MG-WFBP: Efficient Data Communication for Distributed Synchronous SGD Algorithms

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

***Abstract*—Distributed synchronous stochastic gradient descent has been widely used to train deep neural networks on computer clusters. With the increase of computational power, network communications have become one limiting factor on the system scalability. In this paper, we observe that many deep neural networks have a large number of layers with only a small amount of data to be communicated. Based on the fact that merging some short communication tasks into a single one may reduce the overall communication time, we formulate an optimization problem to minimize the training iteration time. We develop an optimal solution named merged-gradient wait-free backpropagation (MG-WFBP) and implement it in our open-source deep learning platform B-Caffe. Our experimental results on an 8-node GPU cluster with 10GbE interconnect and trace-based simulation results on a 64-node cluster both show that the MG-WFBP algorithm can achieve much better scaling efficiency than existing methods WFBP and SyncEASGD.**

***Index Terms*—Deep Learning; GPU; Distributed Stochastic Gradient Descent; Gradient Communication; Merged-gradient**

I. Introduction

The data-parallel synchronous stochastic gradient descent (S-SGD) method is commonly used as the optimizer to train the large scale deep neural networks (DNNs) [1][2]. In S-SGD, the computing tasks for each mini-batch of training data is distributed to a cluster of computing nodes, and the individual results are aggregated to update the global network model before the next iteration can begin. However, with more computing nodes and the fast-growing computing power of hardware accelerators, the data communication between computing nodes gradually becomes the performance bottleneck [3][4][5]. For example, the computing power of Nvidia GPUs has increased by 30x in the last 10 years, whilst it took about 15 years for the network speed to improve from 10Gbps to 100Gbps. Hence it becomes a critical issue to address the imbalance between computing and communication.

Some recent work try to reduce the impact of data communication at both algorithmic and system levels. On one hand, gradients would be quantized and compressed [6][7][8] in order to reduce the data size during communication, but these methods usually sacrifice the model accuracy. On the other hand, the HPC community has proposed several methods to increase the communication performance of the cluster using both hardware and software approaches [9][10].

In terms of hardware, InfiniBand (IB) and Omni-Path networks can provide much higher communication bandwidth, and are deployed to reduce the performance gap between communication and computation [11]. Regarding the software, the implementation of message passing interface (MPI) has been further optimized to support efficient communication in DNN trainings [11][12]. The scaling efficiency of distributed deep learning systems can be modeled as a function of communication-to-computation ratio [7]. For example, training ResNet-50 [13] requires about 7.8 billion floating point operations in computation, while it takes 102 MB data communication in one iteration. Higher communication-to-computation ratio results in lower scaling efficiency.

The layered structure of DNNs makes it possible to overlap the communication and computation during the backward propagation [12][14][15][16], which is known as wait-free backpropagation (WFBP). WFBP begins to exchange the gradients of a layer immediately after they have been calculated; so if the data communication time of a layer is shorter than the computation time of the gradients of its previous layer, then this communication cost can be fully hidden. However, if very fast hardware accelerators are used while the network speed is relatively slow (i.e., a high communication-to-computation ratio), there can exist many layers whose communication time is longer than the corresponding computation time. In such cases, it becomes important to optimize the communications. We observe that the layer-wise gradient communication in WFBP is suboptimal due to the fact that transmitting a small amount of data cannot fully utilize the network bandwidth in current network topologies due to the startup time of message transmitting. Even the RDMA-based network is difficult to eliminate the high overhead of startup time when transmitting messages [17][18]. For example, on our 10GbE platform, exchanging a 200 KB vector across 8 nodes using MPI requires about 1.5ms, while exchanging a 400 KB vector only requires 1.8ms, which means we can merge two 200 KB vectors to one 400 KB vector to reduce the total communication time. Yang et al., [19] have also recently noticed this problem, and propose a single-layer communication (SyncEASGD) method in which all the gradients are merged together and transferred once per iteration. As compared to the layer-wise communication in WFBP, it can reduce most of the startup time of data communications. But in their proposed method, gradient communication can only start after the backward pass for all layers are finished, thus they miss the opportunity of overlapping the communication with computation.

We argue that the best way to reduce the training time needs to consider not only how to overlap communication with computation, but also how to improve the communication efficiency by avoiding transmitting small amount of data.

In this paper, we first formulate the communication scheduling problem in S-SGD as an optimization problem that aims to minimize the total training time of an iteration. And then we propose a merged-gradient wait-free backward propagation (MG-WFBP) method and prove its optimality. The time complexity of MG-WFBP is $O(L^2)$ where L is the number of layers in the DNN, and it only needs to be executed once before the whole training process. We implement MG-WFBP in our open-source distributed DL training platform B-Caffe[1], and evaluate its performance using two popular DNNs (i.e., GoogleNet [20] and ResNet-50 [13]). The experimental results on an 8-node cluster with Nvidia Tesla K80 GPU and 10GbE show that MG-WFBP can achieve about 1.2x to 1.36x improvement than the state-of-the-art communication algorithms WFBP and SyncEASGD, respectively. To investigate its performance on large clusters, we resolve to trace-based simulation (due to limited hardware resources) on a 64-node cluster. In the 64-node simulation, the results show that MG-WFBP achieves more than 1.7x and 1.3x speedups compared to WFBP and SyncEASGD respectively. Our contributions are summarized as follows:

- We formulate an optimization problem for minimizing the training time of DNNs by merging consecutive data communications, and propose an optimal solution with very low computational cost.
- We implement our MG-WFBP algorithm in B-Caffe and make it open-source.
- We evaluate the performance of MG-WFBP through both real experiments and simulations, and make comparisons with WFBP, SyncEASGD and TensorFlow.

The rest of the paper is organized as follows. We present the preliminaries in Section II, followed by the formulation of the existing problem in Section III. We derive an optimal solution to the problem and then present our MG-WFBP S-SGD algorithm in Section IV. Section V demonstrates the evaluation of the proposed method with experimental results. Section VI introduces the related work, and finally we conclude this paper in Section VII.

II. Preliminaries

For ease of presentation, we summarize the frequently used mathematical notations in Table I.

TABLE I
FREQUENTLY USED NOTATIONS

Name	Description
N	The number of nodes in the cluster.
α	Latency (startup time) of the network between two nodes.
β	Transmission time per byte between two nodes.
γ	Summation time of two floating point numbers in one node.
a	Latency (startup time) of all-reduce.
b	Transmission and computation time per byte of all-reduce.
M	The size of a message in bytes.
W	Weights of the DNN.
D_i^g	The input data size for the g^{th} node at the i^{th} mini-batch.
L	The number of learnable layers of a DNN.
$p^{(l)}$	The number of parameters in the learnable layer l.
t_{iter}	Time of an iteration.
t_f	Time of the forward pass in each iteration.
t_b	Time of the backward propagation in each iteration.
t_u	Time of the model update in each iteration.
$t_b^{(l)}$	Time of the backward propagation of layer l in each iteration.
$\tau_b^{(l)}$	The timestamp when layer l begins calculating gradients.
$\mu_b^{(l)}$	The timestamp when layer l finishes calculating gradients.
t_c	Time of gradient aggregation in each iteration.
t_c^{no}	The not overlapped communication cost in each iteration.
$t_c^{(l)}$	Time of gradient aggregation of layer l in each iteration.
$\tau_c^{(l)}$	The timestamp when layer l begins communicating gradients.
$\mu_c^{(l)}$	The timestamp when layer l finishes communicating gradients.

A. Mini-batch SGD

The DNN needs a loss function $\mathcal{L}(W, D)$, where W and D are the model weights and the input data respectively, to define the differences between the prediction values and the ground truth. To minimize the loss function, the mini-batch SGD updates the parameters iteratively. Typically, the i^{th} iteration of the training includes four steps: 1) A mini-batch of data D_i ($D_i \subset D$) is read as inputs of the DNN. 2) D_i is fed forward across the neural network from layer 1 to layer L to compute the prediction values at the last layer, and the value of the loss function $\mathcal{L}(W, D)$ is computed. 3) The first order gradients w.r.t. parameters and inputs are calculated and backpropagated with from layer L to layer 1. 4) Finally, the parameters are updated with the layer-wise gradients. The training is terminated when some stopping criteria are matched. The update of W can be formulated as follows:

$$W_{i+1} = W_i - \eta \cdot \nabla\mathcal{L}(W_i, D_i), \tag{1}$$

where η is the learning rate of SGD, W_i represents the weights at the i^{th} iteration, and $\nabla\mathcal{L}(W_i, D_i)$ are the gradients. The time consumed in the training processes is mainly in steps 2 and 3, because step 1 of the i^{th} iteration can overlap with the $(i-1)^{th}$ iteration, and the time of step 4 is negligible. Therefore, we can simplify the time-line of SGD as forward and backward passes. The time of one iteration is hence represented by $t_{iter} = t_f + t_b$.

B. Synchronized SGD for clusters

For large-scale DNNs, the data-parallelism synchronized SGD (S-SGD) is widely applied to train models with multiple workers (say N workers, and indexed by g). Each worker takes a different mini-batch of data D_i^g and forwards it by step 2), and then follows step 3) to calculate the gradients $\nabla\mathcal{L}(W_i, D_i^g)$. In this way, each worker has a copy of the model, while the gradients are not the same in each iteration since the input data are different; therefore, to keep explicitly the same as SGD, it needs to average the gradients from

[1]https://github.com/hclhkbu/B-Caffe.

different workers before updating the model. The update formula of parameters is rewritten as

$$W_{i+1} = W_i - \eta \cdot \frac{1}{N} \sum_{g=1}^{N} \nabla \mathcal{L}(W_i, D_i^g). \tag{2}$$

As a result, the averaging operation of gradients across the cluster involves extra computation and communication overheads, which makes it difficult to achieve linear scaling in the distributed SGD training. The time-line of the naive S-SGD (i.e., computation and communication are not overlapped) with communication overheads is illustrated in Fig. 1. The naive S-SGD algorithm suffers from the waiting period of data communication of model synchronization at every iteration. The iteration time of the naive S-SGD can be estimated as

$$t_{iter} = t_f + t_b + t_c, \tag{3}$$

where $t_b = \sum_{l=1}^{L} t_b^{(l)}$ is the backward propagation time and $t_c = \sum_{l=1}^{L} t_c^{(l)}$ is the gradient aggregation time which heavily relies on the communication speed.

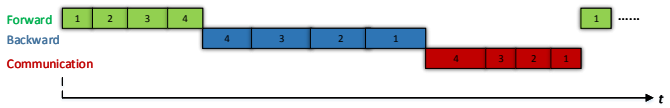


Fig. 1. The time-line of naive S-SGD for a 4-layer network with communication overheads. The feed forward cannot be started until the end of model communication (red rectangles) of its previous iteration.

Considering S-SGD using weak-scaling running on N nodes, we define the speedup of S-SGD compared to the single-node SGD:

$$S(N) = \frac{N|D_i^g|/(t_f + t_b + t_c)}{|D_i^g|/(t_f + t_b)} = \frac{N}{1 + \frac{t_c}{t_f + t_b}}, \tag{4}$$

where $|D_i^g|$ is the number of samples per node at the i^{th} iteration. Let $r = \frac{t_c}{t_f + t_b}$, which reflects the communication-to-computation ratio, we have

$$S(N) = \frac{N}{1 + r}. \tag{5}$$

C. WFBP

In WFBP, the layer-wise gradient communication can be overlapped with the backward propagation of its previous layer. An example of S-SGD with WFBP is illustrated in Fig. 2. For simplicity, we assume that the start timestamp of the forward pass is 0, then the start timestamp of backward pass of each layer can be represented by

$$\tau_b^{(l)} = \begin{cases} t_f & l = L \\ \tau_b^{(l+1)} + t_b^{(l+1)} & 1 \le l < L \end{cases}. \tag{6}$$

And the start timestamp of communication of each layer can be represented by

$$\tau_c^{(l)} = \begin{cases} \tau_b^{(l)} + t_b^{(l)} & l = L \\ \max\{\tau_c^{(l+1)} + t_c^{(l+1)}, \tau_b^{(l)} + t_b^{(l)}\} & 1 \le l < L \end{cases}. \tag{7}$$

The iteration time can be rewritten as

$$\begin{aligned} t_{iter} &= t_f + t_b^{(L)} + t_c^{(1)} - \tau_c^{(L)} + \tau_c^{(1)} \\ &= t_c^{(1)} + \max\{\tau_c^{(2)} + t_c^{(2)}, \tau_b^{(1)} + t_b^{(1)}\}. \end{aligned} \tag{8}$$

Since some communication costs can be overlapped with the computation, the non-overlapped communication cost, t_c^{no}, becomes the bottleneck of the system. In WFBP, we redefine $r = \frac{t_c^{no}}{t_f + t_b}$, so the main problem of WFBP is that when the communication cannot be fully overlapped by computation, i.e., $\tau_c^{(l+1)} + t_c^{(l+1)} > \tau_b^{(l)} + t_b^{(l)}$, t_c^{no} will limit the system scalability.

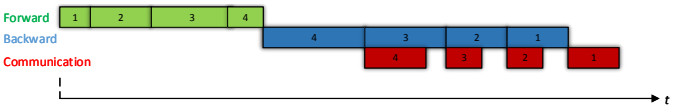


Fig. 2. The time-line of WFBP for a 4-layer neural network with communication overheads. Gradient communication of each layer begins immediately after the backward step of that layer.

D. Communication model

In Eq. 2, we use $\Delta W_i = \sum_{g=1}^{N} \nabla \mathcal{L}(W_i, D_i^g)$ to represent the aggregation of gradients from N workers, which is an all-reduce operation. There are many optimized algorithms for the all-reduce operation with different number of processes and message sizes [21][22][23]. To simplify the problem, we assume that the number of GPUs is power-of-two, and the peer to peer communication cost is modeled as $\alpha + \beta M$ [24], where α is the latency component, β is the transmission time per byte, and M is the message size. Without loss of generality, we do not limit the communication model to one specific algorithm. Given a constant number of GPUs N, the time cost of all-reduce can be generalized as

$$T_{ar}(M) = a + bM, \tag{9}$$

where a and b are two constant numbers that are not related to M. Some optimized all-reduce algorithms are summarized in Table II.

TABLE II
COST OF DIFFERENT ALL-REDUCE ALGORITHMS

All-reduce Algorithm	a	b
Binary tree	$2\alpha log_2 N$	$(2\beta + \gamma) log_2 N$
Recursive doubling	$\alpha log_2 N$	$(\beta + \gamma) log_2 N$
Recursive halving and doubling	$2\alpha log_2 N$	$2\beta - \frac{1}{N}(2\beta + \gamma) + \gamma$
Ring	$2(N-1)\alpha$	$\frac{2(N-1)}{N}\beta + \frac{(N-1)}{N}\gamma$

With a given hardware configuration (i.e., N, α, β, and γ are fixed), the time cost of the all-reduce operation is a linear function of the variable M. The linear function has an y-intercept a and a slope b.

One important property of WFBP is that the messages are communicated layer by layer, which means that it needs to do many all-reduce operations. In each all-reduce, however, there is an extra cost of a which is not related with M. Importantly,

the linear function with a positive y-intercept value has a property of

$$T_{ar}(M_1) + T_{ar}(M_2) > T_{ar}(M_1 + M_2). \tag{10}$$

In other words, communicating an $M_1 + M_2$ bytes of message is more efficient than communicating an M_1 message and an M_2 message separately.

III. Problem Formulation

A. Problem formulation

As we have shown that merging the gradients can improve the communication efficiency in Eq. 10, we further discuss the complete set with three cases of WFBP, to explore under what scenarios can we merge gradients to reduce the iteration time. The three cases are illustrated in Fig. 3.

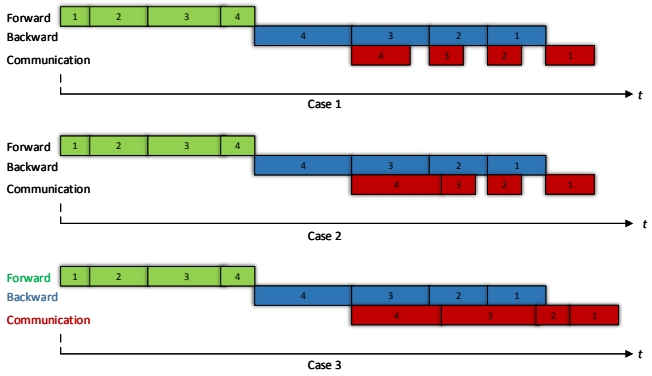


Fig. 3. Three cases of WFBP.

Case 1. In the ideal case, for $2 \leq l \leq L, t_c^{(l)} \leq t_b^{(l-1)}$. The overhead of gradient communication is totally hidden by computation so that it is not necessary to merge the gradients. The iteration time is

$$t_{iter} = t_f + t_b + t_c^{(1)}. \tag{11}$$

Case 2. There exists a layer (e.g., the C^{th} layer) whose communication time cannot be totally overlapped by computation, i.e., $t_c^{(C)} > t_b^{(C-1)}$, but all its following layers' communication can be fully overlapped. In other words, there exists a K ($C < K$), such that $\sum_{j=K}^{C-1} t_c^{(j)} < \sum_{j=K}^{C-1} t_b^{(j-1)}$, which means from K^{th} to C^{th} layers, the communication can be fully overlapped. As shown in the second sub-figure in Fig. 3, the communication time of layer 4 is larger than the computation time of layer 3, while the total communication time of layer 4 and layer 3 is shorter than the total computation time of layer 3 and layer 2. The communication time can also be totally hidden, which is the same with Case 1, so we have

$$t_{iter} = t_f + t_b + t_c^{(1)}. \tag{12}$$

In both Case 1 and Case 2, $r = \frac{t_c^{(1)}}{t_f + t_b}$, which is generally a small value because there is only an extra communication overhead from layer 1.

Case 3. Contrary to Case 2, if $t_c^{(C)} > t_b^{(C-1)}$, there does not exist a K, where $2 \leq K < C$, such that $\sum_{j=K}^{C-1} t_c^{(j)} < \sum_{j=K}^{C-1} t_b^{(j-1)}$. In other words, from the 2^{nd} to the C^{th} layer, the sum of communication costs can not be totally overlapped, so that the communication becomes the bottleneck. We have

$$t_{iter} = t_f + \sum_{j=C-1}^{L} t_b^{(j-1)} + \sum_{j=1}^{C} t_c^{(j)}. \tag{13}$$

Both Case 1 and Case 2 are ideal cases that the overhead of communication can be easily hidden. In the high latency or low bandwidth network environment, Case 3 could be more often happened. The main problem of Case 3 is that many layers' communication overheads cannot be hidden by the computation.

From the property of Eq. 10, two or more small messages can be merged to one larger size message before being exchanged. In other words, there exists an m, where $2 \leq m < C$, such that we can merge the gradients from the m^{th} layer to the C^{th} layer, and the merged gradients can be communicated with a cost that can be hidden by computation or smaller than the original cost. An example is shown in Fig. 4. The gradients

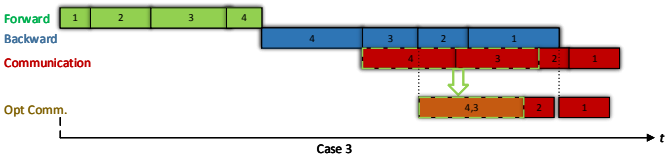


Fig. 4. Merged gradient communication.

of layer 4 and layer 3 are merged into one message to be exchanged with other nodes, which could reduce the overall transmission time. As a result, the communication of layer 2 can also be finished before the computation of layer 1.

In order to reduce the cost of communication, we need to find the optimal $\mathbb{M}$ so that for $m \in \mathbb{M}$, it has $t_c^{(C,m)} + \sum_{j=2}^{m-1} t_c^{(j)} \leq \sum_{j=1}^{m-1} t_b^{(j)}$. There may exist two scenarios: (1) If such m exists, then the iteration time t_{iter} becomes the same as Eq. 11. (2) If such m does not exist, then we need to minimize the total cost of communication. In summary, we need to find the set of layers $\mathbb{M} = \{l | 2 \leq l \leq C\}$, whose gradients are merged with their previous layers and be communicated together, such that the iteration time is minimal.

Definition 1. (Merged-gradient). A layer l is called a merged-gradient layer if at the timestamp of $\tau_c^{(l)}$, instead of communicating the gradients of that layer, merging its gradients to its previous layer $l-1$ to be communicated together. The operator $\oplus$ defines the gradients merging between two consecutive layers, say $(l) \oplus (l-1)$.

Conditions 1. If layer l is a merged-gradient layer, then the following three conditions hold.

$$t_c^{(l)} = 0 \tag{14}$$

$$\tau_c^{(l-1)} = \max\{\tau_c^{(l)}, \tau_b^{(l-1)} + t_b^{(l-1)}\} \tag{15}$$

$$p^{(l-1)} = p^{(l-1)} + p^{(l)} \tag{16}$$

Condition 14 indicates that layer l does not need to be communicated by itself so it has zero overhead. Condition 15

is obvious by Eq. 7. Condition 16 indicates that the gradients of l should be merged to layer $l-1$ such that the number of updated parameters of layer $l-1$ becomes the summation of layer l and layer $l-1$.

From Eq. 17, the communication time of each layer is represented by

$$t_c^{(l)} = T_{ar}(p^{(l)}). \tag{17}$$

To be more generalized, the time cost of backward computation is modeled as a function with respect to the FLOPS (floating-point operation per second) of the processor G, the number of parameters and some other factors θ (e.g., batch size and type of layers) [25], say

$$t_b^{(l)} = T_b(p^{(l)}, G, \theta) \tag{18}$$

Plug Eq. 17 and Eq. 18 in Eq. 7 and Eq. 6, we obtain

$$\tau_b^{(l)} = \begin{cases} t_f & l = L \\ \tau_b^{(l+1)} + T_b(p^{(l+1)}, G, \theta) & 1 \le l < L \end{cases}. \tag{19}$$

And the start time of communication of each layer can be represented by

$$\tau_c^{(l)} = \begin{cases} \tau_b^{(l)} + T_b(p^{(l)}, G, \theta) & l = L \\ \max\{\tau_c^{(l+1)} + T_{ar}(p^{(l+1)}), \tau_b^{(l)} + t_b^{(l)}\} & 1 \le l < L \end{cases}. \tag{20}$$

The iteration time can be rewritten as

$$t_{iter} = T_{ar}(p^{(1)}) + \max\{\tau_c^{(2)} + T_{ar}(p^{(2)}), \tau_b^{(1)} + T_b(p^{(1)}, G)\}. \tag{21}$$

So we can formulate the problem as follows. Given a DNN with L learnable layers training with S-SGD across N nodes, we want to find a set of merged-gradient layers

$$\mathbb{M} = \{l | \text{layer } l \text{ is a merged-gradient layer, and } 2 \le l \le L\}, \tag{22}$$

such that the iteration time t_{iter} of Eq. 21 is minimal. In a specific cluster of N nodes, and each node has G FLOPS of computation capability, we can remove the notation of G in Eq. 21. At last, we formulate the following optimization problem:

$$\text{minimize } T_{ar}(p^{(1)}) + \max\{\tau_c^{(2)} + T_{ar}(p^{(2)}), \tau_b^{(1)} + T_b(p^{(1)}))\}. \tag{23}$$

IV. Solution

In this section, we first perform some theoretical analysis on the optimization problem, and then we propose an optimal and efficient solution to the problem.

A. Theoretical analysis

The first term of Eq. 23 can be neglected to find the solution, so the objective function becomes

$$\begin{aligned} t &= \max\{\tau_c^{(2)} + T_{ar}(p^{(2)}), \tau_b^{(1)} + T_b(p^{(1)})\} \\ &= \max\{\max\{\tau_c^{(3)} + T_{ar}(p^{(3)}), \tau_b^{(2)} + T_b(p^{(2)})\} + T_{ar}(p^{(2)}), \\ &\quad \tau_b^{(1)} + T_b(p^{(1)})\} \end{aligned} \tag{24}$$

Assume that layer 3 is a merged-gradient layer, we have $t_c^{(3)} = 0$ and $t_c^{(2)} = T_{ar}(p^{(2)} + p^{(3)})$. We plug in these two new values to the above equation. Thus,

$$\begin{aligned} \hat{t} &= \max\{\max\{\tau_c^{(3)}, \tau_b^{(2)} + T_b(p^{(2)})\} + T_{ar}(p^{(2)} + p^{(3)}), \\ &\quad \tau_b^{(1)} + T_b(p^{(1)})\} \end{aligned} \tag{25}$$

Compare Eq. 24 to Eq. 25, we want to prove that in what conditions $\hat{t} < t$, i.e., layer 3 can be a gradient-merged layer. In Eq. 24 and Eq. 25, the second term of outer max function is the same, so we can just compare the first term, i.e.,

$$\begin{aligned} &\max\{\tau_c^{(3)}, \tau_b^{(2)} + T_b(p^{(2)})\} + T_{ar}(p^{(2)} + p^{(3)}) < \\ &\max\{\tau_c^{(3)} + T_{ar}(p^{(3)}), \tau_b^{(2)} + T_b(p^{(2)})\} + T_{ar}(p^{(2)}). \end{aligned}$$

Since $\tau_b^{(1)} = \tau_b^{(2)} + T_b(p^{(2)})$, we simplify the above inequality and obtain

$$\begin{aligned} &\max\{\tau_c^{(3)}, \tau_b^{(1)}\} + T_{ar}(p^{(2)} + p^{(3)}) < \\ &\max\{\tau_c^{(3)} + T_{ar}(p^{(3)}), \tau_b^{(1)}\} + T_{ar}(p^{(2)}). \end{aligned} \tag{26}$$

To eliminate the max function, we consider the following three conditions.

C.1. $\tau_b^{(1)} < \tau_c^{(3)}$. The inequality 26 becomes

$$\tau_c^{(3)} + T_{ar}(p^{(2)} + p^{(3)}) < \tau_c^{(3)} + T_{ar}(p^{(3)}) + T_{ar}(p^{(2)}),$$

i.e.,

$$T_{ar}(p^{(2)} + p^{(3)}) < T_{ar}(p^{(3)}) + T_{ar}(p^{(2)}),$$

which holds due to the property of T_{ar} in Eq. 10.

C.2. $\tau_c^{(3)} \le \tau_b^{(1)} < \tau_c^{(3)} + T_{ar}(p^{(3)})$. Then we obtain

$$\tau_b^{(1)} + T_{ar}(p^{(2)} + p^{(3)}) < \tau_c^{(3)} + T_{ar}(p^{(3)}) + T_{ar}(p^{(2)}).$$

According to Eq. 9,

$$T_{ar}(p^{(3)}) + T_{ar}(p^{(2)}) - T_{ar}(p^{(2)} + p^{(3)}) = a,$$

then the above inequality holds under the condition

$$\tau_b^{(1)} - \tau_c^{(3)} < a.$$

C.3. $\tau_b^{(1)} \ge \tau_c^{(3)} + T_{ar}(p^{(3)})$. The inequality 26 becomes

$$\tau_b^{(1)} + T_{ar}(p^{(2)} + p^{(3)}) < \tau_b^{(1)} + T_{ar}(p^{(3)}) + T_{ar}(p^{(2)}),$$

i.e., $T_{ar}(p^{(2)} + p^{(3)}) < T_{ar}(p^{(3)}) + T_{ar}(p^{(2)})$, which is in contradiction with Eq. 9. So under **C.3**, layer 3 should not be a merged-gradient layer.

Since a layer of a specific DNN can only be in one of the above conditions, we have the following theorem to find the optimal solution.

Theorem 1. Given an L-layer DNN which is trained with S-SGD in a cluster of N nodes, if the gradient communication is done through all-reduce, one can find all the merged-gradient layers $\mathbb{M}$ such that the iteration time is minimal, and

$$\mathbb{M} = \{l | \tau_b^{(l-2)} - \tau_c^{(l)} < a, \text{and } 1 < l \le L\}. \tag{27}$$

Proof. For the l^{th} layer, where $1 < l \leq L$, since the layer l has only two choices (merged-gradient layer or NOT merged-gradient). If $\tau_b^{(l-2)} - \tau_c^{(l)} < b$, we just need to prove the communication of its previous layer $l-1$ can be finished earlier if layer l is a merged-gradient layer. I.e.,

$$\begin{aligned} &\max\{\tau_c^{(l)}, \tau_b^{(l-2)}\} + T_{ar}(p^{(l-1)} + p^{(l)}) < \\ &\max\{\tau_c^{(l)} + T_{ar}(p^{(l)}), \tau_b^{(l-2)}\} + T_{ar}(p^{(l-1)}). \end{aligned} \tag{28}$$

As we have discussed in both **C.1** and **C.2** conditions, the above inequality holds in the condition of $\tau_b^{(l-2)} - \tau_c^{(l)} < b$. So under this condition, the start time stamp of layer $l-1$ is smaller than that not merged. Consequently, for all $\tau_b^{(l-2)} - \tau_c^{(l)} < a$, we do $(l) \oplus (l-1)$, then Eq. 23 holds. ■

B. Algorithm

Assume that the N-node cluster is connected by a switch network where each node has a bandwidth B and computation capability of G. Then the time cost of layer-wise communication can be computed according to Eq. 17, and the computation cost of backward propagation can be calculated by Eq. 18. Thus, t_f, $t_c^{(l)}$ and $t_b^{(l)}$, where $1 \leq l \leq L$, are known. According to Theorem 1, we design the algorithm to find $\mathbb{M}$ as shown in Algorithm 1.

Algorithm 1 Find all merged-gradient layers: $\mathbb{M}$

Input: a, b, L, N, G, $\boldsymbol{p} = [p^{(1)}, p^{(2)}, ..., p^{(L)}]$.
Output: $\mathbb{M}$

1: Initialize $\boldsymbol{tc}[1...L]$; // Communication time cost
2: Initialize $\boldsymbol{tb}[1...L]$; // Backward computation time cost
3: Initialize $\boldsymbol{\tau b}[1...L]$; // Backward computation start time
4: tf=$\sum_{l=1}^{L} T_f(\boldsymbol{p}[l], G)$;
5: **for** $l = 1 \rightarrow L$ **do**
6: $\quad \boldsymbol{tc}[l] = T_{ar}(\boldsymbol{p}[l], N)$;
7: $\quad \boldsymbol{tb}[l] = T_b(\boldsymbol{p}[l], G)$;
8: $\boldsymbol{\tau b}[L]$=tf;
9: **for** $l = L-1 \rightarrow 2$ **do**
10: $\quad \boldsymbol{\tau b}[l]$ = $\boldsymbol{\tau b}[l+1]$ + $\boldsymbol{tb}[l+1]$
11: $\boldsymbol{\tau c}$=CalculateCommStart($\boldsymbol{tc}, \boldsymbol{tb}, \boldsymbol{\tau b}, L$);
12: **for** $l = L \rightarrow 2$ **do**
13: $\quad$ **if** $\boldsymbol{\tau b}[l-2] - \boldsymbol{\tau c}[l] < a$ **then**
14: $\quad\quad$ Merge($\boldsymbol{\tau b}, \boldsymbol{tc}, \boldsymbol{p}, l, N$);
15: $\quad\quad$ $\boldsymbol{\tau c}$=CalculateCommStart($\boldsymbol{tc}, \boldsymbol{tb}, \boldsymbol{\tau b}, L$);
16: $\quad\quad$ $\mathbb{M}$.push(l);
17: **procedure** Merge($\boldsymbol{\tau b}, \boldsymbol{tc}, \boldsymbol{p}, l, N$)
18: $\quad \boldsymbol{tc}[l] = 0$;
19: $\quad \boldsymbol{p}[l-1] = \boldsymbol{p}[l-1] + \boldsymbol{p}[l]$;
20: $\quad \boldsymbol{tc}[l-1] = T_{ar}(\boldsymbol{p}[l-1], N)$;
21: **procedure** CalculateCommStart($\boldsymbol{tc}, \boldsymbol{tb}, \boldsymbol{\tau b}, L$)
22: $\quad$ Initialize $\boldsymbol{\tau c}[1...L]$; // Communication start time
23: $\quad \boldsymbol{\tau c}[L] = \boldsymbol{\tau b}[L] + \boldsymbol{tb}[L]$;
24: $\quad$ **for** $l = L-1 \rightarrow 1$ **do**
25: $\quad\quad \boldsymbol{\tau c}[l] = \max\{\boldsymbol{\tau c}[l+1] + \boldsymbol{tc}[l+1], \boldsymbol{\tau b}[l] + \boldsymbol{tb}[l]\}$;
26: $\quad$ Return $\boldsymbol{\tau c}$;

The algorithm first (line 1-7) initializes the layer-wise backward computation cost and communication cost according to Eq. 18 and Eq. 17 respectively with system settings and bechmarks in the first several iterations. Then (line 8-11) the layer-wise start time of backward computation, which is not changed in its followed computation, is calculated based on Eq. 19, and then iteratively compute the layer-wise start time of communication based on the formula of Eq. 20. After that (line 12-16), the merged-gradient layers are found according to Eq. 27, in which if there is a layer found as a merged-gradient layer, the communication time of its previous layer should be updated according to Eq. 14, Eq. 15 and Eq. 16.

The proposed algorithm has a time complexity of $O(L^2)$. For a merged-gradient layer, the algorithm needs to re-calculate the start time of communication of each layer, which is an $O(L)$ search, and it has maximal $L-1$ merged-gradient layers, so the time complexity of the algorithm is $O(L^2)$. Since the algorithm is a one-off calculation at the beginning of the training and it needs not be re-calculated during the training process, so the overhead of finding $\mathbb{M}$ has no impact on the training performance.

Algorithm 2 MG-WFBP S-SGD

Input: $\boldsymbol{D} = [\{X_1, y_1\}, ..., \{X_n, y_n\}]$, I, net, N, bs
Output: $\boldsymbol{W} = [W^{(1)}, W^{(2)}, ...W^{(L)}]$

1: **for** $k = 1 \rightarrow N$ **do**
2: $\quad$ Initialize shared and synchronized queue Q;
3: $\quad$ Obtain the parameter size $\boldsymbol{p}$ from net;
4: $\quad$ Allocate memories $\boldsymbol{W}$;
5: $\quad$ Initialize $\boldsymbol{W}$ in all accelerators;
6: $\quad$ Get $\mathbb{M}$ from Algorithm 1;
7: $\quad$ AsyncHandleComputation($Q, \mathbb{M}$);
8: $\quad$ **for** $i = 1 \rightarrow I$ **do**
9: $\quad\quad di = (i * bs * N)\%n + (k-1) * bs$;
10: $\quad\quad d = D[di : di + bs]$;
11: $\quad\quad$ AsyncHandleComputation(Q, d, L);
12: $\quad\quad$ WaitForLastCommunicationFinished();
13: $\quad\quad \boldsymbol{W} = \boldsymbol{W} - \eta \cdot \nabla \boldsymbol{W}$,
14: NotifyFinished(); // Set $isRunning$ to false
15: **procedure** AsyncHandleComputation(Q, d, L)
16: $\quad o = d$;
17: $\quad$ **for** $l = 1 \rightarrow L$ **do**
18: $\quad\quad o$=FeedForward(l, o);
19: $\quad$ **for** $l = L \rightarrow 1$ **do**
20: $\quad\quad$ BackwardPropagation(l);
21: $\quad\quad Q$.push(l);
22: **procedure** AsyncHandleCommunication($Q, \mathbb{M}$)
23: $\quad$ Initialize lb; // layerBuffer
24: $\quad$ **while** *isRunning* **do**
25: $\quad\quad l = Q$.pop();
26: $\quad\quad lb$.push(l);
27: $\quad\quad$ **if** $l \notin \mathbb{M}$ **then**
28: $\quad\quad\quad$ SynchonizedAllReduce($lb[0], count$);
29: $\quad\quad$ **if** $l = 1$ **then**
30: $\quad\quad\quad$ NotifyLastCommunicationFinished();

We denote the WFBP algorithm integrated with the solution $\mathbb{M}$ as MG-WFBP. In MG-WFBP, the merged-gradient layers should be communicated with their previous layers. As a result, MG-WFBP achieves the minimal iteration time of S-SGD under known DNNs and system configurations. The algorithm of MG-WFBP S-SGD is shown in Algorithm 2. For each worker, the algorithm first (line 2-6) initializes related variables and calculate $\mathbb{M}$ by using Algorithm 1. Then (line 7,

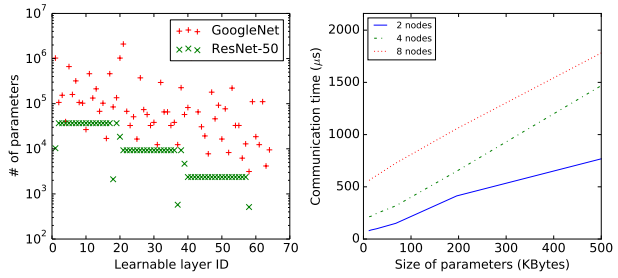
24-30) it reads the layer number from the shared queue Q and decides whether its gradients should be communicated. After that (line 9-13), it starts the loop of iteration, and iteratively reads data (line 16-21) to do feed forward operations and backward propagation followed by pushing the layer number into the shared queue. Finally, the algorithm notifies a message of *isRunning=false* to finish training.

V. Evaluation

We evaluate the performance of MG-WFBP by real experiments on an 8-node GPU cluster with 10GbE, and also by simulations on larger clusters with up to 64 nodes. Two popular CNNs, namely GoogleNet [20] with a batch size of 64 and ResNet-50 [13] with a batch size of 32, are chosen to test the performance of distributed training with ImageNet dataset ILSVRC-2012 [26] which includes about 1.28 million training images of 1000 categories. GoogleNet has about 13 millions of parameters, while ResNet-50 has about 25.5 millions. All parameters and gradients are stored as 32-bit single precision floating point numbers.

A. Statistic data

To verify the communication model in Eq. 17 empirically, we first present some foregone results including the distribution of layer-wise gradient sizes of evaluated CNNs, and the time of the all-reduce operation all-reduce in a 10Gbps Ethernet (10GbE) using OpenMPI v3.1.1. The distribution of gradient size is shown in Fig. 5(a), which shows that the number of parameters of each layer is mainly located in the range of $[10^2, 5\times10^6]$. The measured time of all-reduce under the 10GbE interconnect cluster are shown in Fig. 5(b). Take the size of parameters ($4p$ in floating points) as the variable, we can see that the startup overheads (i.e., $2(N-1)\times\alpha$ in the ring-based all-reduce algorithm) are $90.52\mu s$, $271.56\mu s$ and $633.64\mu s$ in 2-, 4- and 8-node clusters with the 10GbE interconnect respectively.



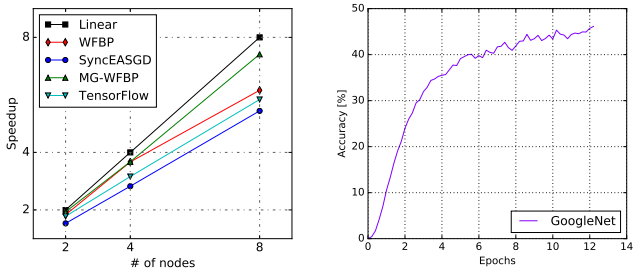
(a) (b)

Fig. 5. (a) The distribution of layer-wise gradient size of two neural networks. (b) The communication time of the all-reduce along with the size of parameters with the prediction and the measurement.

B. Real-world experiments

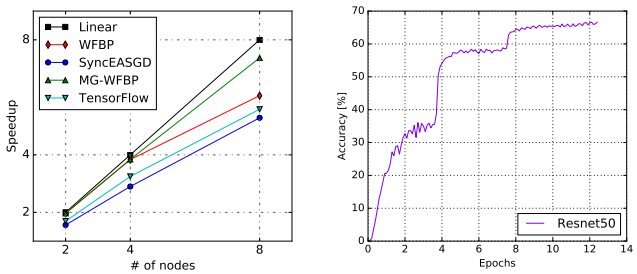
We integrate WFBP [12][14], single-layer communication Sync EASGD (SyncEASGD) [19] and our proposed MG-WFBP into B-Caffe[2], and test the performance across an 8-node GPU cluster with 10GbE. We also compare the scaling efficiencies with TensorFlow. Each server has one Nvidia Tesla K80 card (i.e., 2 GPUs). The OS system is CentOS 7.2, and the major software libraries include CUDA-8.0, cuDNNv6 and NCCL. The compared TensorFlow is at v1.3, and it uses parameter servers to do S-SGD using the official benchmark script[3].

In the real-world experiments, we run 13 epochs to verify the convergence of the CNN training, in which 50000 images are used to test the top-1 accuracies.



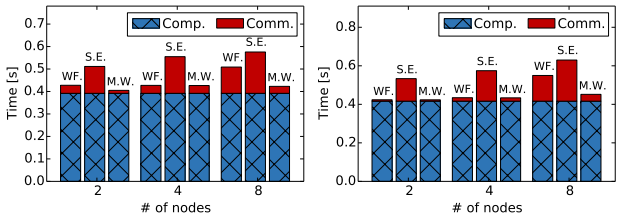
(a) Speedup (b) Top-1 validation accuracy

Fig. 6. The performance of GoogleNet on the K80 cluster connected with 10GbE. Baseline of the speedup of SGD is on a single machine with 2 GPUs.



(a) Speedup (b) Top-1 validation accuracy

Fig. 7. The performance of ResNet-50 on the K80 cluster connected with 10GbE. Baseline of the speedup of SGD is on a single machine with 2 GPUs.



(a) GoogleNet (b) ResNet-50

Fig. 8. Time costs of non-overlapped communication and computation. 'WF.', 'S.E.' and 'M.W.' indicate WFBP, SyncEASGD and MG-WFBP algorithms respectively. 'Comp.' refers to the computation cost (i.e., $t_f + t_b$), and 'Comm.' refers to the non-overlapped communication cost (i.e., t_c^{no}).

The experimental results of GoogleNet and ResNet-50 in the K80 cluster are shown in Fig. 6 and Fig. 7 respectively. The non-overlapped communication cost compared to the computation time is shown in Fig. 8. The baseline is the iteration

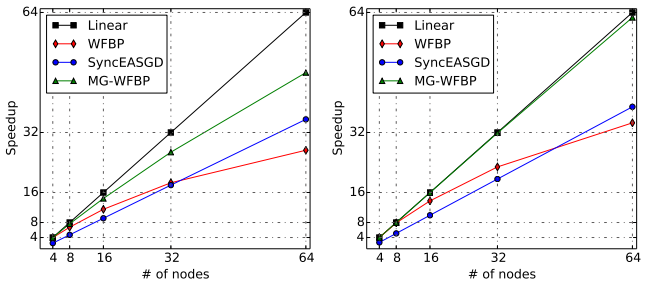
[2]B-Caffe is an optimized distributed deep learning framework based on Caffe [27].

[3]https://github.com/tensorflow/benchmarks

throughput of two GPUs in a single machine, in which no communication via Ethernet is required. And the speedup of throughput on multiple nodes are compared to the baseline. From Fig. 8, we can observe that for both GoogleNet and ResNet, MG-WFBP performs better than WFBP, SyncEASGD and TensorFlow. SyncEASGD dose not overlap the communication with computation; and hence the communication cost increases when the number of nodes increases. As a consequence, the scaling efficiency of SyncEASGD is poor. WFBP achieves near linear scaling on 2 and 4 nodes, in which the non-overlapped communication overhead are small. When scaling to 8 nodes, however, WFBP has an obvious drop in efficiency due to the increased startup time (a larger number of nodes results in higher startup time according to Table II) of layer-wise communication which cannot be totally hidden by computation. Regarding the performance of TensorFlow, it uses parameter servers to do the model aggregation. On one hand, the centralized parameter server based algorithm could easily suffer a bandwidth pressure in the parameter server on the lower speed network [14]. On the other hand, it takes two communication directions (workers to PS, and PS to workers) to finish the model synchronization, which introduces more overhead in the synchronization pass. Therefore, though TensorFlow exploits the WFBP technique, the PS-based method performs worse than the decentralized method. Our proposed algorithm has a very small non-overlapped communication cost even on the 8-node cluster, so the scaling efficiency is still close to linear. In summary, MG-WFBP achieves about 1.2x and 1.36x speedups compared to WFBP and SyncEASGD respectively on the 8-node K80 cluster on both GoogleNet and ResNet-50.

C. Simulation

Due to the hardware limitation, we do not have a very large GPU cluster to support more large-scale experiments. So we conduct simulations based on the real single-GPU performance and the network performance model. Based on the measured layer-wise backward propagation time on the real K80 GPU[4], we simulate WFBP, SyncEASGD and MG-WFBP by scaling from 4 nodes to 64 nodes.



(a) GoogleNet (b) ResNet-50

Fig. 9. The performance comparison on the simulated K80 cluster connected with 10GbE. Baseline of the speedup of SGD is on a single K80.

[4]One can model the performance of computation according to the layer configuration and hardware [7], but it is more accurate to measure it in real-world applications.

Overall Performance. We simulate to train GoogleNet and ResNet-50 by scaling from 4 nodes to 64 nodes. The scaling performances are shown in Fig. 9. On the cluster with K80 GPUs, our proposed algorithm MG-WFBP achieves the best speedup. On the 64-node cluster, MG-WFBP outperforms WFBP and SyncEASGD by 1.78x and 1.35x, respectively on GoogleNet. On ResNet-50, MG-WFBP performs almost linear speedup, while WFBP and SyncEASGD only have around 55% scaling efficiency in the 64-node cluster. It is important to notice that the lines of WFBP and SyncEASGD have a crossing point in Fig. 9. This is because the two algorithms are sub-optimal in utilizing the network bandwidth; when the computation has the opportunity to overlap with communication, and the startup time of network communication is not that large (e.g., 4-16 nodes in the K80 cluster), then WFBP would have the advantage to hide the communication compared to SyncEASGD. But when scaling to large number of nodes (e.g., 64 nodes), the startup time of communication becomes much larger so that it is hard to be hidden, then using a single-layer communication could become a better approach. As we can see, SyncEASGD achieves better scaling efficiency than WFBP in the 64-node cluster on both tested CNNs. MG-WFBP not only overlaps the communication with computation, but it also finds the optimal communication message size. So it achieves better scaling efficiency than SyncEASGD and WFBP. Finally, on training ResNet-50, MG-WFBP achieves about 1.75x and 1.45x speedups compared to WFBP and SyncEASGD respectively on the simulated 64-node K80 cluster.

VI. Related Work

Current distributed training systems [19][28][29] exploit tensor fusion that merges small size of gradients before communicating across workers to reduce the communication overhead. The parameter server (PS) method [30] is proposed for parallelism between computation and communication, but it easily suffers from the communication traffic jam since PS needs to collect the gradients from all the workers. Sufficient factor broadcasting (SFB) [14] uses the matrix factorization technique to reduce the volume of the data that needs to be communicated. Zhang et al. [14] proposed Poseidon system with hybrid communication of PS and SFB combined with the WFBP algorithm. Unfortunately, the optimal pipeline between backward propagation and gradient communication is not achieved.

In the HPC community, the MPI data communication collectives have been redesigned for distributed training to improve the communication performance across multiple machines [12]. Many MPI-like implementations, such as OpenMPI[5], NCCL2[6], Gloo[7] and MVAPICH2-GDR[8], support efficient CUDA-aware communication between GPUs via network, and many state-of-the-art deep learning frameworks (e.g.,

[5]https://www.open-mpi.org/
[6]https://developer.nvidia.com/nccl
[7]https://github.com/facebookincubator/gloo
[8]https://mvapich.cse.ohio-state.edu/

TensorFlow, Caffe2 and CNTK) integrate NCCL2 or Gloo for their distributed training modules. Even though these libraries provide very efficient communication collectives, the data communication would still become bottleneck when the communication-to-computation ratio is high, and S-SGD does not scale very well.

VII. Conclusion

In this work, we first show that existing state-of-the-art communication strategies, say wait-free backward propagation (WFBP) and single-layer communication (SyncEASGD), are sub-optimal in the distributed SGD training of deep learning when the communication-to-computation ratio is high. Then we generalize the communication problem as an optimization problem and develop an efficient optimal solution. We then propose the MG-WFBP strategy and implement it in our open-source platform B-Caffe. MG-WFBP achieves a better scalability than WFBP and SyncEASGD in our tested experiments with two popular CNNs (GoogleNet and ResNet-50) across real-world and simulated 10GbE GPU clusters.

VIII. Acknowledgement

The research was supported in part by Hong Kong RGC GRF grants under the contracts HKBU 12200418, HKUST 16206417 and 16207818, a RGC CRF grant under the contract C7036-15G.

References

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

[2] 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.

[3] P. Watcharapichat, V. L. Morales, R. C. Fernandez, and P. Pietzuch, "Ako: Decentralised deep learning with partial gradient exchange," in *Proceedings of the Seventh ACM Symposium on Cloud Computing*. ACM, 2016, pp. 84–97.

[4] H. Cui, H. Zhang, G. R. Ganger, P. B. Gibbons, and E. P. Xing, "Geeps: Scalable deep learning on distributed GPUs with a GPU-specialized parameter server," in *Proceedings of the Eleventh European Conference on Computer Systems*. ACM, 2016, p. 4.

[5] 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 *INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019.

[6] 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*, 2018.

[7] 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. 1508–1518.

[8] 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," *arXiv preprint arXiv:1901.04359*, 2019.

[9] S. Potluri, K. Hamidouche, A. Venkatesh, D. Bureddy, and D. K. Panda, "Efficient inter-node MPI communication using GPUDirect RDMA for InfiniBand clusters with Nvidia GPUs," in *Parallel Processing (ICPP), 2013 42nd International Conference on*. IEEE, 2013, pp. 80–89.

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

[11] M. Bayatpour, S. Chakraborty, H. Subramoni, X. Lu, and D. K. Panda, "Scalable reduction collectives with data partitioning-based multi-leader design," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2017, p. 64.

[12] 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 *Proceedings of the 22nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. ACM, 2017, pp. 193–205.

[13] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

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

[15] S. Shi, Q. Wang, and X. Chu, "Performance modeling and evaluation of distributed deep learning frameworks on GPUs," in *2018 IEEE DataCom*. IEEE, 2018, pp. 949–957.

[16] S. Shi, X. Chu, and B. Li, "A DAG model of synchronous stochastic gradient descent in distributed deep learning," in *Parallel and Distributed Systems (ICPADS), 2018 IEEE 24rd International Conference*. IEEE, 2018.

[17] M. Handley, C. Raiciu, A. Agache, A. Voinescu, A. W. Moore, G. Antichi, and M. Wójcik, "Re-architecting datacenter networks and stacks for low latency and high performance," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 2017, pp. 29–42.

[18] C. Guo, H. Wu, Z. Deng, G. Soni, J. Ye, J. Padhye, and M. Lipshteyn, "RDMA over commodity ethernet at scale," in *Proceedings of the 2016 ACM SIGCOMM Conference*. ACM, 2016, pp. 202–215.

[19] Y. You, A. Buluç, and J. Demmel, "Scaling deep learning on GPU and Knights Landing clusters," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2017, p. 9.

[20] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich *et al.*, "Going deeper with convolutions." Cvpr, 2015.

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

[22] 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.

[23] T. Hoefler, W. Gropp, R. Thakur, and J. L. Träff, "Toward performance models of MPI implementations for understanding application scaling issues," in *European MPI Users' Group Meeting*. Springer, 2010, pp. 21–30.

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

[25] H. Qi, E. R. Sparks, and A. Talwalkar, "Paleo: A performance model for deep neural networks," in *International Conference on Learning Representations*, 2017, p. 10.

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

[27] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *arXiv preprint arXiv:1408.5093*, 2014.

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

[29] 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," *Workshop on Systems for ML and Open Source Software, collocated with NeurIPS*, 2018.

[30] M. Li, D. G. Andersen, A. J. Smola, and K. Yu, "Communication efficient distributed machine learning with the parameter server," in *Advances in Neural Information Processing Systems*, 2014, pp. 19–27.