2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing

# Adaptive Communication for Distributed Deep Learning on Commodity GPU Cluster

Li-Yung Ho
Institute of Information Science,
Academia Sinica
Taipei, Taiwan
Email: lyho@iis.sinica.edu.tw

Jan-Jan Wu
Institute of Information Science,
Research Center for Information
Technology Innovation,
Academia Sinica
Taipei, Taiwan
Email: wuj@iis.sinica.edu.tw

Pangfeng Liu
Department of Computer Science and
Information Engineering,
Graduate Institute of
Networking and Multimedia,
National Taiwan University
Taipei, Taiwan
Email: pangfeng@csie.ntu.edu.tw

***Abstract*—Deep learning is now the most promising approach to develop human-intelligent computer systems. To speedup the development of neural networks, researchers have designed many distributed learning algorithms to facilitate the training process. In these algorithms, people use a constant to indicate the communication period for model/gradient exchange. We find that this type of communication pattern could incur unnecessary and inefficient data transmission for some training methods e.g., elastic SGD and gossiping SGD. In this paper, we propose an adaptive communication method to improve the performance of gossiping SGD. Instead of using a fixed period for model exchange, we exchange the models with other machines according to the change of the local model. This makes the communication more efficient and thus improves the performance. The experiment results show that our method reduces the communication traffic by 92%, which results in 52% reduction in training time while preserving the prediction accuracy compared with gossiping SGD.**

***Keywords*—deep learning, distributed learning, stochastic gradient descent, adaptive communication**

## I. Introduction

Deep neural networks (DNNs) have been proved to be one of the most promising approaches to achieve human-intelligence computer systems. For example, AlphaGo Zero [1] using reinforcement learning with DNNs becomes the most powerful Go player ever without the assistance of human knowledge. GoogleNet [2] achieves super-human image recognition ability with less than 5% error on ImageNet [3] dataset. With these success stories, people now investigate the possibility of DNNs in various domains including self-driving car [4], [5], language translation [6], [7] and recommendation system [8], [9].

Although DNNs achieve significant success in many domains, training a DNN is not trivial. During a training, people may encounter issues such as divergence and under/overfitting. One may need to adjust and train the model over and over again to achieve high accuracy. However, training a DNN is very time-consuming in general, usually taking days to weeks on a commodity computing cluster [10]. Therefore, how to reduce the training time while maintaining the accuracy becomes one of the most important issues in deep learning.

Distributed learning is one approach to accelerate the training process. There are two types of parallelism – model parallelism and data parallelism. In model parallelism, the parameters are partitioned and each partition is assigned to a worker for training. In data parallelism, every worker duplicates the model and trains a partition of the data. These workers then exchange gradients/parameters to reach a consensus of the model. It has been reported in the literatures [10] that such communication becomes the main bottleneck during the learning process. In this paper, we focus on how to reduce such communication overhead in data-parallel distributed learning.

Researchers have proposed many data-parallel learning algorithms for distributed learning [10], [11], [12], [13]. Elastic stochastic gradient descent (ESGD) [12] is one of the most popular and promising methods. In ESGD, a centralized *parameter server* maintains an "averaged" model among the workers. That is, during the communication, a worker sends its local updated parameters to the server, and the server averages and stores the received parameters. The averaged parameters are then sent back to the workers for further training.

Gossiping stochastic gradient descent (GSGD) [13] is a decentralized version of ESGD. Instead of using a centralized parameter server to maintain an averaged model, during the communication period, ESGD randomly pulls parameters from workers to *approximate* a global averaged model. That is, each worker pulls parameters from other workers and averages them with the local parameters. The difference of communication pattern between ESGD and GSGD is shown in Figure 1.

In both ESGD and GSGD, the amount of communication traffic is affected by the *frequency* of the model exchange. In the original setting, this frequency is determined by a user-defined, fixed parameter $\tau$. As shown in Algorithm 2 of GSGD, we use $\tau$ to control the frequency of model exchange, i.e., a worker pulls parameters from other workers *every* $\tau$ iterations. The value usually ranges from 1 to 10 [12], which may incur significant communication traffic for large models, e.g., AlexNet [14]. In this paper, we aim to investigate a more sophisticated way to determine the value of $\tau$.

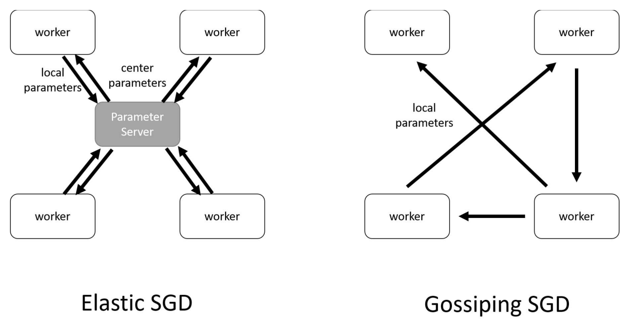

Fig. 1: The communication pattern of ESGD and GSGD

We observe that, the communication should be driven according to the *difference* between the averaged models in two consecutive iterations, instead of a fix number of iterations. As shown in line 5 of Algorithm 2, if the model $x^j$, which is retrieved from other machine, does not vary a lot, we can just use the previously received $x^j$, which is already cached in the local to *approximate* the latest model in other machine. On the other hand, a worker must *notify* other workers if its model ($x^i$ in line 5 of Algorithm 2) has changed significantly after the gradient descent computation.

With this observation in mind, we first investigate how a model changes in a training process. We measure the *delta* of a model during training, which intuitively measures the difference of a model before and after the gradient update. We find that, the delta will *converge* as the model and the accuracy of the model both converge.

According to this investigation, we propose an **adaptive communication** model. In this model, the communication is driven by the *difference of delta* instead of a fix number of iteration. The difference of delta is defined as the difference between the delta of the last communication period and the delta of the current iteration. This is motivated by the fact that if the delta is stable, the model is already stable, therefore, workers do not have to communicate aggressively to exchange model. The experimental results show that our proposed method reduces communication by 92% on average compared with the fix number method, thus accelerates the training process significantly. At the same time, our method also maintains the model accuracy.

In summary, this paper has the following contributions.

- We discover that model delta converges during a training process, and based on this finding, we propose an adaptive communication model to significantly reduce the amount of communication than the existing methods.
- We implement the gossiping stochastic gradient descent algorithm with our proposed adaptive communication method based on Caffe [15].
- We conduct extensive experiments to demonstrate the effectiveness of the adaptive communication method. Experimental results indicate that our proposed method not only reduces the amount of communication, which in turn improves the performance greatly, but also maintains the model accuracy.

The rest of the paper is organized as follows. Section II discusses the related works. Section III introduces preliminary knowledge for this paper. Section IV describes the system architecture of our implementation of gossiping SGD with the adaptive communication model. Section V presents the details of the proposed adaptive communication method. We conduct experiments to evaluate the proposed method, and discuss the results in Section VI. We finally conclude this paper in Section VII.

## II. Related Works

Researchers have developed many parallel stochastic gradient descent methods in the literatures. They can be categorized into two groups – *synchronous* SGD and *asynchronous* SGD. For the *asynchronous* group, we further categorize them into *centralized* or *decentralized* architecture. Note that we focus on parallel SGD methods in *data parallel* training, so model parallel training is out of the scope of this paper.

### A. Synchronous SGD

In synchronous SGD method, the workers synchronize with each other in *every* training iteration. In the data parallel setting, the entire training dataset is partitioned and distributed to all workers for training, that is, each worker trains a subset of the whole training data. In each iteration, the worker calculates the gradients with forward and backward computation using a batch of training data. Then, all workers *synchronize* to wait for everyone to finish the gradient computation. Once all workers are done, they *broadcast* the computed gradients to all other workers. A worker averages the received gradients with locally computed ones into the final gradients, and then uses them to update the model parameters.

Due to the synchronization, synchronous SGD suffers from the problem of *straggler*. That is, it needs to wait for the *slowest* workers to complete the computation. To address this problem, Chen et al. [16] and Zhang et al. [17] proposed *backup* workers [18] to alleviate the affect of stragglers. Instead of using only $N$ workers for training, they add $b$ extra workers. During the synchronization, as long as a worker receives gradients from $N$ workers, it stops waiting and updates the local parameters according to the $N$ received gradients. As a result, the slowest $b$ workers' gradients will be dropped when they arrive.

Das et al. [19] developed a system for distributed synchronous SGD. They analyzed and optimized the system by designing and solving detailed system balance equations for specified neural networks. They also provided the guideline to find the optimal design for Xeon-based CPU training framework. They demonstrated that deep learning training can be performed at scale using synchronous SGD at high throughput on CPUs.

## B. Asynchronous SGD

Unlike synchronous SGD, in asynchronous SGD, the workers do *not* wait for each other in every iteration. This results in fast training progress because there is no need to wait for the slowest workers. However, the models become inconsistent among the workers since there is no synchronization. Due to this inconsistence, the gradients computed by one worker may be *stale* to another worker. As a result, the gradients computed by other workers may point to the wrong direction and result in slow convergence speed.

*1) centralized architecture:* In this type of asynchronous SGD, we use a *parameter server* to maintain a consensus of model among the workers. Each worker retrieves the model from the parameter server, computes the gradients and sends the computed gradients to the parameter server. The parameter server aggregates the received gradients from workers and updates the central model accordingly. The gradients from the workers may not be computed in the same iteration. A parameter server use the *staleness* to decide the proper iteration number for each worker in the computation of the gradients. This processing model is also called *Downpour SGD* [10].

Cui et al. proposed a GPU-specialized parameter server GeePS [11]. They found that the overhead of data movement between CPU and GPU and the limit of GPU memory incur GPU stall and inefficient training. To address the problem, they design GPU-friendly memory management techniques including data staging, and caching to overlap the computation and data movement, so that the GPU can be fully utilized to improve training performance.

Instead of aggregating gradients and updating a central model in the parameter server, the parameter server in Elastic SGD [12] averages the *models* of the workers. The averaged model is then sent back to the workers for further training. The idea of Elastic SGD is allowing workers to do more *exploration* for the local optima. However, it uses an *elastic force* between the local and central model to control the exploration, so that each worker can finally reach a consensus.

*2) decentralized architecture:* The drawback of centralized architecture is that the parameter server easily becomes the communication bottleneck. Instead of using a parameter server to maintain a central model, the *decentralized* framework randomly selects models from other workers to *approximate* a consensus model. This prevents a centralized server from a communication hot spot. During the communication period, a worker retrieves the model from some other randomly selected workers. The worker then takes average of them to obtain an approximated central model for further gradient computation.

Lian et al. [20] gave the first theoretical analysis that decentralized SGD may outperform its centralized counterpart. This is from the observation that decentralized SGD has comparable computational complexity to the centralized version but requires much less communication cost on the busiest node.

Jin et al. [13] developed Gossiping SGD, a decentralized version of Elastic SGD [12]. They proposed a *pull* and a *push* model for the model exchange. For the pull mode, each worker pulls the model from one randomly selected worker. For the push mode, each worker pushes its model to a randomly selected worker, while receiving zero or multiple models.

## C. Summary

In this paper, we propose a novel approach to control the communication in Gossiping SGD. *All* above-mentioned methods use a *fix* communication period to periodically communicate with other machines during the training. We propose an *adaptive* approach to decide the timing for communication, which reduces the communication traffic significantly while preserving the prediction accuracy.

## III. Preliminary

In this section we introduce the background knowledge of our system and approach. We will give an overview of the system architecture of Caffe [15], elastic stochastic gradient descent method [12] and gossiping gradient descent method [13].

## A. Caffe

Caffe [15] is a popular deep learning framework. With modularity and extensible code design, Caffe users are able to easily customize their neural network model and training processes. Caffe also supports both CPU and GPU, so users can utilize GPU for high performance training.

The architecture of Caffe consists of three main components – *solver*, *layer* and *blob*. The *solver* defines the method to optimize the objective when training a neural network, e.g., gradient descent or adagrad [21]. Typically, the solver is defined as a while loop, which performs a series of operations iteratively. The series of operations are *feedforward*, *backpropagation* and *applyupdate*. The feedforward operation performs inference from the first to the last layer to obtain the value of objective using a batch of training data. The backpropagation propagates the difference of objective function from the last to the first layer to calculate the gradients of parameters of each layer. After a feedforward and a backpropagation phase, the solver performs applyupdate to update the model parameters using the gradients.

The *layer* defines how a layer perform forward and backward computation. For a particular kind of layer, e.g., convolution layer or fully connected layer, users inherit the abstract layer class and implement the corresponding foward and backward virtual function for both CPU and GPU. Users also need to define the input data format of a layer by implementing a reshape function. The reshape function reshapes the input data to fit the input format of forward function of a layer.

A Blob is a multi-dimensional array to store the gradients and parameters of a layer in Caffe. The Blob also synchronizes the data between CPU and GPU. Users can easily call the functions ToCPU/ToGPU to move the data in a Blob to CPU or GPU respectively. The Blob also maintains a flag to indicate whether the data in CPU or GPU is the most recent in order to prevent unnecessary data movement.

## B. Elastic stochastic gradient descent

Zhang et al. proposed the elastic stochastic gradient descent (ESGD) method [12] for data-parallel distributed learning. In ESGD, a master (parameter server) and a set of workers train the model collectively. The parameters stored in the master is called *center parameters*. The basic idea of ESGD is to allow local workers to perform more *exploration* and master to perform *exploitation* when searching local optima in gradient descent.

In ESGD, each worker maintains and updates its own local parameters. The communication and coordination among workers are based on an *elastic force* that links the local parameters with center parameters stored on the parameter server. By controlling the elastic force, ESGD enables the local workers to explore more possible solutions by fluctuating the local parameters further from the center parameters to reduce the communication between the local workers and the master. Due to the existence of multiple local optima, more exploration can lead to a better solution.

ESGD updates local and center parameters are shown in Equation 1 and 2 respectively.

$$x^i_{t+1} = x^i_t - \eta g^i_t(x^i_t) - \alpha(x^i_t - \tilde{x}_t) \quad (1)$$

$$\tilde{x}_{t+1} = (1-\beta)\tilde{x}_t + \beta(\frac{1}{p}\sum_{i=1}^{p} x^i_t) \quad (2)$$

$x^i$ and $\tilde{x}$ denote the local parameter on worker $i$ and the center parameter on the master respectively. $g^i_t(x^i_t)$ denotes the gradient with respect to $x^i$ at iteration $t$ and $\eta$ is the learning rate. $\alpha$ in Equation 1 represents the amount of exploration allowed in the model. In particular, small $\alpha$ allows more exploration because it allows $x^i$ to fluctuate further from the center parameter. Equation 2 indicates that ESGD updates the center parameter as a moving average taken both in time (last iteration) and in space (over the parameters computed by local workers).

Algorithm 1 illustrates how ESGD works. For a worker $i$, if it is in a communication period (line 4), it requests the center parameters from the master and calculates the *elastic difference* defined as $\alpha(x - \tilde{x})$ (line 5). The elastic difference is sent back to the master to update the center parameters (line 6). If it is not in a communication phase, the worker $i$ performs normal gradient descent update (line 8).

## C. Gossiping stochastic gradient descent

Gossiping stochastic gradient descent [13] (GSGD) is a decentralized version of ESGD. Instead of using a centralized parameter server to maintain the center parameters, GSGD uses a gossiping communication pattern to aggregate parameters from other workers to approximate the *center parameters*. The center parameters are defined as the averaged model among the workers. GSGD replaces the term $\frac{1}{p}\sum_{i=1}^{p} x^i_t$ in Equation 2 with an unbiased one-node estimator $x^{j_{t,i}}_t$. That is, it randomly selects a remote worker and retrieves its parameters to approximate the center parameter. Therefore,

**Algorithm 1** Elastic SGD: Processing by worker $i$ and the master

```
Require: learning rate $\eta$, moving rate $\alpha$, communication
    period $\tau$, maximum iteration $M$
 1: initial $\tilde{x}$ randomly, $x = \tilde{x}$, $t = 0$
 2: repeat
 3:   $x = x^i$
 4:   if $\tau$ divides $t$ then
 5:     $x^i = x^i - \alpha(x - \tilde{x})$
 6:     $\tilde{x} = \tilde{x} + \alpha(x - \tilde{x})$
 7:   end if
 8:   $x^i = x^i - \eta g^i_{t_i}(x)$
 9:   $t^i = t^i + 1$
10: until $t > M$
```

the update rule of GSGD can be written as Equation 3 and Equation 4.

$$x'_t = x_t - \eta g_t(x_t) \quad (3)$$

$$x_{t+1} = (1-\beta)x'_t + \beta(x'_t - x'_{j_{t,i},t}) \quad (4)$$

Algorithm 2 is the pseudo code of GSGD. Each worker performs the same function. If the current iteration is communication phase (line 3), a worker retrieves the parameters from a remote machine $j$ and averages with the local parameters (line 5). Otherwise, it performs normal gradient descent updates (line 7).

**Algorithm 2** Gossiping SGD: Processing by worker $i$

```
Require: learning rate $\eta$, communication period $\tau$, maximum
    iteration $M$
 1: initial $x^i$ randomly, $t = 0$
 2: repeat
 3:   if $\tau$ divides $t$ then
 4:     choose a target $j$
 5:     $x^i = \frac{x^i + x^j}{2}$
 6:   end if
 7:   $x^i = x^i - \eta g^i_{t_i}(x^i)$
 8:   $t^i = t^i + 1$
 9: until $t > M$
```

## IV. Architecture

To demonstrate the efficiency of our proposed method, we develop a distributed deep learning system based on Caffe. We implement the GSGD algorithm and the proposed adaptive communication method by adding two additional components in Caffe – a *communication manager* and a *synchronization manager*. The communication manager receives/transfers the parameters from/to other machines. The synchronization manager decides when to retrieve the parameters from other machines.

Figure 2 illustrates the interaction among the components in our system. When training a neural network, we first initialize the *layers* and the corresponding *Blobs* which store the parameters and gradient data according to the network definition. A *solver* then trains the network iteratively. After we train a batch of data, the *synchronization manager* decides whether or not to retrieve the parameters from other machines, depending on the difference between the updated model in this iteration and the final model in the last iteration. If the synchronization manager decides to retrieve parameters from other machines, it notifies the *communication manager* to retrieve the parameters. Otherwise, it performs gradient update to the local model.

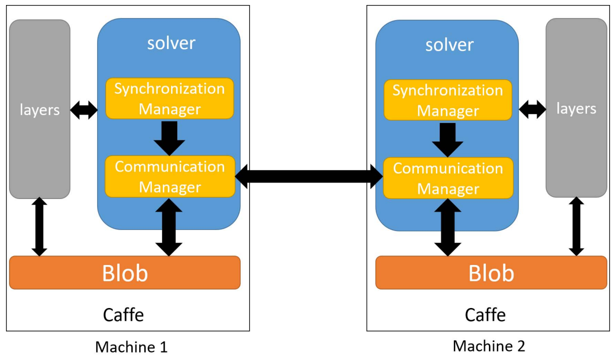

Fig. 2: Distributed deep learning system

We implement the communication manager by socket to achieve high performance. When a synchronization manager decides to retrieve the parameters from other machines, the communication manager sends a *pullModel* request to a randomly selected remote communication manager. Once the remote communication manager receives the request, it copies its parameters from the Blob to a buffer. The Blob returns a pointer to indicate the address of parameters in the CPU memory by using the toCPU function. The communication manager then sends the parameters stored in the buffer to the requester. When the requester receives the parameters, it copies the parameters to its GPU memory and averages the received parameters with local ones. Finally, the communication manager replaces the parameters stored in the Blob with the averaged results.

The synchronization manager decides whether or not to retrieve the parameters from other machines according to a *difference of local model*. We will later formally define it in Section V. The synchronization manager calculates the difference of local model in every iteration. If the difference exceeds a certain threshold, the synchronization manager triggers the communication manager to perform model retrieval. It is easy for users to customize the synchronization manager to apply different retrieval policy. For example, instead of using a certain threshold, users can use a fixed period, or a probability distribution to decide when to retrieve the model from other machines.

## V. Adaptive communication

In this section, we describe how we determine the timing to retrieve models from other workers in gossiping SGD method [13]. The main idea is to use a metric to evaluate the change of the local model. If the local model changes *significantly*, we pull the models from other workers to calculate a new averaged model. Otherwise, we perform normal gradient update on the local model. Now we describe how we measure the difference of a local model.

### A. The delta of a model

We use the *euclidean distance* to measure the difference of a local model before and after an epoch of gradient update. Equation 5 represents how we update the parameters in the gradient descent method, where $x_t$ denotes the model parameters at $t$ iteration and $g_t(x_t)$ is the gradients respect to $x_t$.

$$x_t = x_{t-1} - \eta g_{t-1}(x_{t-1}) \tag{5}$$

We define the *delta* of a model as follows.

$$\begin{aligned} \Delta x_t &= \sqrt{(x_t - x_{t-1})^2} && (6) \\ &= \sqrt{\eta^2 g_{t-1}^2(x_{t-1})} && (7) \end{aligned}$$

In addition to the delta, which indicates the change of a local model, we also define a *distance* which measures the difference between the local and the averaged remote models. Let $x_{t_i}^i$ represents the model of machine $i$ at the $t_i$-th iteration. Suppose we have $N$ machines, for a particular machine $j$ at its iteration $t_j$, the distance between $x_{t_j}^j$ and the averaged remote model is defined as follows.

$$D_{t_j}^j = \sqrt{(x_{t_j}^j - \frac{1}{N}\sum_{i=1}^{N} x_{t_i}^i)^2} \tag{8}$$

We conduct preliminary experiments to investigate how the delta and distance change during a training. We train AlexNet [14] with ImageNet dataset using gossiping SGD method. We run the experiment on four machines with the same computation capability. We set the communication period $\tau$ to be 10 and 50. Therefore, for a particular machine, the distance to the averaged remote models will be calculated every 10/50 iterations, and the delta is aggregated for 10/50 iterations as well.

Figure 3 and Figure 4 depict the value of delta and distance respectively. The x-axis indicates the number of model retrievals. In Figure 3, we first observe that, the higher the $\tau$ is, the larger the delta will be. This is because the delta is always positive, so the more iterations we accumulated, the larger the delta will be. The second observation is that, while counterintuitive, the delta *increases* monotonically. However, it *converges* as the model converges. It seems that as the model converges, the model is ”trapped” in a wide area of the parameter space and fluctuates randomly, so it travels a fixed distance when we fix the accumulated iterations. Figure 4 exhibits the similar behavior for the distance to the averaged remote models. It shows that the distance retains a constant. We believe that this also indicates all the models are in a stable state.

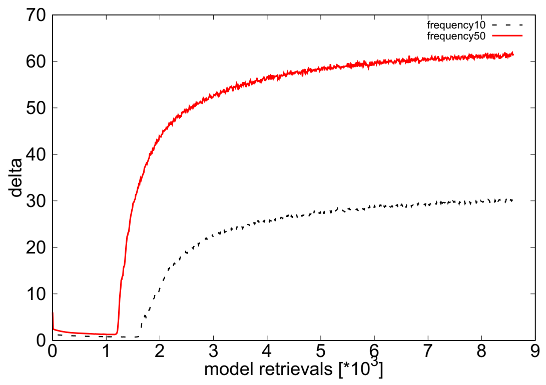

Fig. 3: The delta of local model

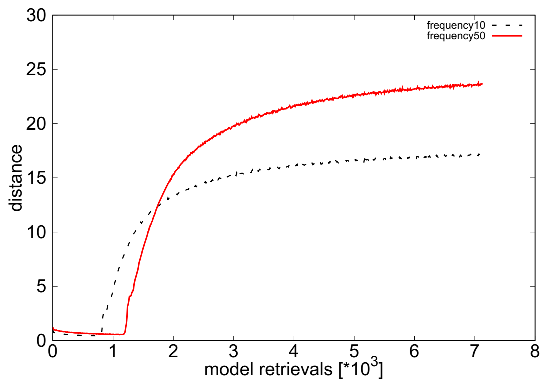

Fig. 4: The distance to averaged remote models

The convergence of delta and distance to averaged remote model gives us the following insight – we may use their *difference* between iterations as an *indicator* to determine whether the model converges. That is, as the difference of delta (or distance to the averaged remote models) becomes small, we know the model is in a stable state.

## B. *Difference of model delta*

Because the delta and the distance to averaged remote models both exhibit the similar convergence behavior, we use the *difference of delta* as an indicator for model convergence. Figure 5 illustrates the difference of delta with respect to the iteration number for AlexNet. As expected, it *decreases* and converges as the model converges.

The difference of delta not only represents the state of a model, but also indicates the *fluctuation* of a model. That is, it represents the *amount of change* for a model state during the training process. Therefore, it is intuitive to use the difference of delta to drive the communication in gossiping SGD. This is because in the communication phase of gossiping SGD, if the

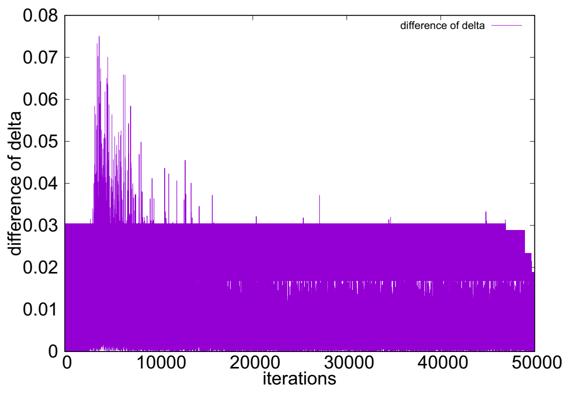

Fig. 5: The difference of delta of the local model

model state does not change dramatically, the averaged model will not change dramatically either. The proposed *adaptive communication* method sets a threshold for the difference of delta. If the difference of delta exceeds the threshold, we pull the models from remote machines to compute a new averaged model. Otherwise, we perform gradient update to the local model. Note that this decision can be made locally, without any communication to the remote machines, so our method can achieve high scalability.

## VI. Experiments

We train three models to evaluate the performance of the proposed method – AlexNet [14], GoogleNet [2] and ConvNet [22]. AlexNet was developed by Alex Krizhevsky et al. [14], which contains 60 million parameters. AlexNet is the first model to prove the efficiency of deep neural network on image recognition. GoogleNet is a neural network for image recognition developed by Google. It improves the performance of AlexNet by using less parameters (11 million parameters). GoogleNet won the ILSVRC classification challenge in 2014. ConvNet is also developed by Alex Krizhevsky for training the CIFAR-10 dataset [23]. It contains about 2 million parameters.

We use the dataset provided by Imagenet Large Scale Visual Recognition Challenge 2012 (ILSVRC-2012) [24] to train AlexNet and GoogleNet. The ConvNet is trained by the dataset CIFAR-10. The training data in ILSVRC-2012 contains 1.2 million images which are categorized into 1,000 groups. The validation date is a random subset of 50,000 images. CIFAR-10 consists of 60,000 images in 10 classes, with 6,000 images per class. Among the 60,000 images, we use 50,000 as training images and 10,000 as validation images.

We compare our method with gossiping SGD, in terms of *training time*, *number of model retrievals* and *prediction accuracy*. For training time, we *fix* the maximum number of iterations in the training process, and measure the elapsed time a particular method needs to complete the training. For the number of model retrievals, we count the number of times a method requesting a model from a remote machine. For prediction accuracy, we measure the top-1 accuracy of the model at the end of the training. In our adaptive communication method, we set the threshold of delta difference as 0.025, 0.02 and 0.01 for AlexNet, GoogleNet and ConvNet respectively. In our experience, the threshold is usually set between 0.01 and 0.1. We set the communication period $\tau = 10$ in gossiping SGD, which is suggested by [12]. Table I lists other parameters used in the training for each model.

TABLE I: Training parameters

| model | AlexNet | GoogleNet | ConvNet |
|---|---|---|---|
| learning rate | 0.01 | 0.01 | 0.001 |
| gamma | 0.1 | 0.96 | 0.1 |
| momentum | 0.9 | 0.9 | 0.9 |
| weight decay | 0.0005 | 0.0002 | 0.004 |
| max iteration | 50,000 | 50,000 | 50,000 |
| threshold | 0.025 | 0.02 | 0.01 |

We conduct the experiments on a commodity GPU cluster with four machines. Each machine has one Intel Core i7 3.5 GHz processor, 16 GB memory and a Nvidia GTX 1080p GPU card. These machines are connected to a switch that has 1 GBit/s bandwidth. We use Caffe version 1.0.0 to develop our system.

Figure 6 illustrates the prediction accuracy of AlexNet. The black line with rectangle (tau10) represents the model trained by gossiping SGD with communication period $\tau = 10$. The red line with circle (adaptive) is the model trained by our adaptive communication method. The blue line with triangle is the model trained by gossiping SGD with $\tau = \infty$. We first observe that gossiping SGD (without communication) has the shortest training time. However, it converges to a model with less accuracy (0.3). On the other hand, gossiping SGD with more frequent communication ($\tau = 10$) increases the accuracy to 0.4. However, it prolongs the training time from 500 to 1440 minutes. Our method adaptively communicates with other machines to retrieve the models only if the local model changes significantly. As a result, we are able to achieve the same accuracy (0.4) in less time (700 minutes), which reduces the training time by 51% and makes the training more efficient.

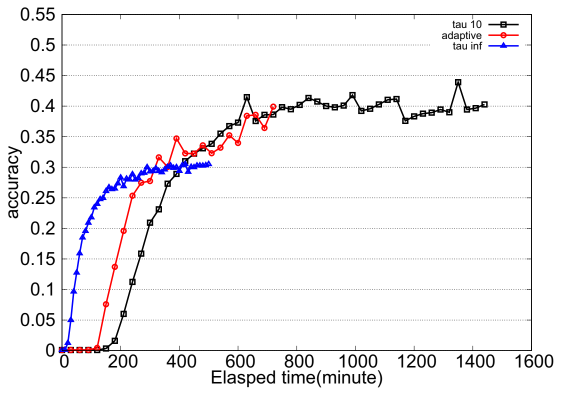

Fig. 6: AlexNet Accruracy

Figure 7 depicts the accumulated number of model retrievals of gossiping SGD with $\tau = 10$ and our adaptive communication method. It shows that we have a *tremendous* reduction in the number of model retrievals. Gossiping SGD with $\tau = 10$ needs 5,000 retrievals for 50,000 iterations in the training. Moreover, the 5,000 retrievals *uniformly* occur among the 50,000 iterations. Our method makes a more sophisticated decision, that is, it frequently communicates with other machines when the model changes significantly (iteration 8,000-12,000). When the model is in a more stable state (after 12,000 iteration), our method *adaptively*, and less often, retrieves the model from others. As a result, we need only 365 model retrievals, which eliminate 92% of communication in gossiping SGD.

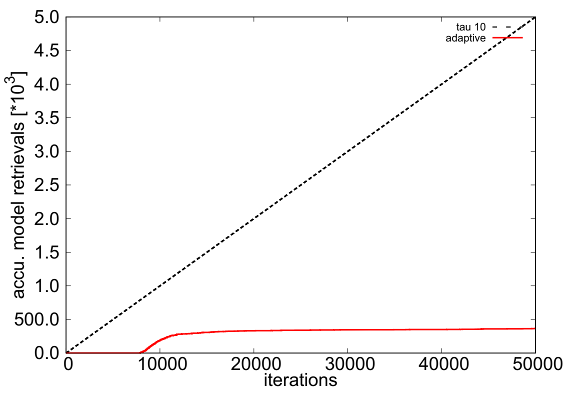

Fig. 7: AlexNet accumulated model retrievals

Table II summarizes the results for the three models with different training methods. Our method reduces the communication by 90%-95% than gossiping SGD, while keeping the accuracy within 5% loss. We reduce the training time by 52%, 20%, and 17% for AlexNet, GoogleNet and ConvNet respectively. This result shows that large networks such as AlexNet benefit more from our method. This is because large network requires more bandwidth to exchange the models and our method can reduce the communication traffic significantly and improve the performance.

TABLE II: Summary of training results

| model/method | AlexNet | | GoogleNet | | ConvNet | |
|---|---|---|---|---|---|---|
| | goss. | adap. | goss. | adap. | goss. | adap. |
| training time(min) | 1440 | 700 | 450 | 360 | 30 | 25 |
| model retrievals | 5000 | 365 | 5000 | 532 | 5000 | 250 |
| accuracy | 0.402 | 0.399 | 0.367 | 0.342 | 0.782 | 0.781 |

## VII. Conclusion

We propose an adaptive communication method to improve the performance of gossiping SGD. Instead of using a fixed period, our method uses the difference of model delta to determine when to exchange models with other machines. By doing so, we prevent exchanging similar models and thus reduce the communication traffic. This results in a more efficient model convergence process.

We evaluate the proposed method by training three popular deep neural networks – AlexNet, GoogleNet and ConvNet.

The experimental results demonstrate that our method achieves the same prediction accuracy while reducing the communication traffic by 92%. This reduces 52% training time for AlexNet than gossiping SGD. The experiments also show that large networks such as AlexNet benefit more from our method. This is because in distributed learning, the exchange of large networks usually becomes the performance bottleneck. Our method can reduce the communication significantly while preserving the accuracy, so as to improve the performance significantly.

## References

[1] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of go without human knowledge," vol. 550, pp. 354–359, 10 2017.

[2] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Computer Vision and Pattern Recognition (CVPR)*, 2015. [Online]. Available: http://arxiv.org/abs/1409.4842

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

[4] Z. Chen and X. Huang, "End-to-end learning for lane keeping of self-driving cars," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, June 2017, pp. 1856–1860.

[5] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to end learning for self-driving cars," *CoRR*, vol. abs/1604.07316, 2016. [Online]. Available: http://arxiv.org/abs/1604.07316

[6] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in *Proceedings of the 25th International Conference on Machine Learning*, ser. ICML '08, 2008, pp. 160–167.

[7] K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," 2014. [Online]. Available: http://arxiv.org/abs/1406.1078

[8] A. van den Oord, S. Dieleman, and B. Schrauwen, "Deep content-based music recommendation," in *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds., 2013, pp. 2643–2651.

[9] A. M. Elkahky, Y. Song, and X. He, "A multi-view deep learning approach for cross domain user modeling in recommendation systems," in *Proceedings of the 24th International Conference on World Wide Web*, ser. WWW '15, 2015, pp. 278–288.

[10] J. Dean, G. S. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, and A. Y. Ng, "Large scale distributed deep networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'12, 2012, pp. 1223–1231.

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

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

[13] P. H. Jin, Q. Yuan, F. N. Iandola, and K. Keutzer, "How to scale distributed deep learning?" 2016. [Online]. Available: http://arxiv.org/abs/1611.04581

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., 2012, pp. 1097–1105.

[15] 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.

[16] J. Chen, R. Monga, S. Bengio, and R. Jozefowicz, "Revisiting distributed synchronous sgd," in *International Conference on Learning Representations Workshop Track*, 2016. [Online]. Available: https://arxiv.org/abs/1604.00981

[17] W. Zhang, S. Gupta, X. Lian, and J. Liu, "Staleness-aware async-sgd for distributed deep learning," in *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, ser. IJCAI'16, 2016, pp. 2350–2356.

[18] J. Dean and L. A. Barroso, "The tail at scale," *Communications of the ACM*, vol. 56, pp. 74–80, 2013. [Online]. Available: http://cacm.acm.org/magazines/2013/2/160173-the-tail-at-scale/fulltext

[19] D. Das, S. Avancha, D. Mudigere, K. Vaidyanathan, S. Sridharan, D. D. Kalamkar, B. Kaul, and P. Dubey, "Distributed deep learning using synchronous stochastic gradient descent," 2016. [Online]. Available: http://arxiv.org/abs/1602.06709

[20] X. Lian, C. Zhang, H. Zhang, C.-J. Hsieh, W. Zhang, and J. Liu, "Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent," 05 2017.

[21] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *J. Mach. Learn. Res.*, vol. 12, pp. 2121–2159, Jul. 2011.

[22] "Convnet," https://code.google.com/archive/p/cuda-convnet/, (Accessed: 2017-11-26).

[23] A. Krizhevsky, "Learning multiple layers of features from tiny images," https://www.cs.toronto.edu/ kriz/learning-features-2009-TR.pdf, (Accessed: 2017-11-26).

[24] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.