Exploiting Simultaneous Communications to Accelerate Data Parallel Distributed Deep Learning

Shaohuai Shi[†], Xiaowen Chu[‡]*, Bo Li[†]
[†]Department of Computer Science and Engineering, The Hong Kong University of Science and Technology
[‡]Department of Computer Science, Hong Kong Baptist University
shaohuais@cse.ust.hk, chxw@comp.hkbu.edu.hk, bli@cse.ust.hk

***Abstract*—Synchronous stochastic gradient descent (S-SGD) with data parallelism is widely used for training deep learning (DL) models in distributed systems. A pipelined schedule of the computing and communication tasks of a DL training job is an effective scheme to hide some communication costs. In such pipelined S-SGD, tensor fusion (i.e., merging some consecutive layers' gradients for a single communication) is a key ingredient to improve communication efficiency. However, existing tensor fusion techniques schedule the communication tasks sequentially, which overlooks their independence nature. In this paper, we expand the design space of scheduling by exploiting simultaneous All-Reduce communications. Through theoretical analysis and experiments, we show that simultaneous All-Reduce communications can effectively improve the communication efficiency of small tensors. We formulate an optimization problem of minimizing the training iteration time, in which both tensor fusion and simultaneous communications are allowed. We develop an efficient optimal scheduling solution and implement the distributed training algorithm ASC-WFBP with Horovod and PyTorch. We conduct real-world experiments on an 8-node GPU cluster of 32 GPUs with 10Gbps Ethernet. Experimental results on four modern DNNs show that ASC-WFBP can achieve about $1.09\times - 2.48\times$ speedup over the baseline without tensor fusion, and $1.15\times - 1.35\times$ speedup over the state-of-the-art tensor fusion solution.**

***Index Terms*—Distributed Deep Learning; Communication-Efficient; Simultaneous Communications**

I. Introduction

Nowadays deep learning (DL) techniques are widely used in numerous artificial intelligence applications including computer vision, natural language processing, robotics, healthcare, and many others. However, training a large deep neural network (DNN) model with a large data set is very time-consuming. To reduce the training time, the data parallel synchronous stochastic gradient descent (S-SGD) and its variants are the main algorithms for training DNN models with multiple processors [1]–[4]. In S-SGD, P processors keep a consistent set of model parameters w_i at any iteration i, where processor p, $p = 1, 2, ..., P$, loads a local batch of data D_i^p and computes the gradients g_i^p with respect to the model parameters. At the end of each iteration, the model parameters are updated with the aggregation of the gradients $\frac{1}{P}\sum_{p=1}^{P} g_i^p$. The gradient aggregation requires extensive data communications between the processors through All-Reduce operations, or between the processors and parameter servers [5], which introduce extra time cost that may limit the system scalability. E.g., according to the latest MLPerf benchmark[1], the training time of ResNet-50 [6] using one Nvidia DGX-A100 server is 39.78 minutes. By using 230 servers connected by 200Gbps IB network, the training time is only reduced to 0.76 minutes (i.e., $52\times$ speedup), resulting in poor system scalability of 22.6%. Optimizing the communication performance is therefore of critical importance in distributed DL.

There are various algorithmic optimizations to address the communication challenge, such as stale synchronous parallel SGD [7,8] and asynchronous parallel SGD [9,10] which relax synchronization conditions, and communication compression techniques like gradient quantization [11,12] and sparsification [13]–[17] which reduce the communication traffic. However, these methods could lead to slower convergence or accuracy loss if the hyper-parameters were not well tuned [7,18]–[20].

On the other hand, pipelining the computing and communication tasks is a system-level optimization technique that has no side-effect on the training convergence or model accuracy, and hence is widely used in state-of-the-art distributed DL frameworks such as TensorFlow, PyTorch, and BytePS[2]. For example, the layered structure of many DNNs (such as Convolutional Neural Networks (CNNs) and transformer-based models) enables the gradient computations to be overlapped with the layer-wise gradient communications [21], which is called wait-free backpropagation (WFBP) [22]. In practice, many DNN layers are with a small number of parameters (e.g., the convolutional layers in CNNs), whose gradient communications suffer from low utilization of network bandwidth due to the small transmission size [23]. Recently, tensor fusion techniques have been proposed to merge some nearby gradient tensors to improve the communication efficiency, and overlap the communication tasks with either the backpropagation computing tasks [2,3,16,24], or the feed-forward computing tasks [4,25]–[27].

Although tensor fusion schemes can improve the communication efficiency to some extend, existing implementations schedule the All-Reduce tasks sequentially and overlook their independence nature [2]–[4]. To this end, we propose to use simultaneous All-Reduce communications as an alternative to

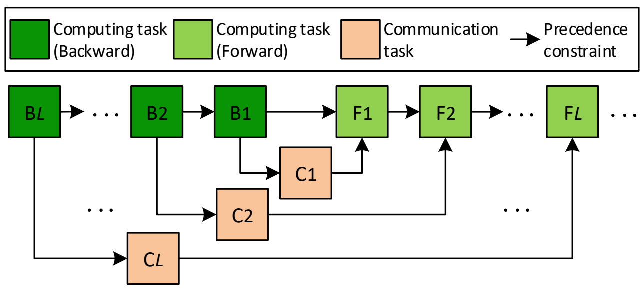
*Corresponding author.

[1]https://mlperf.org/training-results-0-7
[2]https://github.com/bytedance/byteps

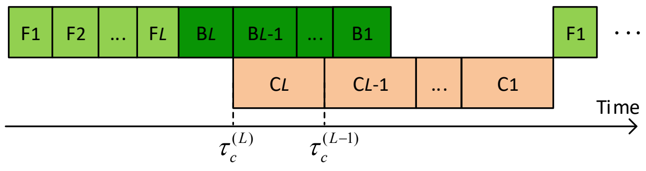
tensor fusion[3]. Our experimental results verify that communicating multiple tensors simultaneously can achieve much higher bandwidth utilization than communicating them sequentially. Therefore, we expand the scheduling design space to three possible methods: naive WFBP, WFBP with tensor fusion, and WFBP with simultaneous communications. To determine the proper communication method for specific tensors during the training process, we formulate an optimization problem of scheduling the communication tasks, targeting at minimizing the wall-clock training iteration time. The novelty of the optimization problem is the relaxed constraint that enables simultaneous All-Reduce communications. We develop an effective optimal solution with polynomial time complexity, based on which we implement an distributed training system named WFBP with Adaptive Simultaneous Communications (ASC-WFBP) atop the communication library Horovod [28]. Our contributions can be summarized as follows.

- We identify that sequentially scheduling the communication tasks in distributed DNN training is sub-optimal. Through theoretical analysis and real-world experiments, we show that simultaneous All-Reduce communications can improve the utilization of network bandwidth.
- We expand the design space of scheduling by considering both **tensor fusion** and **simultaneous communications**, and define a new scheduling problem for distributed DNN training. We develop an optimal solution with polynomial time complexity.
- We conduct real-world experiments on an 8-node GPU cluster with 32 Nvidia RTX2080Ti GPUs connected with 10Gbps Ethernet. Experimental results show that our method achieves $15\% - 35\%$ improvement over the state-of-the-art tensor fusion solution.

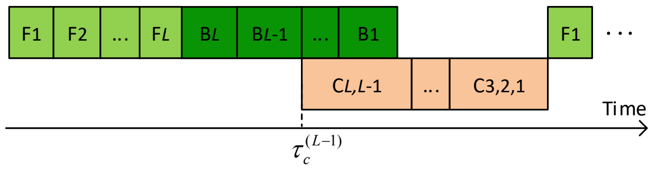
The rest of the paper is organized as follows. We first present some preliminaries related to the distributed training of DNNs in Section II. We discuss the idea of simultaneous communications in Section III. Then we formulate the scheduling problem in Section IV, followed with our proposed optimal solution in Section V. The experimental studies are demonstrated in Section VI. In Section VII, we introduce some related work. We conclude this paper in Section VIII.



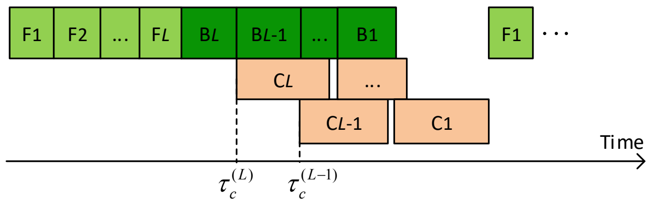
(a) DAG of computing and communication tasks.



(b) WFBP.



(c) WFBP with Tensor Fusion: MG-WFBP.



(d) WFBP with Simultaneous Communications.

Fig. 1: (a) The DAG of computing and communication tasks, and (b-d) the timeline of S-SGD algorithms with different schedules. (b) WFBP: Gradient communication of each layer begins after that layer's gradients are calculated and its previous layer's communication is finished. (c) MG-WFBP: Nearby gradients are merged in WFBP. (d) WFBP with two-way simultaneous communications.

II. Preliminaries

In this section, we present the preliminaries of mini-batch SGD, S-SGD with WFBP, and S-SGD with tensor fusion (e.g., MG-WFBP). For ease of presentation, we summarize the frequently used notations in Table I.

A. Mini-batch SGD

With a single processor, mini-batch SGD is currently the most popular training algorithm in DL, which iteratively updates the model parameters with their first-order gradients estimated from a batch of training data. Mini-batch SGD contains several steps at each iteration with the current state of model parameters w_i. For the i^{th} iteration, it first loads a mini-batch of training samples X_i. Second, it performs the feed-forward computation from the first layer towards the last layer to calculate the loss $\mathcal{L}(w_i, X_i)$. Third, it calculates the gradients $g_i = \nabla\mathcal{L}(w_i, X_i)$ with respect to the model parameters by the backpropagation algorithm (i.e., from the last layer to the first layer). Finally, it updates the model parameters using

$$w_{i+1} = w_i - \eta\nabla\mathcal{L}(w_i, X_i), \tag{1}$$

where η is the learning rate.

B. Synchronous SGD with WFBP

With multiple processors, Synchronous SGD (S-SGD) with data parallelism is widely used to accelerate the training

[3]In this paper, we mainly focus on the All-Reduce framework as it is widely used in high-performance distributed training according to the MLPerf submissions. It is possible to apply the idea to Parameter Server as well.

TABLE I: Notations

Name	Description
P	The number of workers (or GPUs) in the cluster.
a	Startup time of an All-Reduce operation.
b	Transmission and aggregation time per byte of All-Reduce.
$m^{(l)}$	The gradient size in bytes of layer l.
L	The number of learnable layers (or gradient tensors) of a DNN.
t_f	Feed-forward computation time in each iteration.
t_b	Backward computation time in each iteration.
$t_b^{(l)}$	Gradient calculation time of layer l in each iteration.
$\tau_b^{(l)}$	The timestamp when layer l begins to calculate gradients.
t_c	Gradient aggregation time in each iteration.
$t_c^{(l)}$	Gradient aggregation time of layer l in each iteration.
$\tau_c^{(l)}$	The timestamp when layer l begins to aggregate gradients.

process. S-SGD has the same convergence property with mini-batch SGD, as it distributes the training data X_i to P workers with X_i^p, where $p = 1, 2, ..., P$, and all workers keep the consistent model parameters during the whole training process. The update rule of S-SGD is

$$w_{i+1} = w_i - \eta \frac{1}{P} \sum_{p=1}^{P} \nabla \mathcal{L}(w_i, X_i^p). \tag{2}$$

The aggregation of distributed gradients introduces the communication overhead which could limit the system scalability. The training iteration time includes the feed-forward computation time, backpropagation computation time, and the communication time of gradient aggregation. These computing and communication tasks can be modeled by a directed acyclic graph (DAG) as shown in Fig. 1(a). By exploiting the layered structure of many modern DNNs, the gradient communications can be overlapped with gradient computations during backpropagation (i.e., WFBP). In S-SGD with WFBP, once the gradients of layer l have been calculated, the corresponding tensor(s) can be immediately ready for communication through an All-Reduce operation. At the same time, the gradient computation of layer $l-1$ can also be performed, since it is independent of the communication of layer l's gradients. As shown in Fig. 1(b), the overlapping between communications and computations can hide some communication cost and reduce the overall iteration time.

C. S-SGD with Tensor Fusion

The layer-wise communications using All-Reduce may suffer from the high-latency problem because a tensor will be further split into multiple messages in the All-Reduce operation, which generally results in a very low network bandwidth utilization [3,4]. Specifically, the time cost of an All-Reduce operation for aggregating messages with size m (in bytes) can be modeled as

$$t_{AR}(m) = a + b \times m, \tag{3}$$

where a is the startup overhead which is related to the link communication latency, network topology, and All-Reduce algorithm, and b is the transmission time per byte [3]. Therefore, the overall communication time of an L-layer DNN with WFBP is

$$t_c^{WFBP} = \sum_{l=1}^{L} t_{AR}(m^{(l)}) = aL + b\sum_{l=1}^{L} m^{(l)}, \tag{4}$$

where $m^{(l)}$ is the gradient size of layer l. The gradients of some consecutive layers can be merged as a large tensor to be All-Reduced together, as shown in Fig. 1(c), which reduces the number of startup overhead a. Since merging gradients should wait for the completion of previous gradient computation, not all layers need to be merged. According to [3], only the layers that satisfy the condition

$$\tau_b^{(l-l)} + t_b^{(l-1)} < \tau_c^{(l)} + a, \tag{5}$$

should be merged to reduce the iteration time. If layer l is merged to layer $l-1$, the saved time is

$$t_{gain}^{(l)} = a - \max\{0, \tau_b^{(l-l)} + t_b^{(l-1)} - \tau_c^{(l)}\}. \tag{6}$$

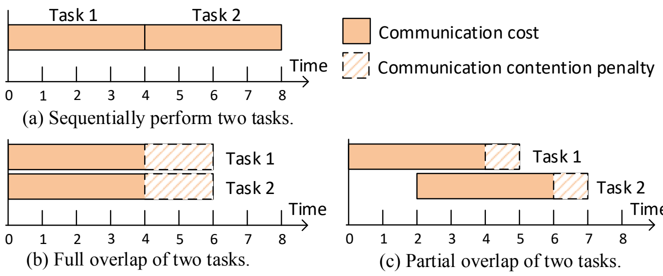

Task 1
Task 2
Time
0 1 2 3 4 5 6 7 8
Communication cost
Communication contention penalty
(a) Sequentially perform two tasks.
Task 1
Task 2
Time
0 1 2 3 4 5 6 7 8
(b) Full overlap of two tasks.
Task 1
Task 2
Time
0 1 2 3 4 5 6 7 8
(c) Partial overlap of two tasks.

Fig. 2: Three communication schedules for two communication tasks.

III. Simultaneous Communications

The purpose of tensor fusion is to better utilize the network bandwidth by reducing the impact of startup overhead. We notice that an alternative to improving bandwidth utilization is to simultaneously invoke multiple communication tasks [29,30]. We use two-way simultaneous communications as an example to illustrate the idea.

If there is only one All-Reduce operation with message size m (as shown in Fig. 2(a)), the effective throughput is

$$T_{AR}^1 = \frac{m}{a + bm}. \tag{7}$$

Thus, sequentially performing two All-Reduce operations takes the communication time of

$$t_a = t_{AR}(m) + t_{AR}(m) = 2a + 2bm. \tag{8}$$

If we invoke two All-Reduce operations simultaneously (as shown in Fig. 2(b)), each All-Reduce task can only use part of the bandwidth. Assume the average transmission time per byte is increased to γb, where $\gamma > 1$ is used to model the contention of simultaneous communications. Our extensive experiments on ring-based All-Reduce show that $1 < \gamma \leq 2$ for a large spectrum of m in practice. Thus the overall throughput of two simultaneous communications is

$$T_{AR}^2 = \frac{2m}{a + \gamma bm}. \quad (9)$$

Obviously, we have

$$T_{AR}^2 > T_{AR}^1, \quad (10)$$

which indicates that the overall communication throughput can be improved by simultaneous communications. The improvement comes from two factors: (1) the startup overhead is reduced from $2a$ to a; (2) the bandwidth utilization is improved by multiplexing multiple communications. Still, the communication time of each All-Reduce operation is increased. We call the extra communication time for each All-Reduce operation as **penalty**, i.e.,

$$t_{penalty}^{(1,2)} = a + \gamma bm - (a + bm) = (\gamma - 1)bm. \quad (11)$$

In practice, the two tensors may not arrive at the same time; hence they are likely to be partially overlapped, as shown in Fig. 2(c). So the duration of communication contention is shorter than that of fully overlapped case, and the penalty is also smaller. For example, in the case of Fig. 2(c), only half of the message of Task 1 is overlapped with Task 2, so the penalty is equal to $(\gamma - 1)bm/2$.

Without loss of generality, assume that the tensor sizes of Task 1 and Task 2 are $m^{(1)}$ and $m^{(2)}$ respectively, and the arriving time of Task 2, denoted by $\tau_c^{(2)}$, is in the duration of Task 1's communication, i.e., $\tau_c^{(1)} \leq \tau_c^{(2)} \leq \tau_c^{(1)} + t_c^{(1)}$, where $\tau_c^{(1)}$ and $t_c^{(1)}$ are the beginning timestamp and the communication time of Task 1 respectively. Then, we can calculate the overlapped message size by

$$m = m^{(1)} - \frac{(\tau_c^{(2)} - \tau_c^{(1)})m^{(1)}}{a + bm^{(1)}}. \quad (12)$$

Thus, the penalty is

$$t_{penalty}^{(1,2)} = (\gamma - 1)bm = (\gamma - 1)bm^{(1)}(1 - \frac{\tau_c^{(2)} - \tau_c^{(1)}}{a + bm^{(1)}}). \quad (13)$$

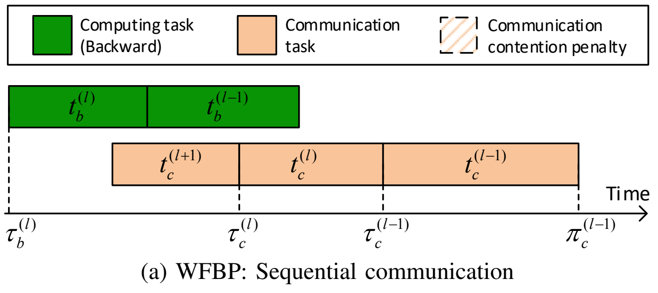
IV. Problem Formulation

We demonstrate three possible schedules of two consecutive layers in Fig. 3: WFBP, merging gradients (MG), and simultaneous communications (SC). We want to determine which method can achieve the shortest completion time. Let π_{WFBP}, π_{MG}, and π_{SC} denote the completion time using WFBP, MG, and SC respectively. We should also assume

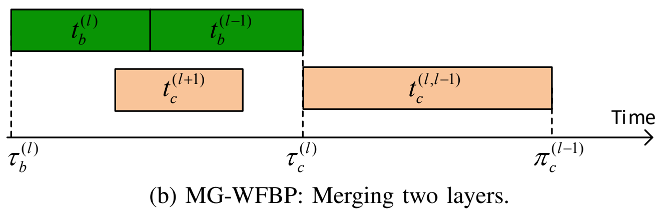
$$\tau_c^{(l)} + t_c^{(l)} > \tau_b^{(l-1)} + t_b^{(l-1)}, \quad (14)$$

because if the above inequality does not hold, the communication of layer l can be fully overlapped by the computation of layer $l-1$, which should be scheduled as WFBP and it is obviously better than MG.

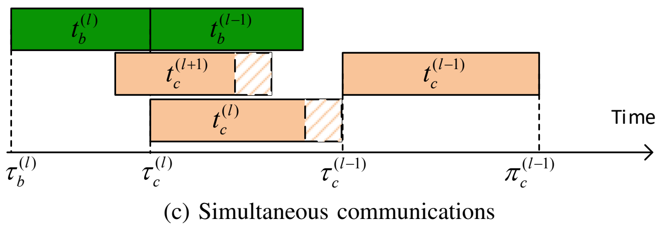
In WFBP (Fig. 3(a)), since there is at most one communication task being executed, the communication of layer l can begin only after the completion of both gradient computation of layer l and communication of layer $l+1$. Thus,



(a) WFBP: Sequential communication



(b) MG-WFBP: Merging two layers.



(c) Simultaneous communications

Fig. 3: Three possible schedules for two consecutive layers, where the second layers' ready time is behind the first layer's.

$$\begin{aligned}\pi_{WFBP} &= \tau_c^{(l-1)} + t_c^{(l-1)} = \tau_c^{(l)} + t_c^{(l)} + t_c^{(l-1)} \\ &= \tau_c^{(l)} + t_{AR}(m^{(l)}) + t_{AR}(m^{(l-1)}) \\ &= \tau_c^{(l)} + 2a + b(m^{(l)} + m^{(l-1)}). \end{aligned} \quad (15)$$

In MG (Fig. 3(b)), we merge the two nearby gradients (layer l and layer $l-1$) to a single tensor. The communication of this merged tensor should be after the completion of the gradient computation of layer $l-1$. Thus,

$$\begin{aligned}\pi_{MG} &= \max\{\tau_c^{(l)}, \tau_b^{(l-1)} + t_b^{(l-1)}\} + t_c^{(l,l-1)} \\ &= \max\{\tau_c^{(l)}, \tau_b^{(l-1)} + t_b^{(l-1)}\} + t_{AR}(m^{(l)} + m^{(l-1)}) \\ &= \max\{\tau_c^{(l)}, \tau_b^{(l-1)} + t_b^{(l-1)}\} + a + b(m^{(l)} + m^{(l-1)}). \end{aligned} \quad (16)$$

In SC (Fig. 3(c)), the communication of layer l can be immediately started after the gradients have been calculated, which means that there are two simultaneous communications during the communication period of layer l. Thus, we have

$$\begin{aligned}\pi_{SC} = &\max\{\tau_b^{(l)} + t_b^{(l)} + t_c^{(l)} + t_{penalty}^{(l+1,l)}, \tau_b^{(l-1)} + t_b^{(l-1)}\} \\ &+ t_c^{(l-1)}, \end{aligned} \quad (17)$$

where $t_{penalty}^{(l+1,l)}$ can be derived by Eq. (13).

Through the above discussion, we can quantitatively compare π_{WFBP}, π_{MG}, and π_{SC} to determine which one is the best communication schedule for layer l. We first define the following terminologies.

Definition 1. *(Normal layer). If a layer l is originally scheduled with WFBP to communicate its gradients, then it is a normal layer and its gradients are communicated only after the completion of the backward computation of layer l and the completion of the communication of layer $l+1$.*

Definition 2. *(Merged-gradient layer [3,24]). A layer l is called a merged-gradient (MG) layer if at the timestamp of $\tau_c^{(l)}$, the gradients of that layer are merged to layer $l-1$ to be communicated together.*

Definition 3. *(SC layer). A layer l is called a simultaneous-communications (SC) layer if at the timestamp of $\tau_c^{(l)}$, the communication task of layer $l+1$ has not been completed, and the gradient communication of layer l is immediately executed without waiting for the completion of layer $(l+1)$'s communication.*

We use l_n, l_m, and l_s to denote the type of normal, MG, and SC layer respectively. Let the variable $e^{(l)}$ denote the type of layer l. We have $e^{(l)} \in \{l_n, l_m, l_s\}$. According to the definitions, layer L should be a normal layer or an MG layer, i.e., $e^{(L)} \in \{l_n, l_m\}$, and layer 1 should always be the SC layer, i.e., $e^{(1)} = l_s$. For a given L-layer DNN, the problem is how to determine each layer's type so as to minimize the iteration time. We use $\mathbb{S}$ to denote the set of possible combinations of layer types:

$$\mathbb{S} = \{[e^{(l)}] | e^{(l)} \in \{l_n, l_m, l_s\} \text{ and } 1 \leq l \leq L\}. \tag{18}$$

As we focus on analyzing the elapsed time of one iteration, we can assume the beginning time of layer 1's feed-forward computation is zero, i.e., $\tau_f^{(1)} = 0$. Thus, we can formulate the scheduling problem as follows:

For an L-layer DNN training job on P workers, determine $e^{(l)}$ for each l such that the iteration time is minimal, i.e.,

$$\text{minimize: } \tau_c^{(1)} + t_c^{(1)}, \tag{19}$$

$$\text{s.t. } \boldsymbol{s} := [e^{(l)}] \in \mathbb{S}. \tag{20}$$

V. Optimal Solution

In this section, we first analyze how to determine the layer type to achieve performance gain (i.e., shorter time) under different conditions. Then we develop a theorem to optimally determine each layer's type, followed with the distributed training algorithm with an optimal schedule.

A. Theoretical Analysis

We first discuss under different conditions, which one is the smallest among π_{WFBP}, π_{MG}, and π_{SC}.

As we discussed in Section IV, if Eq. (14) is true, then the communication of layer l can be fully overlapped with the computation of layer $l-1$, which means layer $l-1$ should be a normal layer. Thus, we define a condition for setting layer l to a normal layer:

$$\textbf{Q1: } \tau_c^{(l)} + t_c^{(l)} \leq \tau_b^{(l-1)} + t_b^{(l-1)}. \tag{21}$$

Formally,

$$\textbf{Q1} \implies \min\{\pi_{WFBP}, \pi_{MG}, \pi_{SC}\} = \pi_{WFBP}. \tag{22}$$

Next, we should solve $\min\{\pi_{WFBP}, \pi_{MG}, \pi_{SC}\}$ if **Q1** is false. We first prove that $\pi_{SC} < \pi_{WFBP}$ if **Q1** is false. As in SC, there are some overlaps between the communication tasks of layer l and layer $l-1$, which means the network throughput in the overlapping part is larger than WFBP which executes sequentially according to Eq. (10). In both SC and WFBP, the communication traffic is the same, thus the higher throughput should have a shorter time, which proves that $\pi_{SC} < \pi_{WFBP}$ holds if **Q1** is false. Formally,

$$\neg\textbf{Q1} \implies \min\{\pi_{WFBP}, \pi_{SC}\} = \pi_{SC}. \tag{23}$$

We further solve $\min\{\pi_{MG}, \pi_{SC}\}$ when **Q1** is false. We use a conclusion from [3,24] which says that if and only if

$$\textbf{Q2: } \tau_b^{(l-1)} + t_b^{(l-1)} < \tau_c^{(l)} + a \tag{24}$$

is true, then $\pi_{MG} < \pi_{WFBP}$. Thus, we have

$$\neg\textbf{Q2} \implies \min\{\pi_{WFBP}, \pi_{MG}\} = \pi_{WFBP}, \tag{25}$$

and

$$\neg\textbf{Q1} \wedge \neg\textbf{Q2} \implies \min\{\pi_{WFBP}, \pi_{MG}, \pi_{SC}\} = \pi_{SC}. \tag{26}$$

We decouple the two max functions in π_{SC} and π_{MG}. Firstly, we use a new condition

$$\textbf{Q3: } \tau_c^{(l)} + t_b^{(l)} + t_c^{(l)} + t_{penalty}^{(l+1,l)} < \tau_b^{(l-1)} + t_b^{(l-1)} \tag{27}$$

to decompose the max function in π_{SC}. If **Q3** is true, then simultaneous communications of layer l and $l+1$ can be fully hidden by the gradient computation of layer $l-1$, which is obviously better than MG. Thus, we have

$$\neg\textbf{Q1} \wedge \textbf{Q2} \wedge \textbf{Q3} \implies \min\{\pi_{WFBP}, \pi_{MG}, \pi_{SC}\} = \pi_{SC}. \tag{28}$$

If **Q3** is false, then

$$\pi_{SC} = \tau_b^{(l)} + t_b^{(l)} + t_c^{(l)} + t_{penalty}^{(l+1,l)} + t_c^{(l-1)}. \tag{29}$$

We further decompose π_{MG} by introducing a condition

$$\textbf{Q4: } \tau_c^{(l)} < \tau_b^{(l-1)} + t_b^{(l-1)}. \tag{30}$$

If **Q4** is true, then

$$\begin{aligned}\pi_{MG} &= \max\{\tau_c^{(l)}, \tau_b^{(l-1)} + t_b^{(l-1)}\} + a + b(m^{(l)} + m^{(l-1)}) \\ &= \tau_b^{(l)} + t_b^{(l)} + a + b(m^{(l)} + m^{(l-1)}).\end{aligned} \tag{31}$$

Thus, we have

$$\begin{aligned}\pi_{MG} - \pi_{SC} =& \tau_b^{(l)} + t_b^{(l)} + a + b(m^{(l)} + m^{(l-1)}) - (\tau_b^{(l)} + \\ & t_b^{(l)} + a + bm^{(l)} + t_{penalty}^{(l+1,l)} + a + bm^{(l-1)}) \\ =& -a - t_{penalty}^{(l+1,l)} < 0,\end{aligned} \tag{32}$$

which means $\min\{\pi_{MG}, \pi_{SC}\} = \pi_{MG}$. Therefore, we conclude that

$$\neg\textbf{Q1} \wedge \textbf{Q2} \wedge \neg\textbf{Q3} \wedge \textbf{Q4} \implies \min\{\pi_{MG}, \pi_{SC}\} = \pi_{MG}. \tag{33}$$

On the other hand, if **Q4** is false, it means that both layers are ready for communications at the timestamp of $\tau_c^{(l-1)}$, then

$$\pi_{MG} = \tau_c^{(l)} + a + b(m^{(l)} + m^{(l-1)}), \tag{34}$$

which indicates that both messages have a network throughput of

$$T_{MG} = \frac{m^{(l)} + m^{(l-1)}}{a + b(m^{(l)} + m^{(l-1)})}. \tag{35}$$

However, SC has a partial overlap between layer $l+1$ and layer l, so the overall network throughput of simultaneous communications T_{SC} should be smaller than T_{MG}. Therefore, we have

$$\neg\mathbf{Q1} \wedge \mathbf{Q2} \wedge \neg\mathbf{Q3} \wedge \neg\mathbf{Q4} \implies \min\{\pi_{MG}, \pi_{SC}\} = \pi_{MG} \tag{36}$$

According to the above analysis, we conclude that the assertions of (22)(26)(28)(33)(36) can achieve shorter time for any layer l, which can be summarized as follows

$$\mathbf{Q1} \implies e^{(l)} = l_n. \tag{37}$$

$$\neg\mathbf{Q1} \wedge \mathbf{Q2} \wedge \neg\mathbf{Q3} \implies e^{(l)} = l_m. \tag{38}$$

$$(\neg\mathbf{Q1} \wedge \neg\mathbf{Q2}) \vee (\neg\mathbf{Q1} \wedge \mathbf{Q2} \wedge \mathbf{Q3}) \implies e^{(l)} = l_s. \tag{39}$$

Note that assertion (38) can be derived by combining (33) and (36). Now we can conclude the above results as the following theorem.

Theorem 1. *Given an L-layer DNN to be trained with distributed SGD using the All-Reduce collective for communication on a P-worker cluster, we can set the type of layer l ($1 \le l \le L$) according to the following equation to achieve optimal overlapping between gradient communications and backpropagation computations:*

$$e^{(l)} = \begin{cases} l_n, & \textbf{\textit{Q1}} \text{ is true;} \\ l_m, & \neg\textbf{\textit{Q1}} \wedge \textbf{\textit{Q2}} \wedge \neg\textbf{\textit{Q3}} \text{ is true;} \\ l_s, & \text{otherwise.} \end{cases} \tag{40}$$

Proof. According to the analysis of any layer l under the assertions of (37), (38), and (39), any violation of these assertions would bring longer time of layer l's communication, and thus a longer iteration time. Therefore, by setting the layer type by Eq. (40), any changes in the layer type cannot achieve a shorter iteration time, which concludes the proof. □

B. Algorithm

According to Theorem 1, we derive the algorithm to determine the layer types for a given DNN assuming that the computation time of each layer is known. In practice, we can simply profile each layer's computation time with several iterations before the training process. The pseudo-code of the algorithm is shown in Algorithm 1.

In Algorithm 1, the procedure CALCCOMMTIME (Lines 20-36) is to calculate the layers' communication time and their beginning time for invoking the communication according to their types. The algorithm first (Line 1) initializes the output to the default type of normal layer, and then calculates the beginning time of backpropagation computation of each layer (Lines 2-5). In Line 6, it calculates the layers' communication time and their beginning time according to their initialized types. Lines 7-16 determine the layer types according to Theorem 1. In Line 17, the layers' communication time and their beginning time should be refreshed after the layer type has changed from the normal layer to the MG or SC layer.

Algorithm 1 has a time complexity of $O(L^2)$. In the loop of Line 7, each time of changing the layer type requires to re-calculate the communication time with the procedure CALCCOMMTIME which has another loop in Line 23. Therefore, the total time complexity of the algorithm is $O(L^2)$.

With the solution derived by Algorithm 1, we can use the layer type during the training process to invoke the communication task at a proper time. The training algorithm with layer types is shown in Algorithm 2 by using a similar framework in [16]. During the asynchronous communication, there are multiple communication streams that can be used to invoke the All-Reduce operations simultaneously. The communication

Algorithm 1 DetermineLayerType

```
Input: $a$, $b$, $\gamma$, $L$, $\boldsymbol{t}_b = [1...L]$, $\boldsymbol{m} = [m^{(1)}, m^{(2)}, ..., m^{(L)}]$.
Output: $\boldsymbol{s}$
 1: $\boldsymbol{s}[1...L] = l_n$;                          ▷ Initialize as normal layers.
 2: Initialize $\boldsymbol{\tau}_b$;              ▷ Layer-wise computation beginning time.
 3: $\boldsymbol{\tau}_b[L] = 0$;
 4: for $l = L-1 \to 2$ do
 5:     $\boldsymbol{\tau}_b[l] = \boldsymbol{\tau}_b[l+1] + \boldsymbol{t}_b[l+1]$;
 6: $\boldsymbol{t}_c$, $\boldsymbol{\tau}_c$ = CALCCOMMTIME($\boldsymbol{t}_b, \boldsymbol{\tau}_b, \boldsymbol{s}, \boldsymbol{m}, L$);
 7: for $l = L \to 2$ do
 8:     $Q1 = \boldsymbol{\tau}_c[l] + \boldsymbol{t}_c[l] \le \boldsymbol{\tau}_b[l] + \boldsymbol{t}_b[l-1]$;
 9:     if $Q1$ then $\boldsymbol{s}[l] = l_n$;
10:     else
11:         $Q2 = \boldsymbol{\tau}_b[l-1] + \boldsymbol{t}_b[l-1] < \boldsymbol{\tau}_c[l] + a$;
12:         $Q3 = \boldsymbol{\tau}_c[l] + \boldsymbol{t}_b[l] + \boldsymbol{t}_c[l] + (\gamma-1)b\boldsymbol{m}[l+1](1 - \frac{\boldsymbol{\tau}_c[l]-\boldsymbol{\tau}_c[l+1]}{a+b\boldsymbol{m}[l+1]}) < \boldsymbol{\tau}_b[l-1] + \boldsymbol{t}_b[l-1]$;
13:         if $Q2 \wedge \neg Q3$ then
14:             $\boldsymbol{s}[l] = l_m$;
15:         else
16:             $\boldsymbol{s}[l] = l_s$;
17:         $\boldsymbol{t}_c$, $\boldsymbol{\tau}_c$ = CALCCOMMTIME($\boldsymbol{t}_b, \boldsymbol{\tau}_b, \boldsymbol{s}, \boldsymbol{m}, L$);
18: $\boldsymbol{s}[1] = l_s$;
19: Return $\boldsymbol{s}$;
20: procedure CALCCOMMTIME($\boldsymbol{t}_b, \boldsymbol{\tau}_b, \boldsymbol{s}, \boldsymbol{m}, L$)
21:     Initialize $\boldsymbol{t}_c$;                  ▷ Layer-wise communication time
22:     Initialize $\boldsymbol{\tau}_c$;    ▷ Layer-wise communication beginning time
23:     for $l = L \to 1$ do
24:         $e = \boldsymbol{s}[l]$;
25:         if $e == l_n$ then                          ▷ The normal layer
26:             $\boldsymbol{\tau}_c[l] = \boldsymbol{\tau}_b[l] + \boldsymbol{t}_b[l]$;
27:             $\boldsymbol{t}_c[l] = a + b \times \boldsymbol{m}[l]$;
28:         else if $e == l_m$ then                        ▷ The MG layer
29:             $\boldsymbol{m}[l-1] = \boldsymbol{m}[l-1] + \boldsymbol{m}[l]$;
30:             $\boldsymbol{m}[l] = 0$;
31:             $\boldsymbol{\tau}_c[l-1] = \max\{\boldsymbol{\tau}_c[l], \boldsymbol{\tau}_b[l-1] + \boldsymbol{t}_b[l-1]\}$;
32:             $\boldsymbol{t}_c[l-1] = a + b \times \boldsymbol{m}[l-1]$;
33:         else                                          ▷ The SC layer
34:             $\boldsymbol{\tau}_c[l] = \boldsymbol{\tau}_b[l] + \boldsymbol{t}_b[l]$;
35:             $\boldsymbol{t}_c[l] = a + b\boldsymbol{m}[l] + (\gamma-1)b\boldsymbol{m}[l+1](1 - \frac{\boldsymbol{\tau}_c[l]-\boldsymbol{\tau}_c[l+1]}{a+b\boldsymbol{m}[l+1]})$;
36:     Return $\boldsymbol{t}_c$, $\boldsymbol{\tau}_c$;
```

is invoked according to the layer type as in Line 21, where it indicates that the normal layer and the SC layer can be immediately communicated with other workers, while the MG layer should merge its gradients with the previous layer.

Algorithm 2 ASC-WFBP at worker p

```
Input: $\boldsymbol{D} = [\{X_1, y_1\}, ..., \{X_n, y_n\}]$, $net$
 1: Initialize a shared and synchronized queue $Q$;
 2: Get $\boldsymbol{s}$ by calling Algorithm 1.
 3: AsyncCommunication($Q, \boldsymbol{m}$);
 4: while not stop do
 5:     Sample a mini-batch of $data$ from $\boldsymbol{D}$;
 6:     AsyncComputation($data, net, Q, \boldsymbol{s}$);
 7:     WaitForLastCommunicationFinished();
 8:     $net.W = net.W - \eta \cdot \nabla net.W$,
 9: procedure AsyncComputation($data, net, Q, \boldsymbol{s}$)
10:     $o = d$;
11:     for $l = 1 \to L$ do
12:         $o$=FeedForward($l, o$);
13:     for $l = L \to 1$ do
14:         BackwardPropagation($l$);
15:         $Q$.push($l$);
16: procedure AsyncCommunication($Q, \boldsymbol{s}$)
17:     Initialize $gb$; //gradient buffer
18:     while isRunning do
19:         $l = Q$.pop();
20:         if $\boldsymbol{s}[l] == l_n$ or $\boldsymbol{s}[l] == l_m$ then
21:             AsyncAllReduce($gb$);
22:             Clear $gb$;
23:         else if $\boldsymbol{s}[l] == l_m$ then
24:             $gb$.push($l$);
25:         if $l == 1$ then
26:             SynchronizeAllReduceOperations();
27:             NotifyLastCommunicationFinished();
```

VI. Experimental Studies

In this section, we first evaluate the parameters of a and b in Eq. (3) on our testbed. Then we measure the contention model of two simultaneous communications to verify the overall throughput of Eq. (9). After that we present the iteration time of WFBP [22], MG-WFBP[4] [3], and our new ASC-WFBP to show the effectiveness of our solution. Finally, we discuss the experimental results.

A. Experimental Settings

Cluster Configuration. Our testbed is a 8-node GPU cluster, each of which installs 4 Nvidia RTX2080Ti GPUs connected with PCIe3.0x16. Each node has a network interface card of 10Gbps Ethernet (10GbE). The details of the node configuration are shown in Table II. The common libraries related to communication performance are OpenMPI-4.0.1, NCCL-2.4.7, and Horovod-1.9.2. The DL framework is PyTorch-1.4.0 with cuDNN-7.6 and CUDA-10.1.

Benchmark DNN Models. We measure the performance with four modern DNN models from two widely used applications: image classification with CNNs and natural language processing with Transformer. For CNNs, we choose ResNet-152 [6], DenseNet-201 [31] and Inception-v4 [32], and for Transformer, we exploit BERT [33] with the relative small-size model BERT-Base. The details of the models are shown in Table III, where the batch size indicates the number of samples used on each GPU at each iteration. The input size in CNNs is the resolution of input images, while in Transformer is the sequence length of input sentences.

[4]https://github.com/HKBU-HPML/MG-WFBP

TABLE II: The server configuration.

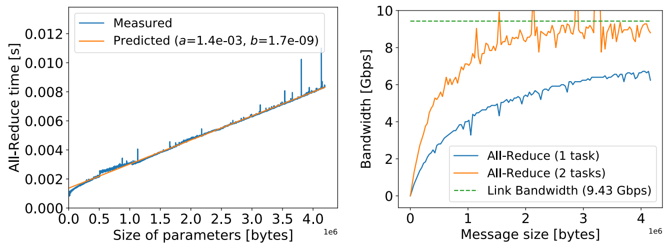
Name	Model
CPU	Dual Intel(R) Xeon(R) Gold 6230 CPU @ 2.10GHz
GPU	Nvidia RTX2080Ti (@1.35GHz and 11GB Memory)
Memory	512GB DDR4
OS	Ubuntu-16.04

TABLE III: DNN details for experiments.

Model	# Parameters (million)	# Gradient Tensors	Batch Size	Input Size
ResNet-152	~ 60	467	32	$3 \times 224 \times 224$
DenseNet-201	~ 20	604	32	$3 \times 224 \times 224$
Inception-v4	~ 42	449	32	$3 \times 224 \times 224$
BERT-Base	~ 110	206	64	64

B. Communication Models

We estimate the cluster related parameters: a and b in Eq. (3) using the benchmark tool of nccl-tests[5] by measuring All-Reduce on a range of message sizes in [8K, 4M]. The results are shown in Fig. 4(a), which indicates that the measured time cost of All-Reduce is almost linear to the message size as the predicted model of Eq. (3), where $a = 1.4 \times 10^{-3}$ and $b = 1.7 \times 10^{-9}$ on the 8-node GPU cluster.



(a) Elapsed time of All-Reduce. (b) Bandwidth of All-Reduce.

Fig. 4: Time cost of All-Reduce on the 8-node cluster.

For the communication contention, we measure the overall throughput by invoking two communications simultaneously compared to that of only one communication occupying the link resource on the 8-node cluster. The results are shown in Fig. 4(b). The link bandwidth is measured by iperf3[6] with the parallel mode enabled. We can see that a single All-Reduce operation has a very low bandwidth utilization, while two simultaneous All-Reduce operations can significantly improve

[5]https://github.com/NVIDIA/nccl-tests
[6]https://iperf.fr/

the bandwidth utilization so that it can achieve nearly optimal bandwidth when the message size is greater than 2MB. According to the results in Fig. 4(b), the transmission speed penalty term $\gamma \approx 1.5$ in our cluster. We further compare the measured All-Reduce time with the simple analytical model for different message sizes in Fig. 5. In general, the communication performance model well matches the measured results on our 8-node cluster.

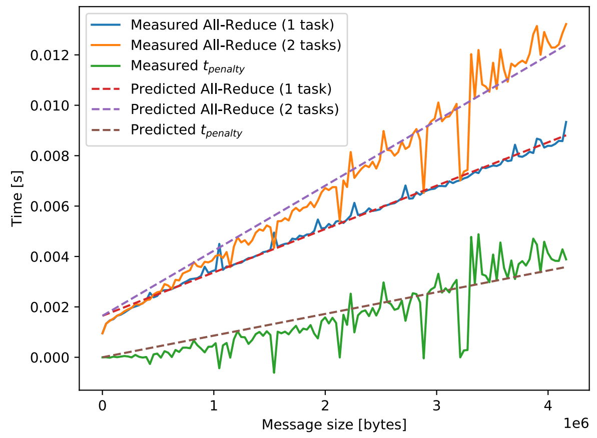


Fig. 5: Measured and predicted communication contention.

C. *Wall-clock Iteration Time*

TABLE IV: Average iteration wall-clock time (in seconds) of 1,000 running iterations. S_1 and S_2 represent the speedup of ASC-WFBP over WFBP and MG-WFBP, respectively.

Model	WFBP	MG-WFBP	ASC-WFBP	S_1	S_2
ResNet-152	0.956	0.710	0.524	1.82	1.35
DenseNet-201	0.778	0.375	0.314	2.48	1.20
Inception-v4	0.719	0.510	0.377	1.91	1.35
BERT-Base	1.010	1.062	0.924	1.09	1.15

The wall-clock iteration time is measured by $1,000$ iterations with extra 100 warmup iterations for each algorithm. Note that the scheduling algorithms have no side-effect on the convergence speed of S-SGD in terms of the number of iterations, so we focus on the wall-clock iteration time comparison. The experimental results are shown in Table IV. As expected, MG-WFBP is generally better than WFBP, while our ASC-WFBP outperforms MG-WFBP in all evaluated models. Overall, ASC-WFBP achieves $1.15-1.35\times$ speedup over MG-WFBP, and $1.09-2.48\times$ speedup over WFBP.

D. *Time Breakdown*

To further understand the performance improvement, we breakdown the iteration time into computation time and communication time as shown in Fig. 6. Note that the communication time shown in Fig. 6 is the non-overlapped communication overhead, i.e., it excludes the time that has been hidden by the computation tasks. We can observe that the communication time occupies a large proportion of the iteration time due

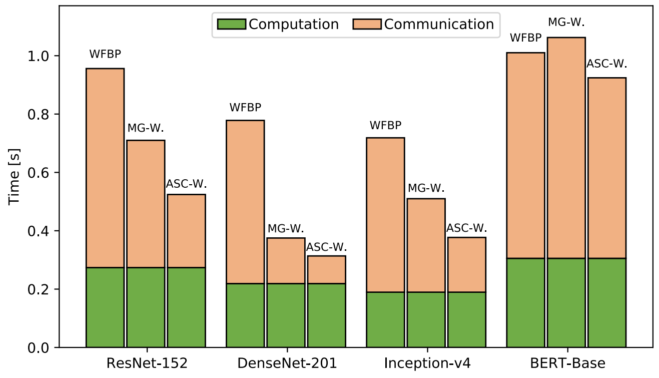


Fig. 6: Breakdown of iteration time. The computation time includes forward and backward computations. The communication time excludes the part hidden by computations.

to two main reasons: 1) the 10GbE interconnect is not fast enough, and 2) the small GPU memory limits the batch size and hence the communication-to-computation ratio is high. In terms of the communication cost, MG-WFBP can achieve more than 65% reduction over WFBP on CNNs, while our proposed ASC-WFBP further achieves $22\%-74\%$ reduction over MG-WFBP. The experimental results show that even though MG-WFBP generates a good tensor fusion solution for merging nearby gradients, it still cannot fully utilize the bandwidth resource to achieve efficient communications. Our ASC-WFBP exploits the opportunity of simultaneous communications to further improve the link bandwidth utilization and alleviate the waiting cost for some layers.

E. *Discussion*

In the above presented results, the overall end-to-end training time of ASC-WFBP is about $15\%-35\%$ better than MG-WFBP, which is significant because the maximum speedup achieved by scheduling is limited by the communication-to-computation ratio of the target DNN model.

As we discussed in Section II-B, there are three major steps in S-SGD: forward computation (t_f), backward computation (t_b), and gradient communication (t_c). Assume that the overall gradient size of the DNN is m, the communication cost can not be less than the communication time when the link bandwidth is fully utilized. For the ring-based All-Reduce algorithm, the minimal time of aggregating the gradients should be at least $t_c^{min} = 2m/B$ [34], where B is the link bandwidth. For any scheduling algorithms that pipeline the backward computations and communications on a P-worker cluster, the speedup of the overall throughput over the single worker is limited by

$$S^{max} = \frac{P \times (t_f + t_b)}{t_f + t_b + t_c^{min} - t^{hidden}}, \tag{41}$$

where t^{hidden} is the overlapped time between communications and backward computations. In the optimal scenario, either backward computations or communications are fully hidden, so $t^{hidden} = \min\{t_b, t_c^{min}\}$. Thus, the P-worker system can achieve the following maximal speedup over a singe worker:

$$S^{max} = \frac{P \times (t_f + t_b)}{t_f + t_b + t_c^{min} - \min\{t_b, t_c^{min}\}}. \quad (42)$$

Taking the ResNet-152 model as an example, which has about 60 million parameters (each gradient is represented by a 32-bit floating point number), we can calculate t_c^{min} on the 10GbE connected cluster by

$$t_c^{min} = \frac{2 \times 60 \times 10^6 \times 32}{9.43 \times 10^9} = 0.407 second. \quad (43)$$

The forward and backward computation time of ResNet-152 with a mini-batch size of 32 on an Nvidia RTX2080Ti GPU are about $0.091s$ and $0.182s$ respectively. Therefore, in ResNet-152, we obtain $S^{max} = 17.5$. The iteration time of ASC-WFBP is $0.524s$ on the 32-GPU cluster, so the actual speedup of throughput over single GPU is $S = 32 \times (0.091 + 0.182)/0.524 = 16.7$, which is very close to S^{max}. We compare the real throughput speedup of ASC-WFBP and the theoretical optimal speedup in Table V. On average, ASC-WFBP achieves 87.7% of the theoretical optimal speedup.

TABLE V: Comparison between the real speedup (S) of ASC-WFBP on the 32-GPU cluster over single GPU and the theoretical maximal speedup (S^{max}).

	ResNet-152	DenseNet-201	Inception-v4	BERT-Base
S^{max}	17.5	32.0	17.2	11.5
S	16.7	22.3	16.1	10.6
S/S^{max}	95.4%	69.7%	93.6%	92.2%

In particular, with BERT-Base, ASC-WFBP only achieves 15% improvement over MG-WFBP as shown in Table V, while our algorithm scales out in good scalability that is very close to the maximum scalability in the 32-GPU system. However, with DenseNet-201, ASC-WFBP has only about $22\times$ speedup while the optimal speedup can be 32. This is mainly because DenseNet-201 has the largest number of tensors but the smallest number of parameters among the four evaluated models, i.e., its average tensor size is the smallest. Hence its bandwidth utilization is the lowest. The current design of ASC-WFBP only considers two-way simultaneous communications. Allowing more simultaneous communications makes the theoretical analysis more challenging, and we leave it as our future work.

VII. Related Work

There are extensive designs of communication-efficient techniques for S-SGD based distributed DL. Due to limited space, we only discuss two most relevant areas: 1) efficient communication collectives, and 2) scheduling algorithms to overlap communication tasks and computing tasks.

Efficient Collectives: The communication cost in S-SGD is introduced by the gradient aggregation with an All-Reduce collective or through parameter servers. Thus, the communication complexity of the All-Reduce algorithms directly determines the communication cost. The ring-based All-Reduce algorithm is successfully applied in dense-GPU clusters in DL applications, and it is still widely used in highly optimized libraries like NCCL and Horovod. There exist other highly optimized All-Reduce algorithms [2,21,35]–[40] for different systems. For example, Wang et al., [38] proposed the efficient All-Reduce algorithms for Fat-tree and BCube topologies, while Ying et al., [41] proposed a two rings algorithm on the Torus topology. Some optimizations [42,43] target at the Nvidia GPU systems equipped with NVLink connections, while the PLink system in [44] targets at public cloud clusters.

Scheduling Algorithms: Due to the tensor-wise or layer-wise structure of DNNs, computation and communication tasks can be highly parallelized so that some communication costs can be hidden by the computing cost. Thus, it requires wise and dynamic scheduling algorithms to better overlap these tasks during the training process. The classical wait-free backpropagation (WFBP) algorithm [21,22] invokes the gradient communication when the gradients are ready and the communication link is free. By decoupling the communication priority of gradient aggregation, one can make the communication tasks be overlapped with feed-forward computations [4,25]–[27]. To reduce the impact of startup overhead of small-message communications when parallelizing the layer-wise communications and computations, tensor fusion techniques have been proposed [2]–[4,16]. However, existing tensor fusion solutions do not allow multiple communication tasks to be executed simultaneously, which could not fully utilize the available network bandwidth.

VIII. Conclusion

In synchronous SGD with data-parallelism, wait-free back-propagation (WFBP) overlaps the layer-wise computation and communication tasks, but it could suffer from the startup problem of small tensors. Existing tensor fusion techniques can merge multiple small tensors to better utilize the network bandwidth. In this paper, we proposed to exploit simultaneous All-Reduce communications to further improve communication efficiency. We built a simple communication model to predict the time performance of simultaneous All-Reduce communications and verified its accuracy through experiments. We then formulated an optimization problem to determine which strategy should be used for specific layers and developed an efficient optimal solution. We evaluated our prototype system ASC-WFBP on our testbed of 32-GPU cluster using four modern DNNs. The experimental results showed that our system significantly outperforms existing WFBP and MG-WFBP algorithms by adaptively selecting the best schedule of communication tasks.

Acknowledgments

The research was supported in part by Hong Kong RGC GRF grants under the contracts HKBU 12200418 and RMGS2019_1_23 from Hong Kong Research Matching Grant Scheme. It was also supported in part by Hong Kong RGC GRF grants under the contracts 16206417 and 16207818.

References

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

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

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

[4] Y. Bao, Y. Peng, Y. Chen, and C. Wu, "Preemptive all-reduce scheduling for expediting distributed DNN training," in *Proc. of IEEE INFOCOM*. IEEE, 2020, pp. 626–635.

[5] S. Shi, W. Qiang, and X. Chu, "Performance modeling and evaluation of distributed deep learning frameworks on GPUs," in *Proc. of DataCom 2018*. IEEE, 2018.

[6] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. of CVPR*, 2016, pp. 770–778.

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

[8] C. Chen, W. Wang, and B. Li, "Round-robin synchronization: Mitigating communication bottlenecks in parameter servers," in *Proc. of IEEE INFOCOM*. IEEE, 2019, pp. 532–540.

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

[10] X. Lian, H. Zhang, C.-J. Hsieh, Y. Huang, and J. Liu, "A comprehensive linear speedup analysis for asynchronous stochastic parallel optimization from zeroth-order to first-order," in *Advances in Neural Information Processing Systems*, 2016, pp. 3054–3062.

[11] 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.

[12] 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.

[13] 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.

[14] C. Renggli, S. Ashkboos, M. Aghagolzadeh, D. Alistarh, and T. Hoefler, "Sparcml: High-performance sparse communication for machine learning," in *Proc. of SC'19*, 2019, pp. 1–15.

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

[16] S. Shi, Q. Wang, X. Chu, B. Li, Y. Qin, R. Liu, and X. Zhao, "Communication-efficient distributed deep learning with merged gradient sparsification on GPUs," in *Proc. of IEEE INFOCOM*. IEEE, 2020, pp. 406–415.

[17] C.-Y. Chen, J. Ni, S. Lu, X. Cui, P.-Y. Chen, X. Sun, N. Wang, S. Venkataramani, V. V. Srinivasan, W. Zhang *et al.*, "ScaleCom: Scalable sparsified gradient compression for communication-efficient distributed training," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[18] S. Zheng, Q. Meng, T. Wang, W. Chen, N. Yu, Z.-M. Ma, and T.-Y. Liu, "Asynchronous stochastic gradient descent with delay compensation," in *International Conference on Machine Learning*, 2017, pp. 4120–4129.

[19] S. P. Karimireddy, Q. Rebjock, S. Stich, and M. Jaggi, "Error feedback fixes SignSGD and other gradient compression schemes," in *International Conference on Machine Learning*, 2019, pp. 3252–3261.

[20] S. Shi, K. Zhao, Q. Wang, Z. Tang, and X. Chu, "A convergence analysis of distributed SGD with communication-efficient gradient sparsification." in *Proc. of IJCAI*, 2019, pp. 3411–3417.

[21] A. A. Awan, K. Hamidouche, J. M. Hashmi, and D. K. Panda, "S-caffe: Co-designing MPI runtimes and Caffe for scalable deep learning on modern GPU clusters," in *Proc. of PPoPP*, 2017, pp. 193–205.

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

[23] Y. You, A. Buluç, and J. Demmel, "Scaling deep learning on gpu and knights landing clusters," in *Proc. of SC'17*, 2017, pp. 1–12.

[24] S. Shi, X. Chu, and B. Li, "MG-WFBP: Merging gradients wisely for efficient communication in distributed deep learning," *IEEE Transactions on Parallel and Distributed Systems*, 2021.

[25] S. H. Hashemi, S. A. Jyothi, and R. Campbell, "TicTac: Accelerating distributed deep learning with communication scheduling," in *Proc. of MLSys 2019*, 2019.

[26] A. Jayarajan, J. Wei, G. Gibson, A. Fedorova, and G. Pekhimenko, "Priority-based parameter propagation for distributed DNN training," in *Proc. of MLSys 2019*, 2019.

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

[28] A. Sergeev and M. D. Balso, "Horovod: fast and easy distributed deep learning in TensorFlow," *arXiv preprint arXiv:1802.05799*, 2018.

[29] Z. Liu, K. Chen, H. Wu, S. Hu, Y.-C. Hut, Y. Wang, and G. Zhang, "Enabling work-conserving bandwidth guarantees for multi-tenant datacenters via dynamic tenant-queue binding," in *Proc. of IEEE INFOCOM*. IEEE, 2018, pp. 1–9.

[30] S. Shi, X. Zhou, S. Song, X. Wang, Z. Zhu, X. Huang, X. Jiang, F. Zhou, Z. Guo, L. Xie *et al.*, "Towards scalable distributed training of deep learning on public cloud clusters," *preprint arXiv:2010.10458*, 2020.

[31] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. of CVPR*, 2017, pp. 4700–4708.

[32] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Proc. of the 31st AAAI*, 2017.

[33] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proc. of ACL*, 2019, pp. 4171–4186.

[34] R. Rabenseifner, "Optimization of collective reduction operations," in *International Conference on Computational Science*. Springer, 2004, pp. 1–9.

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

[36] A. A. Awan, K. Hamidouche, A. Venkatesh, and D. K. Panda, "Efficient large message broadcast using NCCL and CUDA-aware MPI for deep learning," in *Proc. of the 23rd European MPI Users' Group Meeting*, 2016, pp. 15–22.

[37] H. Mikami, H. Suganuma, Y. Tanaka, Y. Kageyama *et al.*, "Massively distributed sgd: Imagenet/resnet-50 training in a flash," *arXiv preprint arXiv:1811.05233*, 2018.

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

[39] M. Cho, U. Finkler, and D. Kung, "Blueconnect: Novel hierarchical all-reduce on multi-tired network for deep learning," in *Proc. of MLSys 2019*, 2019.

[40] S. Li, T. Ben-Nun, S. D. Girolamo, D. Alistarh, and T. Hoefler, "Taming unbalanced training workloads in deep learning with partial collective operations," in *Proc. of the 25th PPoPP*, 2020, pp. 45–61.

[41] C. Ying, S. Kumar, D. Chen, T. Wang, and Y. Cheng, "Image classification at supercomputer scale," in *Proc. of Workshop on Systems for ML and Open Source Software, collocated with NeurIPS 2018*, 2018.

[42] G. Wang, S. Venkataraman, A. Phanishayee, N. Devanur, J. Thelin, and I. Stoica, "Blink: Fast and generic collectives for distributed ML," in *Proc. of MLSys 2020*, 2020, pp. 172–186.

[43] C.-H. Chu, P. Kousha, A. A. Awan, K. S. Khorassani, H. Subramoni, and D. K. Panda, "NV-group: link-efficient reduction for distributed deep learning on modern dense GPU systems," in *Proc. of the 34th ACM International Conference on Supercomputing*, 2020, pp. 1–12.

[44] L. Luo, P. West, J. Nelson, A. Krishnamurthy, and L. Ceze, "PLink: Discovering and exploiting locality for accelerated distributed training on the public cloud," in *Proc. of MLSys 2020*, 2020, pp. 82–97.