, achieving the per iteration communication complexity of $\mathcal{O}(1)$.

Fig 1 shows the distributed training using multiple workers in the data-parallel model. As shown in Fig. 1a, in Step 1 of our approach, each worker computes the gradient after going through the Feed-Forward phase where a mini-batch of data is fed across all the neural net layers in the forward direction. After the gradients are generated in each worker, each worker flattens the multidimensional gradient vector. Then it contributes the positive and negative mean of the gradients and participates in allreduce as shown by Step 2 in Fig. 1a. After each worker completes the allreduce operation and gets a synchronized copy of positive-negative contributions, the error-correction is performed. Finally, each worker continues with their Back Propagation and iteratively updates the model parameters, shown as Step 3 in Fig. 1(a).

While this is different from gradient clipping in prior studies [23], [25], [44], it would also lead to some distorted gradients for the workers. Several prior studies [9], [25], [30], [31], [41] pursued the idea of error-feedback to correct either the momentum or variance, or both, to improve the convergence accuracy of training. To limit the impact on variance, we equip A2SGD with a similar error-feedback mechanism through a local error vector. Each workers need to perform some additional tasks in order to efficiently store and handle the error-feedback mechanism We compute the difference between the gradients and the two means, and use a local error vector to store the difference. Moreover, the subtraction operation takes place for 1) the positive values which are greater than the positive mean and 2) the negative values which are less than the negative mean. When the global synchronization completes, the error vector is added back to the global means, according to the corresponding positions of the original positive and negative gradients, to generate the updated gradients. We keep the values lying in the interval between negative and positive means as they are. Although, these values remain intact for the particular iteration, they participate in later phase of the training and try to converge towards zero.

Figure 2 shows the frequency distribution of gradients and its progression with an increasing number of iterations, for two representative models: FNN-3 and ResNet-20. Most of the values are close to zero on either side, following a normal distribution. Besides, as the models finish more iterations of the training, more gradient values converge to the center around



(a) FNN3 (b) ResNet20

Fig. 2: Progression of Gradient Distribution with number of iterations.

zero. These distribution plots from one representative worker provide a visualization of the convergence of gradients across all workers. The distribution of gradients has been previously studied thoroughly in [48] which supports our results.

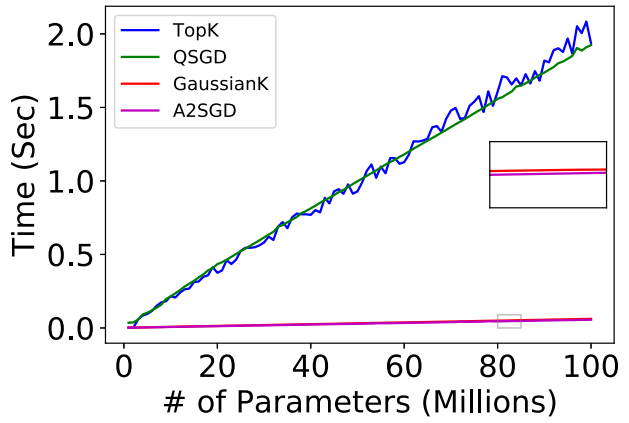


Fig. 3: Comparison of A2SGD computation time with other algorithms.

For an initial assessment on A2SGD's computation cost, we have measured its computation time with an increasing number of parameters, compared to Top-K, Gaussian-K and QSGD. A2SGD, Top-K, Gaussian-K use PyTorch [33] APIs with GPU support, and QSGD is implemented in Numpy [49] with CPU specific implementation for quantization (see §IV-A for more details on our experimental setup). Figure 3 shows that A2SGD and Gaussian-K have much lower computation latency than QSGD and Top-K. A2SGD has a slightly lower computation cost than Gaussian-K because Gaussian-K has to estimate a threshold [28] before gradient selection. We have elaborated the detailed computation complexity for these algorithms in §IV-C. These initial results on the computation time suggest that A2SGD is very promising to support efficient gradient synchronization because of its significant reduction on communication traffic at very low computation costs.

A. Details of A2SGD

For a gradient vector $v = \{v_1, v_2, ..., v_n\} \in \mathbb{R}^n$, we denote $\mu_+(v) = \mathbb{E}\{(v_i) \mid v\}\ \forall\ v_i \geq 0$ as the absolute mean of all positive values v_i in v, and $\mu_-(v) = \mathbb{E}\{|v_i| \mid v\}\ \forall\ v_i < 0$, the absolute mean of all negative values in v. We introduce a new operator **enc** below to transform the values of v.

$$\mathbf{enc}(v) = pos(v) \cdot \mu_+(v) - neg(v) \cdot \mu_-(v) \tag{2}$$

where $pos(v)$ and $neg(v) \in \mathbb{R}^n$ are vectors with values $\in \{1, 0\}$. The former has 1 in the corresponding positions $\forall\ v_i \geq \mu_+(v) \wedge \forall\ v_i \geq 0, i \in [1, n]$, and the latter with 1, $\forall\ v_i \leq (-1) \cdot \mu_-(v) \wedge \forall\ v_i < 0, i \in [1, n]$. This encoding helps to keep track of the corresponding indices for both positive and negative values of the gradient vector during error correction.

Algorithm 1 describes the proposed A2SGD algorithm in detail. Each worker p starts with a learning rate η_0 and an initial weight w_0. At any iteration t, worker p computes its stochastic gradients g_t^p by training SGD with its portion of mini-batch M_t^p **(Line 2)**. It then extracts the means for positive and negative gradients **(Line 3)** and subtracts the vector constructed from the means **(Line 4)**. The errors are stored in a local error vector ϵ_t^p. All workers call the Allreduce operation to exchange their local means and get back the global means **(Line 5)**. The global means are then combined with the errors stored in ϵ_t^p into the new gradients g_t' and eventually as new g_t^p **(Line 6 and 7)**. Assume, worker 1 has gradients [+1,+2,-1] and worker 2 [-1,-2,+3], A2SGD will send [1.5, 1] from worker 1 and [3, 1.5] from worker 2 during allreduce. The first one in mean vector represents the positive contribution while the second one represents the negative contribution. The value of $pos(g_t^p)$ and $neg(g_t^p)$ needs to be calculated before performing encoding for local error calculation. In worker 1, $pos(g_t^p)$ and $neg(g_t^p)$ is calculated as [0, 1, 0] and [0, 0, 1] respectively. Similarly, in worker 2, the value of $pos(g_t^p)$ = [0, 0, 1] and $neg(g_t^p)$ = [0, 1, 0]. The value of $\mathbf{enc}(g_t^p)$ becomes [0, +1.5, -1] and [0, -1.5, +3]. Local errors, i.e., Worker-1:[+1, +0.5, 0], Worker-2:[-1,-0.5,0], are stored in both workers and applied after allreduce. After the allreduce and error correction is completed, each worker hold the synchronized mean values [2.25, 1.25] and the updated gradient vector, Worker 1: [+1, +2.75, -1.25] and Worker 2: [-1, -1.75, 2.25] For a gradient vector of dim=1 (model with only one parameter), we replace the missing scalar representing the mean with zero. Finally, the model weight is updated at **Line 8** using the new gradient and the current learning rate. At the end of the training iterations, one more iteration is performed to synchronize the model across all workers (**Lines 10, 11 and 12**).

The gradient generated in data parallel dense SGD can be expressed as : $g_t = \frac{1}{P}\sum_{p=1}^{P} g_t^p$

However, for each iteration of A2SGD, gradient g_t^p subtracts its own means and then gains back the global means. We are basically assuming the following properties: (1) Most of the gradients are located near zero and (2) We can achieve acceptable accuracies as shown in previous studies [24], [25], [27], [28], [50] without transferring the entire gradient vector.

Algorithm 1 Parallel A2SGD Algorithm at Worker p

Input: Initial learning rate η_0, weight $w_0 = \vec{0}$
Output: Learned model consists of weight w_T after T^{th} iteration

for $t \leftarrow 1$ to $T-1$ **do**
  $g_t^p \leftarrow SGD$ through training with M_t^p ▷ training M_t^p, portion of mini-batch for p
  $\mu_{t,+}^p \leftarrow \mu_+(g_t^p)$ and $\mu_{t,-}^p \leftarrow \mu_-(g_t^p)$ ▷ Calculate positive and negative mean in each worker
  $\epsilon_t^p \leftarrow g_t^p - \mathbf{enc}(g_t^p)$ ▷ Calculate error in ϵ_t^p for error correction after allreduce
  $(\bar{\mu}_{t,+}, \bar{\mu}_{t,-}) \leftarrow \mathbf{Allreduce}((\mu_{t,+}^p, \mu_{t,-}^p), \mathbf{average})$ ▷ Allreduce in t^{th} iteration using pos and neg means
  $g_t' \leftarrow \epsilon_t^p + pos(g_t^p) \cdot \bar{\mu}_{t,+} - neg(g_t^p) \cdot \bar{\mu}_{t,-}$ ▷ Error Correction ϵ_t
  $g_t^p \leftarrow g_t'$ ▷ Assign the error corrected value as new gradient
  $w_{t+1}^p \leftarrow w_t^p - \eta_t \cdot g_t^p$ ▷ Update the model
end for
$w_{T-1} \leftarrow \mathbf{Allreduce}(w_{T-1}^p, \mathbf{average})$
$g_{T-1} \leftarrow \mathbf{Allreduce}(g_{T-1}^p, \mathbf{average})$ ▷ Global synchronization with entire gradient vector from all workers
$w_T \leftarrow w_{T-1} - \eta_{T-1} \cdot g_{T-1}$ ▷ Global update of the model at last iteration

We assume set(A) as the set of all gradients in a local worker and we are selecting a subset of them in each iteration using mean values of positive and negative gradient values as threshold. Rest of the content in gradient vector are kept in residual for future error correction. Furthermore, it selects worker zero as the master node and expects to get a consistent copy of the model after T^{th} iteration. Although, our approach is implemented in a decentralized data parallel way, it can be easily adopted to Parameter Server setting. We can denote $\bar{\mu}_t$ as the vector composed global means, and the net gain as :

$$\Delta\mu_t = \bar{\mu_t}^p - \mathbf{enc}(g_t^p)$$

As per our algorithm,

- $\bar{\mu_t} = [pos(g_t^p).\bar{\mu_{t,+}} - neg(g_t^p).\bar{\mu_{t,-}}]$
- $enc(g_t) = [pos(g_t^p).\mu_{t,+}^p - neg(g_t^p).\mu_{t,-}^p]$
- $\bar{\mu_{t,+}} = \frac{1}{P}\sum_{p=1}^{P} \mu_{t,+}^p$ and $\bar{\mu_{t,-}} = \frac{1}{P}\sum_{p=1}^{P} \mu_{t,-}^p$

Furthermore- $\Delta\mu_t =$

$$[(pos(g_t^p).\bar{\mu_{t,+}} - neg(g_t^p).\bar{\mu_{t,-}}) - (pos(g_t^p).\mu_{t,+}^p - neg(g_t^p).\mu_{t,-}^p)]$$

$$= [pos(g_t).\bar{\mu_{t,+}} - neg(g_t).\bar{\mu_{t,-}} - pos(g_t^p).\mu_{t,+}^p + neg(g_t^p).\mu_{t,-}^p]$$

$$= pos(g_t^p)[\bar{\mu_{t,+}} - \mu_{t,+}^p] - neg(g_t^p)[\bar{\mu_{t,-}} - \mu_{t,-}^p]$$

The updated gradient in A2SGD approach for each iteration can be denoted as : $g_t' = g_t^p + \Delta\mu_t$

Finally, in synchronous data parallel SGD (SSGD), we get consitent value of gradients across each worker which in turn denotes the allreduced mean of each coordinate across workers. However, the Asynchronous version of it claims to converge but suffers from significant slowdown due to stale update as shown in [51]. In our approach the view of model is not consistent across worker till T^{th} iteration. Hence, we can assume in any intermediate iteration, gradient in first worker represents the entire worker vector. However, due to synchronization in the T^{th} iteration, we get a final consistent view of the model parameters.

IV. Empirical Results

In this section, we first describe our experimental setup, then we present our evaluation results validating the convergence of A2SGD. In addition, we compare its performance with dense SGD (Dense in short), two sparsification techniques Top-K [30] and Gaussian-K [28] and one quantization technique QSGD [24]. Our performance evaluation covers several aspects, including convergence accuracy, computation and communication complexities, scaling efficiency, execution time, and effectiveness of our approach with Adam optimizer.

A. Experimental Setup

We employ data-parallel SGD for all the experiments with the data divided among P workers, each maintaining a copy of entire model to train a portion of the entire dataset. Each worker also receives a copy of an entire gradient vector for weight update.

We implement A2SGD on top of PyTorch [33] v1.3.0 with CUDA [52] v10.1, and utilize Horovod [32] v0.19.1 with Allreduce [53] for data-parallel implementation of different models. Top-K and Gaussian-K implementations are adapted from a GitHub repository [54]. Both implementations use the PyTorch Tensor API. We adapt the QSGD implementation from a GitHub implementation [49]. We conduct all our experiments with varying number of Nvidia V100 GPUs (2 to 16). Each node in our testbed is equipped with 256 GB CPU memory and 1 V100 GPUs per node with 16 GB GPU memory. Furthermore, all nodes in the system are connected with a high-bandwidth 100-Gbps InfiniBand network.

In our tests, we employ four different DNN models, including (1) FNN-3 which is a Feed-forward Neural Network (FNN) with three hidden fully connected layers; 2) two types of Convolutional Neural Networks (CNNs), i.e., VGG-16 and ResNet-20 using CIFAR10 dataset; and 3) LSTM-PTB, i.e., the Long Short Term Memory (LSTM) model using Penn Treebank (PTB) dataset. We use LARS [14], Linear Scaling(LS), Gradual Warmup (GW) and Polynomial Decay (PD) of learning rate

TABLE I: Experimental Setup

Model	Dataset Train Sample Size	# Parameters	BatchSize/worker	LR	Policy
FNN-3	MNIST 60,000-[28x28] images	199,210	128	0.01	LS(1 x) + GW + PD
VGG-16	CIFAR10 50,000-[32x32] images	14,728,266	128	0.1	LS(1.5 x) + GW + PD + LARS
ResNet-20	CIFAR10 50,000-[32x32] images	269,722	128	0.1	LS(1 x) + GW + PD
LSTM-PTB	PTB 9,12,344 tokens	66,034,000	128	22	PD

(LR) for the large batch experiments. LARS is used in all the experiments related to VGG16, including Dense, Top-K, Gaussian-k and QSGD. In LS, we increase LR by multiplying it with the number of workers. LR value with a given number of workers is equal to $(baseLR * numWorker * n)$ if LS (nx) is specified. GW is executed to start from a smaller one and ramp-up to the base learning rate gradually in first 5 epochs. LR is also decayed using the equation $((1 - Epoch)/maxEpochs)^2$ in case Polynomial Decay(PD) is specified. In all figures, we label Top-K and Gaussian-K without the hyphen for brevity. Table I lists the detailed hyperparameters for these models.

1) Value of Sparsification (k) and Quantization (q) level:: For all the experiments we consider the value of k as $0.001 * d$ for Top-K and Gaussian-K. $0.001 * d$ represents that only 0.1% of the entire gradient is used for gradient synchronization of sparsification methods, i.e, k = 199, 14728, 269 and 66034 for FNN-3, VGG-16, ResNet20, LSTM-PTB. We perform all the QSGD experiments with quantization level $q = 4$ as we compare with the quantization functionality of QSGD without any additional compression technique.

B. Convergence Accuracy

To demonstrate the convergence of A2SGD, we run all four models, with 30 epochs for FNN-3, 150 epochs for VGG-16 and ResNet-20, and 100 epochs for LSTM-PTB with a varying number of workers. We measure the top-1 convergence accuracy for FNN-3, VGG-16 and ResNet-20, and the perplexity score[1] for LSTM. We perform each experiment multiple times for validation purposes and reported the results with required epochs to reach convergence.

Figure 4(a-d), Figure 4(e-h) and Figure 4(i-l) show the convergence performance with 4, 8 and 16 workers, respectively. These results demonstrate that, for all the cases, A2SGD achieves the closest top-1 accuracy to dense SGD within the same number of epochs, and outperforms the other algorithms in terms of convergence accuracy. Top-K performs the best overall among the rest of algorithms. A2SGD achieves 97.82%, 87.82%, 88.80% top-1 accuracy, and 135.53 perplexity for FNN-3, ResNet20, VGG16 and LSTM, respectively with 8 workers. In addition, A2SGD achieves 2.5% and 1.3% better top-1 accuracies than Top-K for ResNet20 and VGG16, respectively. Furthermore, Top-K, Gaussian-K, and QSGD all exhibit a varying amount of accuracy drops with more workers. Moreover, Figure 5a depicts the validation loss of ResNet50 with ImageNet as dataset. We use minibatch size of 128 with LR 0.1 for this set of experiments with 8 workers. Although validation loss for TopK and A2SGD is close to Dense SGD, A2SGD achieves better validation loss with time over Gaussian-K.

[1]Captures the degree of uncertainty a particular model have for predicting some text. Lower the better.

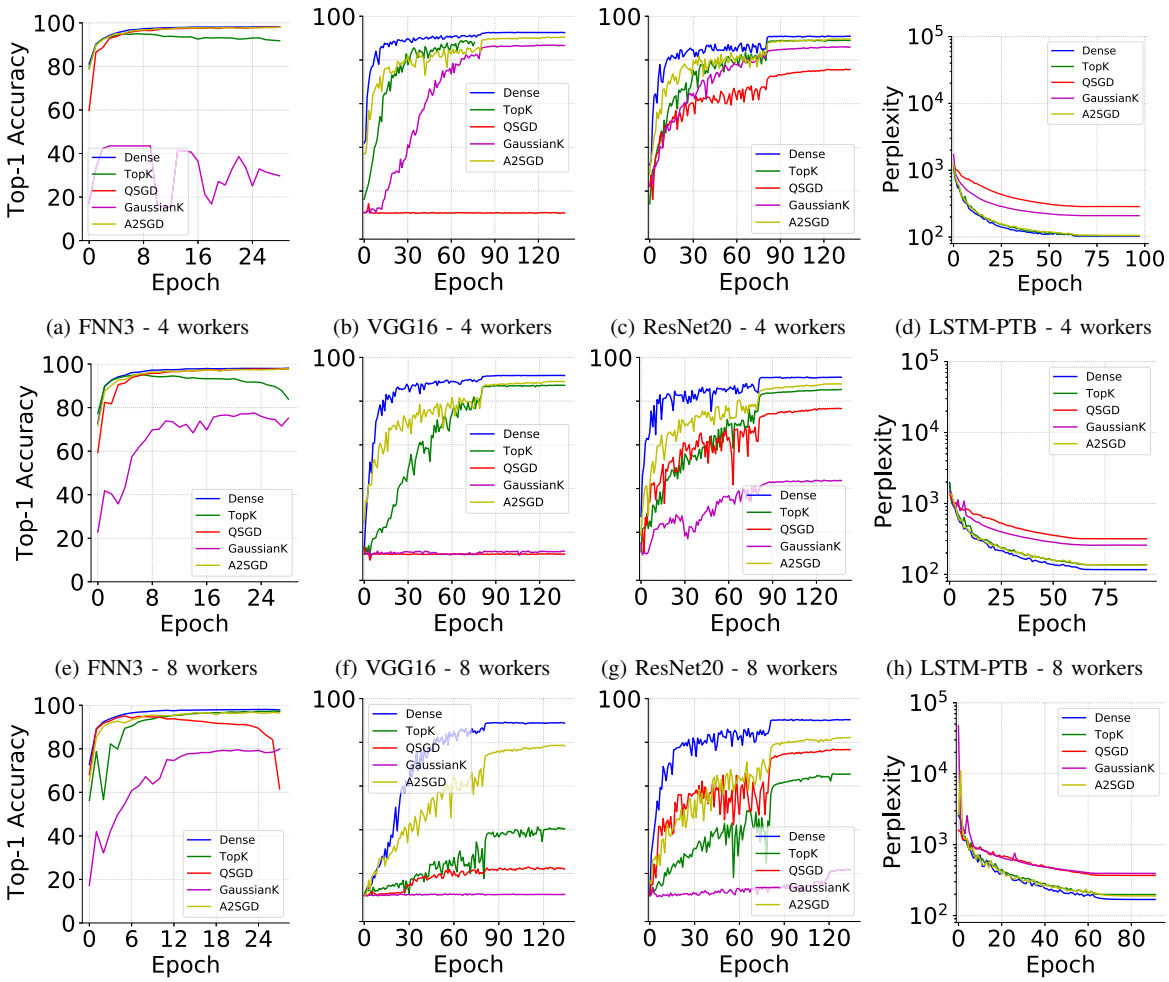
TABLE II: Comparison of Gradient Synchronization Complexities

Algorithm	Computation Complexity	Communication (# bits)
Dense SGD	$\mathcal{O}(1)$	$32n$
QSGD	$\mathcal{O}(n)$	$2.8n + 32$
Top-K	$\mathcal{O}(n + klogn)$	$32k$
Gaussian-K	$\mathcal{O}(n)$	$32k$
A2SGD	$\mathcal{O}(n)$	64

C. Gradient Synchronization Complexities and Scaling Efficiency

As discussed in §III, our A2SGD algorithm is designed to improve gradient synchronization with reduced communication traffic without costly computation to process the gradients. To gain an insight on its impact to computation and communication in gradient synchronization, we have characterized the asymptotic computation complexity and the amount of communication traffic (# bits) per worker for A2SGD, in comparison to dense SGD, QSGD, Top-K and Gaussian-K. In data-parallel distributed SGD, each worker hosts a full copy of the model and gradients after each training iteration. We assume a model with n parameters, i.e., n gradients.

1) Communication Complexity.: In terms of communication traffic, it is evident that dense SGD has to transfer all gradients from each worker, i.e., $32n$ bits. A2SGD transfers two means, i.e., 64 bits. Top-K and Gaussian-K both transfer k gradients, i.e., $32k$ bits, where $k = xd$ and d represents density of the gradient vector. [24] reported that QSGD transfers $2.8n + 32$ bits. Thus A2SGD is the only algorithm that achieves $\mathcal{O}(1)$ communication complexity per worker, which can greatly increase the communication efficiency in the

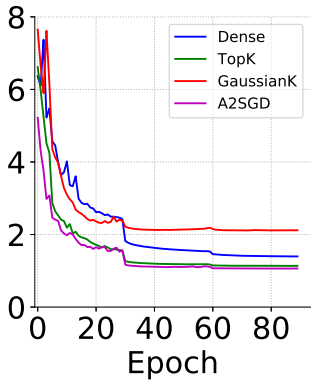


(a) FNN3 - 4 workers (b) VGG16 - 4 workers (c) ResNet20 - 4 workers (d) LSTM-PTB - 4 workers

(e) FNN3 - 8 workers (f) VGG16 - 8 workers (g) ResNet20 - 8 workers (h) LSTM-PTB - 8 workers

(i) FNN3 - 16 workers (j) VGG16 - 16 workers (k) ResNet20 - 16 workers (l) LSTM-PTB - 16 workers

Fig. 4: Comparison of Convergence Accuracy with 4, 8, 16 Workers



(a) ResNet50 - 8 workers

gradient synchronization of large DNN models. The amount of communication traffic for these algorithms is shown by Column 3 of Table II.

2) Computation Complexity.: Dense SGD does not process local gradients, and has a computation complexity of $\mathcal{O}(1)$. All other algorithms store a local residual or error vector from the transferred gradients, with a computation complexity of $\mathcal{O}(n)$. A2SGD has an overall computation complexity of $\mathcal{O}(n)$ because, for each model, it traverses all gradients to compute two separate averages, which is still $\mathcal{O}(n)$. Gaussian-K has a computation complexity of $\mathcal{O}(n)$ in its formulation of the gaussian estimation model as stated in [28]. In addition, it has an additional overhead to estimate the threshold based on its gaussian model.

In the Python implementation without GPU support [49], QSGD computes the second norm (a complexity of $\mathcal{O}(n)$) and applies quantization for each gradient. Thus its total computation complexity is $\mathcal{O}(n)$. A max heap-based implementation of Top-K has an overall computation complexity of $\mathcal{O}(n+klogn)$, where n is the complexity of constructing the max heap and

$klogn$ for selecting the largest k elements. The computation comparison is shown by Column 2 of Table II. Note that GPU based implementations can have higher complexity for the need of GPU parallelization [55], [56]. For both QSGD and Top-K, the cost to maintain a local error vector does not change the overall computation complexity. Our analysis of these algorithms in terms of their asymptotic computation complexity confirms the superb overall efficiency of A2SGD. While achieving $\mathcal{O}(1)$ communication complexity per worker, it maintains the minimal asymptotic computation complexity of $\mathcal{O}(n)$, without the overhead for threshold estimation like Gaussian-K. Furthermore, we evaluate the scaling efficiency along with the validation loss while using 8 workers for the training of A2SGD in comparison to the other algorithms. The reported validation loss in Table III follows the convergence accuracy, as reported in §IV-B. Our evaluation results show that A2SGD and Top-K achieve the closest loss to dense SGD than others. However, for VGG16, ResNet20 and LSTM, A2SGD achieves the lowest loss after the same number of epochs. A2SGD outperforms dense SGD by 0.20% and 0.6% for image classification models VGG16 and ResNet-20.

We also measure the throughput of each algorithm as the number of images processed per second. Then the scaling efficiency is calculated as the normalized throughput with an increasing number of workers. Since there is no gradient synchronization for only one worker, we use the throughput of dense SGD with two workers for normalization. Specifically, it is calculated as *Scaling Efficiency* = (t_8/t_2^D), where t_2^D is the throughput (average images processed per iteration) of dense SGD with 2 workers, and t_8 is the overall throughput with 8 workers for any specific algorithm. A higher throughput reflects the better efficiency in processing images. As shown by the column 2 in Table III, A2SGD and Gaussian-K have better scaling efficiency than the other three algorithms. The reason Gaussian-K performs similarly to A2SGD, because it uses Allgather implementation of gradient exchange as discussed in §IV-D.

D. Execution Time

Given our understanding on the complexity of A2SGD and its scaling efficiency, we further evaluate the benefit of A2SGD to the execution time of DNN training models with fixed epochs as mentioned in §IV-B. We first measure the average execution time per iteration for all algorithms with varying number of workers. Figure 6 shows the comparison across four different models. For the smaller models, i.e., FNN-3 and ResNet-20, A2SGD and Gaussian-K perform comparably to dense SGD, and slightly better than QSGD and Top-K. These two models have a smaller number of parameters, which lead to an insignificant difference on the 100-Gbps high-bandwidth network. The longer execution time per iteration of QSGD and Top-K is due to their higher computation costs.

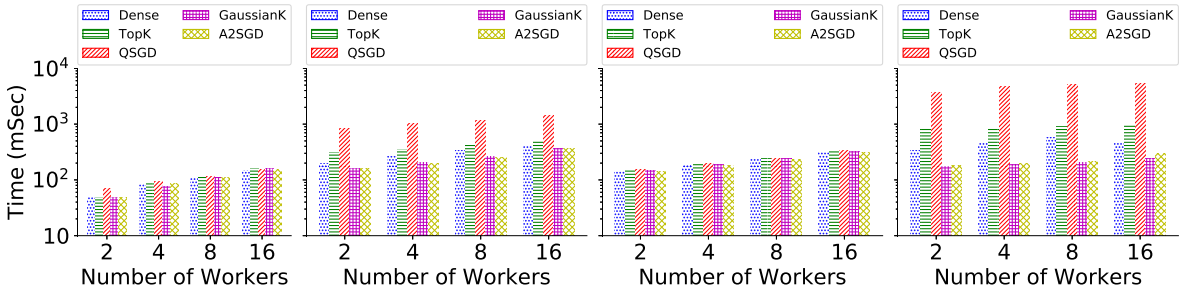
For the larger models, i.e., VGG-16 and LSTM-PTB, A2SGD and Gaussian-K deliver much faster execution time per iteration, compared to dense SGD, Top-K and QSGD. For the largest model, Gaussian-K achieves slightly better execution time per iteration than A2SGD. We could not attribute this difference to the comparisons listed in Table II. By examining the implementation of Gaussian-K, we realize that this is because Gaussian-K uses Allgather for exchanging gradients, which is faster than Allreduce adopted by A2SGD on 100-Gbps high-bandwidth networks [53], [57] with increasing number of workers. Furthermore, the execution time per iteration is always the longest for QSGD compared to dense SGD. The reason behind this is the overhead from its high computation domination over the benefit of communication reduction on the 100-Gbps high-bandwidth network. Moreover, all algorithms exhibit longer execution time per iteration with an increasing number of workers. This is because of the collective nature of gradient synchronization, i.e., more communication time for synchronization across more workers.

We also evaluate the benefit of A2SGD to the total execution time with an increasing number of workers. Figure 7 shows the comparison across all algorithms for four different models. Despite the increasing execution time per iteration, all algorithms deliver faster total execution time with more workers, a manifestation on the strength of data-parallel distributed SGD algorithms. Again, for FNN-3 and ResNet-20, A2SGD and Gaussian-K perform similarly to dense SGD, and slightly better than QSGD and Top-K, for the same reasons as previously stated. For the larger models VGG-16 and LSTM-PTB, A2SGD and Gaussian-K again achieve better performance than dense SGD, Top-K and QSGD. Gaussian-K is slightly faster than A2SGD but loses in terms of validation loss and perplexity score in case of LSTM-PTB. A2SGD is $3.2\times$ and $23.2\times$ faster compared to Top-K and QSGD in terms of total execution time. QSGD suffers from its high computation overhead compared to dense SGD, and its high communication costs compared to the rest of the models.

1) Slow Interconnect: We further compare the performance of our proposed A2SGD method with dense SGD while using slow interconnect. We have observed that, FNN-3 model consists of small number of parameters and ResNet-20 is considerably computation intensive than the others. Hence, we select VGG-16 and LSTM-PTB for this set of experiments as a representative model of image classification (using CNN) and text processing (using LSTM), respectively. We use **tcp** as the Byte Transfer Layer(BTL) in IBM Spectrum-MPI for selecting the slow interconnect. We observe that there is no further deterioration of accuracy while using the same hyperparameter configuration as before. However, A2SGD achieves considerable performance improvement in speedup and overall execution time compared with the default dense SGD approach. Figure 10 shows overall performance improvement in terms of speedup while using A2SGD with varying number of workers. The performance speedup is 70.6%, 128%, 94.6% and 88.4% for VGG-16 and $4.61\times$, $7.59\times$, $10.37\times$ and $8.08\times$ for LSTM-PTB using 2, 4, 8 and 16 workers respectively. Again, Fig 9 measures the overall execution time for A2SGD and dense SGD with varying number of workers. A2SGD clearly shows huge performance improvements over dense SGD with around 88.8% less exection time for LSTM-

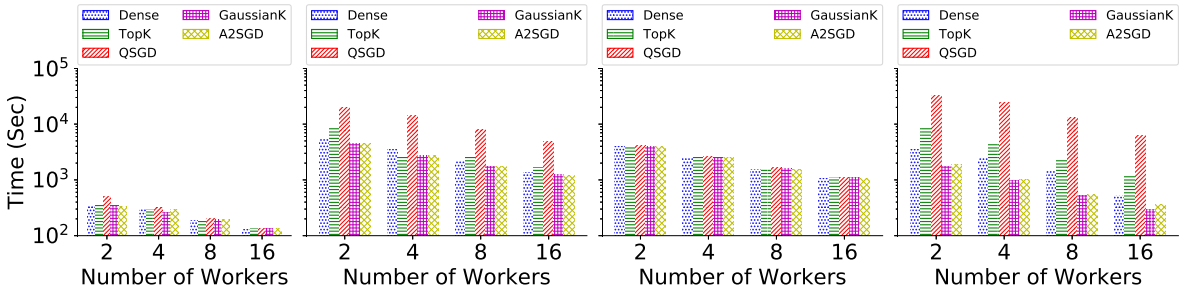
TABLE III: Comparison of Scaling Efficiency and Validation Loss

Algorithm	Scaling Efficiency (8 Workers) (FNN/VGG/ResNet/LSTM)	Validation Loss (8 Workers) (FNN/VGG/ResNet/LSTM)
Dense SGD	(**1.83** / 2.34 / 2.52 / 2.34×)	(**0.068** / 0.55 / 0.42 / **4.76**%)
QSGD	(1.73 / 0.66 / 2.34 / 0.26×)	(49.5 / 7.47 / 0.68 / 5.76%)
Top-K	(1.76 / 2.40 / 1.92 / 1.50×)	(0.07 / 0.45 / 0.45 / 4.92%)
Gaussian-K	(1.79 / 2.97 / 2.40 / **6.58**×)	(0.88 / 2.61 / 1.53 / 6.55%)
A2SGD	(1.80 / **3.06** / **2.50** / 6.37×)	(0.07 / **0.35** / **0.36** / 4.9%)



(a) FNN3 (b) VGG16 (c) ResNet-20 (d) LSTM

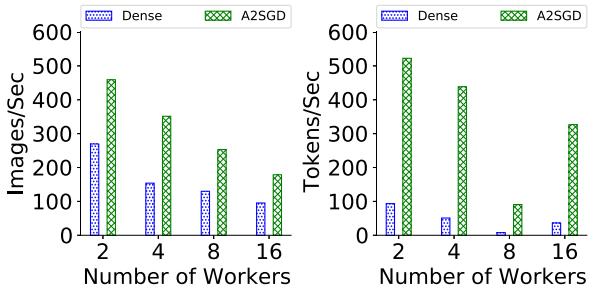
Fig. 6: Comparison of Average Execution Time



(a) FNN3 (b) VGG16 (c) ResNet-20 (d) LSTM

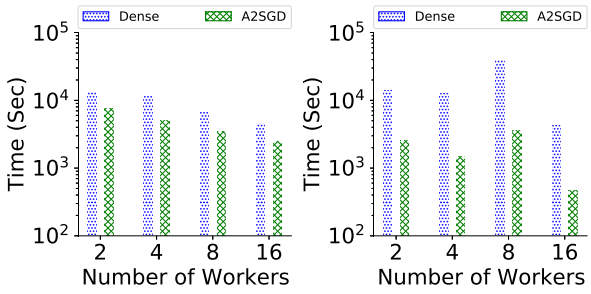
Fig. 7: Comparison of Total Execution Time

PTB which has around 66 Million parameters while running it on 16 nodes for same number of epochs.



(a) VGG16 (b) LSTM

Fig. 8: Speed with Slow Interconnect

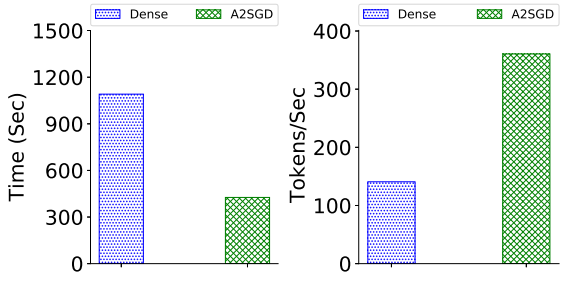


(a) VGG16 (b) LSTM

Fig. 9: Total Execution Time with Slow Interconnect

2) Using Hardware Collectives: Since A2SGD has low communication overhead due to transferring only constant

number of representative gradients in each epoch, we achieve significant performance improvements for A2SGD by changing the option from the default configuration of Spectrum-MPI to use hardware based collectives operation using HCOLL. We evaluate both the dense SGD and A2SGD using HCOLL and compare them. As described in Fig 10b, we use LSTM-PTB for this set of experiments as it has a large number of parameters. The overall speedup is around 1.56× for 16 workers when compared to dense SGD. Similarly, A2SGD completes the same number of iteration as Dense SGD with 61% lesser time than dense SGD while both using HCOLL as illustrated in Fig 10a.



(a) LSTM Exection Time (b) LSTM Speedup

Fig. 10: Total Execution Time and Speed with HCOLL

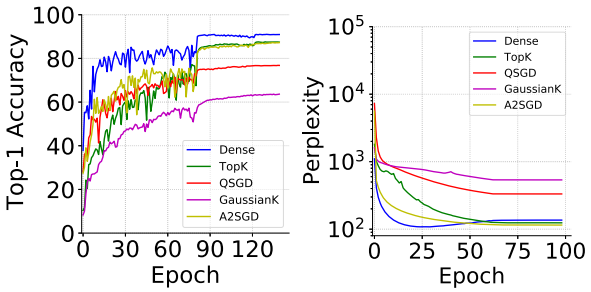
E. Performance Evaluation with Adam

While other method employs expensive allreduce with large number of data, the data needs to be allreduced in our method consists of only two scalars. We investigate further and check the impact of our approach while using Adam as our optimizer instead of SGD. We select ResNet-20 and LSTM as our representative dataset and incorporate Adam. We set the batchsize to 128 while keeping the LR at 0.01 and 0.001 for ResNet-20 and LSTM, respectively. The validation losses and total execution times are reported in Table IV. Dense SGD uses Adam optimizer instead of default SGD in this scenario and is used as baseline for comparison with other approaches.

Algorithm	Validation Loss (ResNet/LSTM)	Total Execution Time (Sec) (ResNet/LSTM)
Dense SGD	(0.398/4.92)	(3195/2966)
QSGD	(0.676/5.81)	(3194/8617)
Top-K	(0.407/4.82)	(3534/2578)
Gaussian-K	(1.037/6.29)	(3490/825)
A2SGD	(0.387/4.75)	(3147/796)

TABLE IV: Setup and Validation Loss using Adam Optimizer with 8 workers

A2SGD helps achieve the loss metric closer to dense SGD and better than other approaches while taking lowest total execution time among all. The top-1 accuracies are also reported in Fig 11. While Gaussian-K has closer total execution time to A2SGD, A2SGD performs far better in terms of Top-1 accuracy also for both ResNet-20 and LSTM-PTB. The reported validation losses for A2SGD are 0.65 and 1.54 less than Gaussian-K results after the same number of epochs. While Top-k achieves Top-1 accuracy and loss closer to A2SGD, the result also confirms that Top-K has much slower total execution time than what our approach reported. Moreover, A2SGD achieves better loss and top-1 accuracy than dense SGD for both the models with Adam as optimizer.



(a) ResNet20 (b) LSTM

Fig. 11: Accuracy using Adam Optimizer with 8 workers

V. Discussion

These evaluation results demonstrate that, while achieving $\mathcal{O}(1)$ communication complexity, A2SGD delivers the best overall performance regarding convergence accuracy, scaling efficiency and execution time. Also, note that, compared to Gaussian-K, A2SGD does not have to go through an initial estimation of the threshold, and its computation cost is slightly lower. Furthermore, A2SGD is an initial prototype that can be refined further. Our initial evaluation with different communication medium and optimizer shows the strength of A2SGD due to significantly reducing the communication bit required during distributed training while maintaining the quality of the models.

VI. Conclusion

In this paper, we have examined the scalability challenge of the gradient synchronization in distributed SGD and proposed a two-level gradient averaging algorithm, A2SGD, for distributed workers to exchange only two averages and achieve $\mathcal{O}(1)$ per iteration communication complexity per worker. Our experimental results have confirmed the convergence of A2SGD and demonstrated that A2SGD achieves an overall improvement compared to the other sparsification and quantization algorithms [24], [28], [30]. For all these algorithms, we systematically analyze the computation and communication complexities during gradient synchronization and point out that A2SGD outperforms the other approaches asymptotically. Moreover, A2SGD shows its portability and remarkable performance improvement over the rest of the compression techniques we evaluated.

ACKNOWLEDGMENT

We would like to thank Ms. Yue Zhu from the Computer Architecture and SysTems Research Lab (CASTL) of Florida State University for her help on the initial experimental setup and valuable suggestions related to this work. This work used the Extreme Science and Engineering Discovery Environment (XSEDE [58]), which is supported by National Science Foundation grant number ACI-1548562.

This work is supported in part by the National Science Foundation awards 1561041, 1564647, 1744336, 1763547, and 1952302, and has used the NoleLand facility funded by the National Science Foundation award CNS-1822737. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich *et al.*, "Going Deeper with Convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Cvpr, 2015.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Identity Mappings in Deep Residual Networks," *CoRR*, vol. abscaffe,/1603.05027, 2016. [Online]. Available: http://arxiv.org/abs/1603.05027

[3] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," *OpenAI Blog*, vol. 1, no. 8, p. 9, 2019.

[4] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 5998–6008. [Online]. Available: http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf

[5] J. Rasley, S. Rajbhandari, O. Ruwase, and Y. He, "Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 3505–3506.

[6] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *arXiv preprint arXiv:2005.14165*, 2020.

[7] W. Fedus, B. Zoph, and N. Shazeer, "Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity," *arXiv preprint arXiv:2101.03961*, 2021.

[8] A. Agarwal and J. C. Duchi, "Distributed delayed stochastic optimization," 2011.

[9] N. Strom, "Scalable distributed dnn training using commodity gpu cloud computing," in *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

[10] 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. Red Hook, NY, USA: Curran Associates Inc., 2013, p. 1223–1231.

[11] 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. USA: USENIX Association, 2014, p. 583–598.

[12] T. Chilimbi, Y. Suzue, J. Apacible, and K. Kalyanaraman, "Project adam: Building an efficient and scalable deep learning training system," in *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*. Broomfield, CO: USENIX Association, Oct. 2014, pp. 571–582. [Online]. Available: https://www.usenix.org/conference/osdi14/technical-sessions/presentation/chilimbi

[13] 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.

[14] Y. You, I. Gitman, and B. Ginsburg, "Scaling SGD batch size to 32k for Imagenet Training," *arXiv preprint arXiv:1708.03888*, 2017.

[15] M. Li, T. Zhang, Y. Chen, and A. J. Smola, "Efficient Mini-batch Training for Stochastic Optimization," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2014, pp. 661–670.

[16] A. Gibiansky and J. Hestness, "Baidu Research, TensorFlow-Allreduce," https://github.com/baidu-research/tensorflow-allreduce, 2017.

[17] A. A. Awan, J. Bédorf, C. Chu, H. Subramoni, and D. K. Panda, "Scalable distributed dnn training using tensorflow and cuda-aware mpi: Characterization, designs, and performance evaluation," in *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, May 2019, pp. 498–507.

[18] P. Jiang and G. Agrawal, "A linear speedup analysis of distributed deep learning with sparse and quantized communication," in *Advances in Neural Information Processing Systems*, 2018, pp. 2525–2536.

[19] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Advances in Neural Information Processing Systems 2*, D. S. Touretzky, Ed. Morgan-Kaufmann, 1990, pp. 598–605. [Online]. Available: http://papers.nips.cc/paper/250-optimal-brain-damage.pdf

[20] B. Hassibi and D. G. Stork, "Second order derivatives for network pruning: Optimal brain surgeon," in *Advances in Neural Information Processing Systems 5*, S. J. Hanson, J. D. Cowan, and C. L. Giles, Eds. Morgan-Kaufmann, 1993, pp. 164–171. [Online]. Available: http://papers.nips.cc/paper/647-second-order-derivatives-for-network-pruning-optimal-brain-surgeon.pdf

[21] Y. Guo, A. Yao, and Y. Chen, "Dynamic network surgery for efficient dnns," in *Advances in Neural Information Processing Systems 29*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds. Curran Associates, Inc., 2016, pp. 1379–1387. [Online]. Available: http://papers.nips.cc/paper/6165-dynamic-network-surgery-for-efficient-dnns.pdf

[22] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2016. [Online]. Available: http://arxiv.org/abs/1510.00149

[23] 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.

[24] 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.

[25] S. P. Karimireddy, Q. Rebjock, S. U. Stich, and M. Jaggi, "Error feedback fixes signsgd and other gradient compression schemes," *arXiv preprint arXiv:1901.09847*, 2019.

[26] J. Bernstein, J. Zhao, K. Azizzadenesheli, and A. Anandkumar, "signsgd with majority vote is communication efficient and fault tolerant," 2018.

[27] D. Alistarh, T. Hoefler, M. Johansson, N. Konstantinov, S. Khirirat, and C. Renggli, "The convergence of sparsified gradient methods," in *Advances in Neural Information Processing Systems*, 2018, pp. 5973–5983.

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

[29] 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. [Online]. Available: https://www.aclweb.org/anthology/D17-1045

[30] S. U. Stich, J.-B. Cordonnier, and M. Jaggi, "Sparsified sgd with memory," in *Advances in Neural Information Processing Systems*, 2018, pp. 4447–4458.

[31] 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.

[32] A. Sergeev and M. Del Balso, "Horovod: fast and easy distributed deep learning in tensorflow," *arXiv preprint arXiv:1802.05799*, 2018.
[33] Adam Paszke and Sam Gross and Soumith Chintala and Gregory Chanan, "Tensors and Dynamic neural networks in Python with strong GPU acceleration," https://pytorch.org/.
[34] "IBM Spectrum MPI," https://www.ibm.com/products/spectrum-mpi, 2020.
[35] M. Technologies, "Hierarchical Collectives (HCOLL)," https://docs.mellanox.com/display/HPCXv27/HCOLL, 2018.
[36] U. Köster, T. Webb, X. Wang, M. Nassar, A. K. Bansal, W. Constable, O. Elibol, S. Gray, S. Hall, L. Hornof, A. Khosrowshahi, C. Kloss, R. J. Pai, and N. Rao, "Flexpoint: An adaptive numerical format for efficient training of deep neural networks," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 1742–1752.
[37] S. Narang, G. Diamos, E. Elsen, P. Micikevicius, J. Alben, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh *et al.*, "Mixed precision training," in *Proc. 6th Int. Conf. on Learning Representations (ICLR)*, 2018.
[38] X. Jia, S. Song, W. He, Y. Wang, H. Rong, F. Zhou, L. Xie, Z. Guo, Y. Yang, L. Yu *et al.*, "Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes," *arXiv preprint arXiv:1807.11205*, 2018.
[39] N. Wang, J. Choi, D. Brand, C.-Y. Chen, and K. Gopalakrishnan, "Training deep neural networks with 8-bit floating point numbers," in *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds. Curran Associates, Inc., 2018, pp. 7675–7684. [Online]. Available: http://papers.nips.cc/paper/7994-training-deep-neural-networks-with-8-bit-floating-point-numbers.pdf
[40] J. Choi, S. Venkataramani, V. Srinivasan, K. Gopalakrishnan, Z. Wang, and P. Chuang, "Accurate and efficient 2-bit quantized neural networks," in *The 2nd SysML Conference*, Palo Alto, CA, USA, 2019.
[41] J. Wu, W. Huang, J. Huang, and T. Zhang, "Error compensated quantized sgd and its applications to large-scale distributed optimization," *arXiv preprint arXiv:1806.08054*, 2018.
[42] H. Tang, X. Lian, T. Zhang, and J. Liu, "Doublesqueeze: Parallel stochastic gradient descent with double-pass error-compensated compression," *arXiv preprint arXiv:1905.05957*, 2019.
[43] 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.
[44] Y. Lin, S. Han, H. Mao, Y. Wang, and W. J. Dally, "Deep gradient compression: Reducing the communication bandwidth for distributed training," *arXiv preprint arXiv:1712.01887*, 2017.
[45] J. Wangni, J. Wang, J. Liu, and T. Zhang, "Gradient sparsification for communication-efficient distributed optimization," in *Advances in Neural Information Processing Systems*, 2018, pp. 1299–1309.
[46] 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*, 2019, pp. 3411–3417.
[47] C. Renggli, S. Ashkboos, M. Aghagolzadeh, D. Alistarh, and T. Hoefler, "Sparcml: High-performance sparse communication for machine learning," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3295500.3356222
[48] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 2010, pp. 249–256.
[49] "QSGD," https://github.com/epfml/sparsifiedSGD/blob/master/qsgd.py, 2018.
[50] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *arXiv preprint arXiv:1610.05492*, 2016.
[51] J. Chen, X. Pan, R. Monga, S. Bengio, and R. Jozefowicz, "Revisiting distributed synchronous sgd," *arXiv preprint arXiv:1604.00981*, 2016.
[52] "CUDA," https://developer.nvidia.com/cuda-10.1-download-archive-base, 2019.
[53] R. Thakur, R. Rabenseifner, and W. Gropp, "Optimization of collective communication operations in mpich," *The International Journal of High Performance Computing Applications*, vol. 19, no. 1, pp. 49–66, 2005.
[54] "Understanding top-k sparsification in distributed deep learning," https://github.com/hclhkbu/GaussianK-SGD, 2019.
[55] "K-selection implementation of torch topk," https://github.com/torch/torch7/pull/496, 2016.
[56] A. Shanbhag, H. Pirk, and S. Madden, "Efficient top-k query processing on massively parallel hardware," in *Proceedings of the 2018 International Conference on Management of Data*, ser. SIGMOD '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 1557–1570. [Online]. Available: https://doi.org/10.1145/3183713.3183735
[57] T. Ben-Nun, "Torsten hoe er. 2018. demystifying parallel and distributed deep learning: An in-depth concurrency analysis," *arXiv preprint arXiv:1802.09941*, 2018.
[58] J. Towns, T. Cockerill, M. Dahan, I. Foster, K. Gaither, A. Grimshaw, V. Hazlewood, S. Lathrop, D. Lifka, G. D. Peterson *et al.*, "Xsede: accelerating scientific discovery," *Computing in science & engineering*, vol. 16, no. 5, pp. 62–74, 2014.