ESync: An Efficient Synchronous Parallel Algorithm for Distributed ML in Heterogeneous Clusters

Zonghang LI
University of Electronic Science and Technology of China
zhli0117@163.com

Huaman ZHOU
University of Electronic Science and Technology of China
zhouhuaman@163.com

Hongfang YU
University of Electronic Science and Technology of China
yuhf@uestc.edu.cn

Zenglin XU
University of Electronic Science and Technology of China
zenglin@gmail.com

Abstract

In recent years, researchers have paid more attention to distributed machine learning to train large-scale models with mass data in parallel, which is much more efficient than training on a single machine. Classical synchronous / asynchronous parallel algorithms (such as SSGD and ASGD) achieve excellent performance in aggregating model updates from devices in large-scale isomorphic clusters, however, these algorithms are not suitable in heterogeneous clusters, which consist of devices with different computing capabilities. This paper proposes *ESync*, an efficient synchronous parallel algorithm which allows devices in heterogeneous clusters to take full use of their computing capabilities while waiting for the straggling devices, meanwhile achieves the accuracy of SSGD and the training speed of ASGD with lowest traffic load. We evaluate *ESync* on a heterogeneous cluster with 2 Intel E5-2650v4 CPUs and 4 GTX 1080TI GPUs, and find that *ESync* achieves 27$\times$ training speedup to SSGD without loss of accuracy on AlexNet and achieves the state-of-the-art performance in both i.i.d. and non-i.i.d. data distributions.

1 Introduction

Massive data and large-scale machine learning models contributed a lot for the rapid development of Artificial Intelligence. In recent years, researchers have paid more attention to distributed machine learning to train large-scale models with mass data in parallel, which is much more efficient than training on a single machine. For example, Jia et al.[14] trained ResNet-50 on ImageNet dataset with 2048 Tesla P40 GPUs and achieved 75.8% top-1 test accuracy in only 6.6 minutes. However, with the rapid development of GPU, TPU, FPGA, ASIC and other powerful computational resources, replacing all the old computing devices has become impractical and costly for enterprises and research institutes. Therefore, clusters are usually equipped with computing devices of different types and manufacturers, which leads to difference in computing capabilities among devices. Besides, Edge AI supports cooperative training of millions of edge devices, such as mobile phones, smart watches, cameras and sensors, and the diversity of edge devices makes the heterogeneity of clusters much more significant.

The training process of a standard distributed machine learning algorithm in synchronous mode can be divided into three steps: (1) Each device calculates model updates with local data independently; (2) Aggregate all the model updates and broadcast to all devices; (3) Update model parameters with aggregated model updates. Synchronous SGD (SSGD)[3] requires all the devices to aggregate

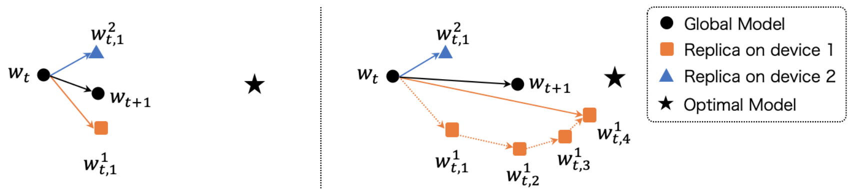

Figure 1: An overview of SSGD (left) and *ESync* (right) within a communication round. Suppose we are training on a weak device (blue triangle) and a powerful device (orange square). When the weak device completes an iteration, the powerful device has completed four iterations in *ESync* while only one iteration in SSGD. The powerful device provides model updates with a long-term vision, which makes the global model w_{t+1} closer to the optimal model.

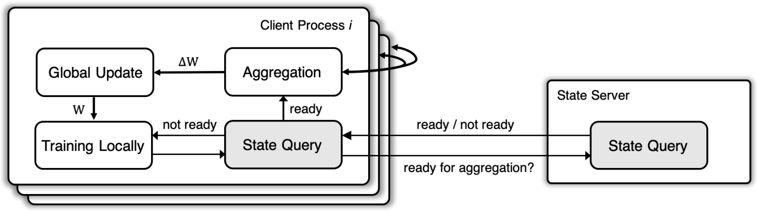

Figure 2: An overview of *ESync* architecture. In each iteration, devices always query the State Server to decide whether to perform aggregation or continue training locally, and they start to aggregate only when *READY* ACK is received explicitly.

model updates at the same pace, in other words, devices with strong computing capabilities are blocked to wait for the straggling devices. The heterogeneity of clusters severely limits the training efficiency of SSGD to the weakest device, which causes the "Short Board" problem and results in inefficient training, low data throughput and great waste of computational resources. Therefore, SSGD is preferred in isomorphic, stable and reliable clusters, in which devices have similar computing capabilities. A natural idea to balance the computing time of each device in heterogeneous clusters is processing more samples while waiting for other devices, i.e. set larger batch size for devices with stronger computing capabilities. However, large batch size causes loss of accuracy[15]. In our preliminary experiments, we find that simply increase batch size according to the computing capability can not accelerate convergence, and probably lead to a decline in accuracy.

Asynchronous parallel algorithms (e.g. ASGD[5]) show strong adaptability to the heterogeneity of clusters (the performance reduction or failure of one device will not block other devices), but they also introduce "Delayed Gradient" problem, which uses outdated model updates to update the latest model, resulting in a decline in accuracy and an increase in the number of training iterations. Some researchers have analyzed the negative effect of delayed gradient to the convergence[19][2][1] and tried to improve ASGD from different perspectives to suppress the effect of delayed gradient on model accuracy, such as penalizing delayed gradient by well-designed learning rate[21][24], compensate delayed gradient[29], and control the difference of iterations[12]. However, these approaches will be difficult to meet the training and convergence requirements in clusters with strong heterogeneity, and ultimately lead to inefficient training and loss of accuracy.

To solve the "Short Board" problem and improve the training efficiency without loss of accuracy in heterogeneous clusters, we tackle the challenge of "Short Board" problem based on synchronous parallel algorithms directly to avoid the introduction of delayed gradient, instead of continuing to alleviate the negative effect caused by delayed gradient. This paper introduces *ESync*, an efficient synchronous parallel algorithm which takes full use of the computing capabilities of devices with lowest traffic load in heterogeneous clusters, meanwhile accelerates the training speed greatly without loss of accuracy. The key idea of *ESync* is to produce higher quality model updates that have a long-term vision while waiting for other devices. As shown in Figure 1 and Figure 2, at the beginning of each communication round, all the devices hold the same global model, save an additional replica of the global model, and train the replica for n_i iterations (on device i) with local data via SGD (or other optimization algorithms), where n_i is given implicitly by the state query operation. When all devices are ready, they calculate model updates based on difference between the global model and their latest replicas, and send model updates to other devices to perform aggregation. Finally, the aggregated model updates are averaged and used to update the global model. The challenge of *ESync* is to assign the number of local iterations n_i to device i automatically according to the changing computing capabilities. To this end, *ESync* introduces State Server to coordinate the devices by the state query operation to ensure that when the slowest device completes a local iteration, other devices have completed local iterations as many times as possible. Specially, *ESync* allows aggregation operations to be performed in a synchronous manner in heterogeneous clusters, and supports many efficient collective communication algorithms (e.g. Ring Allreduce[8], Butterfly[22]).

We evaluate *ESync* on a heterogeneous cluster with 2 Intel E5-2650v4 CPUs and 4 GTX 1080TI GPUs. Results show that *ESync* realizes a good balance of computing time among devices, achieves $7\times$ training speedup to SSGD on ResNet (up to $27\times$ on AlexNet) without loss of accuracy, processes $33\times$ more samples per second than SSGD, transmits $4\times$ less data per second than ASGD, and achieves the state-of-the-art performance in both i.i.d. and non-i.i.d. data distributions.

2 Related Work

Short Board. In Synchronous SGD (SSGD)[3], devices calculate model updates independently with local data, and push these model updates to other devices to perform aggregation. By using a barrier, devices with strong computing capabilities need to wait until other devices reach the barrier, which makes computational resources idle for a long time and causes the "Short Board" problem. To improve the training efficiency, Dean et al.[5] propose Asynchronous SGD (ASGD), which allows devices to perform computation and aggregation in different pace, with no need to wait for model updates from other devices. ASGD solves the "Short Board" problem and accelerates the training speed significantly. However, ASGD also introduces delayed gradient, which reduces the accuracy and convergence of the model, and the effect will be more serious when the difference in computing capabilities among devices is large in non-i.i.d. data distribution.

Delayed Gradient. ASGD updates model $w_{t+\tau}$ with outdated model updates $g(w_t)$, resulting in mismatches between the model and model updates, and causes undesirable damage to the accuracy and convergence. Ho et al.[12] propose Stale Synchronous Parallel (SSP) to make a tradeoff between the accuracy and training speed by allowing the fastest device to be ahead of the slowest device by up to a bounded number of iterations τ. However, SSP still apply the outdated model updates to update the latest model and staleness still exist during the training process, eventually, SSP will degenerate into Bulk Synchronous Parallel (BSP)[7] with staleness in heterogeneous clusters when all other devices reach the threshold $t+\tau$ and blocked to wait for the slowest device, which makes the problems of "Short Board" and "Delayed Gradient" worse. McMahan et al.[21] proposes AdaptiveRevision and Sra et al.[24] propose Adadelay to penalize the delayed gradient by well-designed learning rate, but these approaches suppress the contributions of data on the straggling devices, which leads to unexpected loss of accuracy when the difference of steps is large. Zheng et al.[29] propose DC-ASGD, which leverages the first-order term of Taylor expansion of the gradient function and approximates the Hessian matrix of the loss function to compensate delayed gradient, but its convergence requirements will be difficult to satisfy in clusters with strong heterogeneity, for the value of the high-order term $O((w_{t+\tau} - w_t)^2)I$ may be large and can not be ignored.

Although the derivative ASGD algorithms (e.g. DC-ASGD) alleviate the decline of accuracy and convergence caused by delayed gradient, they still require a long time to converge to the accuracy of sequential SGD in heterogeneous clusters. Instead, *ESync* tackles the challenge of "Short Board" problem from an innovative perspective, i.e. optimizes based on synchronous parallel algorithms directly to avoid the introduction of delayed gradient, and utilizes the blocking time of aggregation operations to generate more valuable model updates.

3 The *ESync* Algorithm

Consider training a classification task on a heterogeneous cluster with p devices, and dataset are distributed on these devices $D = \{D_1, D_2, ..., D_p\}$. For each dataset D_i on device i, we define the local loss l_i as:

$$l_i = \frac{1}{|D_i|} \sum_{(x_j, y_j) \in D_i} L(F(x_j, w), y_j) \tag{1}$$

where $|\cdot|$ denotes the size of the dataset, (x_j, y_j) denotes the data and label of the j-th sample in dataset D_i, $(F(\cdot), w)$ denotes the forward function and parameters of the model, and $L(\cdot)$ denotes the loss function. The global loss function on the distributed dataset can be defined as:

$$l = \frac{\sum_{i=1}^{p} |D_i| \cdot l_i}{|D|} \tag{2}$$

Our goal is to minimize the global loss l, i.e. to find the optimal shared parameters w^*:

$$w^* = \arg\min_{w} l_i \qquad i = 1, 2, ..., p \tag{3}$$

Gradient-descent techniques such as SGD is often used to solve such optimization problems. In this paper, we take mini-batch SGD as an example for simplicity, and it can be easily replaced by other optimization algorithms.

3.1 Algorithm Design of *ESync*

Distributed machine learning tasks can be considered as a treasure hunt game with multiple explorers. In standard synchronous parallel algorithms (e.g. SSGD), each explorer shares one discovery in every step, and the explorers are forced to wait until all of them have a discovery. Instead, *ESync* allows experienced explorers to explore as many steps as possible, and share their discoveries when the freshmen are ready. The training process of *ESync* contains three steps: state query, training locally, aggregation and global update, as shown in and Figure 2.

We define the global model at the t-th communication round as w_t, and the replica of w_t at device i after k local iterations as $w_{t,k}^i$. w_0 is random initialized and shared by all devices. For $t > 0$, each device keeps the same global model w_t at the beginning, and saves a replica $w_{t,0}^i$ at device i:

$$w_{t,0}^i \leftarrow w_t \tag{4}$$

Step1: State Query. We introduce State Server to support state query, which decides whether devices should perform aggregation or continue training locally at the beginning of each local iteration. The State Server assigns the number of local iterations n_i for device i implicitly, where n_i is the number of times signal *NOT-READY* is received during a communication round. The design of State Server is described in Section 4.

Step2: Training Locally. For each local iteration $k (1 \leq k < n_i)$, we sample a small batch data $D_e = \{(x_{i_1}, y_{i_1}), (x_{i_2}, y_{i_2}), ..., (x_{i_b}, y_{i_b})\}$ from dataset D_i in random, calculate gradients and apply SGD to update the model replica $w_{t,k}^i$:

$$w_{t,k+1}^i \leftarrow w_{t,k}^i - \eta \nabla_w \frac{1}{b} \sum_{(x_j, y_j) \in D_e} L(F(x_j, w), y_j) \tag{5}$$

where η denotes the local learning rate, and b denotes the batch size. We can easily replace SGD with other optimization algorithms (e.g. Momentum[25], RMSProp[27], Adam[16]) by modifying Formula (5).

Step3: Aggregation and Global Update. When device i completes local iterations for n_i times, calculates the model updates based on difference between the latest replica w_{t,n_i}^i and the global model w_t:

$$\Delta w_t^i = w_{t,n_i}^i - w_t \tag{6}$$

then model updates Δw_t^i are sent to other devices to perform aggregation, and the aggregated model updates Δw_t are retrieved to each device to update the global model w_t:

$$\Delta w_t = \frac{1}{p} \sum_{i \in \{1,2,...,p\}} \Delta w_t^i \tag{7}$$

$$w_{t+1} = w_t + \epsilon \Delta w_t \tag{8}$$

where ϵ denotes the global learning rate and can be simply set to 1.

The pseudo-code of *ESync* is shown in Algorithm 1. We can implement the aggregation operation *GLOBAL-SUM-UPDATES* at line 19 based on Parameter Server[18] or MPI Allreduce[9], and apply compression technologies (e.g. 1-bit SGD[23] and DGC[20]) to reduce the amount of data that needs to be transmitted.

Algorithm 1 *ESync* $(D_i, T, p, \eta, b, r, \epsilon = 1)$

```
Input: Dataset $D_i = \{(x_1, y_1), (x_2, y_2), ..., (x_d, y_d)\}$; Maximum number of global iterations $T$;
    Number of devices $p$; Local learning rate $\eta$; Batch size $b$; Device id $r$; Global learning rate $\epsilon$
    (optional, the default value is 1).
 1: RANDOM-INITIALIZE-MODEL-PARAMETERS($w_0, r$);
 2: INITIALIZE-STATE-SERVER($p, \varepsilon$);
 3: $c \leftarrow$ RUN-BATCH-DATA-TEST($b$);
 4: $t_e \leftarrow$ GET-CURRENT-TIME();
 5: $t \leftarrow 0$; $k \leftarrow 0$;
 6: $w^r_{0,0} \leftarrow w_0$;
 7: while $t \leq T$:
 8:     $ready \leftarrow$ QUERY-STATE($r, k, c, t_e$);
 9:     if not $ready$:
10:         $t_s \leftarrow$ GET-CURRENT-TIME();
11:         $data \leftarrow$ GET-NEXT-RANDOM-MINIBATCH($b$);
12:         $gradients \leftarrow$ COMPUTE-AVERAGE-GRADIENTS($w^r_{t,k}, data$);
13:         $w^r_{t,k+1} \leftarrow w^r_{t,k} - \eta \cdot gradients$;
14:         $t_e \leftarrow$ GET-CURRENT-TIME();
15:         $c \leftarrow t_e - t_s$;
16:         $k \leftarrow k + 1$;
17:     else:
18:         $\Delta w^r_t \leftarrow w^r_{t,k} - w_t$;
19:         $aggregated\_updates \leftarrow$ GLOBAL-SUM-UPDATES($r, \Delta w^r_t$);
20:         $w_{t+1} \leftarrow w_t + \epsilon \cdot aggregated\_updates/p$;
21:         $t \leftarrow t + 1$;
22:         $w^r_{t,0} \leftarrow w_t$;
23:         $k \leftarrow 0$;
24:         $t_e \leftarrow$ GET-CURRENT-TIME();
25:         RESET-STATE-SERVER($r, t, t_e$);
Output: Trained model parameters $w_t$.
```

3.2 Convergence Analysis

In this section, we analyze the convergence of *ESync* and compare with the standard synchronous mini-batch SGD[6]. Assume the number of local iterations of p devices are $\{n_1, n_2, ..., n_p\}(n_i \geq 1, n_i \in \mathbb{Z})$. To simplify the analysis, we consider *ESync* as a synchronized algorithm that performs $\bar{n} = \frac{1}{p}\sum_{i=1}^{p} n_i$ local iterations on each device and $b\bar{n}p$ divides m without loss of generality.

Theorem 1. *Let $L(w, z)$ be a loss function with L-smooth convexity, and assume that the variance of the stochastic gradient $\nabla_w L(w, z)$ is bounded by σ^2 for all $w \in W$ and $z \in Z$. If the update rule has the serial expected optimality gap bound $\tilde{\psi}(\sigma^2, m) = \frac{1}{m}\psi(\sigma^2, m)$ when processed m samples, the expected optimality gap bound of Algorithm 1 is*

$$\tilde{\psi}\left(\frac{\sigma^2 \bar{n}}{bp}, \frac{m}{b\bar{n}p}\right) \tag{9}$$

specially, given $\psi(\sigma^2, m) = 2D^2L + 2D\sigma\sqrt{m}$, and Formula (9) is equal to

$$\frac{2b\bar{n}pD^2L}{m} + \frac{2D\sigma\bar{n}}{\sqrt{m}} \tag{10}$$

Proof of Theorem 1 is given in Appendix A. We can see that the bound of *ESync* is asymptotically equivalent to the bound of standard synchronous mini-batch SGD $O(\frac{1}{m} + \frac{1}{\sqrt{m}})$, which ensures that *ESync* can converge to the same accuracy as SSGD. Besides, *ESync* alleviates the "Large Batch Training" problem (larger batch size often leads to lower test accuracy)[15] by expanding the variance of the stochastic gradients by n_i times on each device.

4 State Server

We introduce State Server, which can be deployed on any devices, to assign the number of local iterations n_i for each device automatically and dynamically. The main idea is that when the slowest device completes a local iteration, other devices have completed local iterations as many times as possible. The State Server is the key for *ESync* to solve the "Short Board" problem, and a good design for State Server can significantly reduce the blocking time of the aggregation operations.

4.1 State Query

Considering that computing capabilities among devices may change, the State Server does not inform device i of its number of local iterations n_i directly. Instead, devices query the State Server whether they should perform aggregation or continue training locally in each iteration. This approach provides fine-grained control over devices in dynamic clusters compared with informing devices of n_i directly.

A simple but inefficient way is to perform aggregation when all devices complete at least one local iteration. Suppose there are two straggling devices A and B with the same computing capabilities. Although A and B complete computations at the same time, there will be a device that first reports to the State Server, and the device reports first (A) will continue training locally but the latter (B) starts to perform aggregation and then be blocked. Therefore, the "Short Board" problem is not solved because other devices still need to wait until A finishes its new iteration.

State Server adopts a more efficient way, i.e. performs aggregation when the querying device can not finish new iteration before the slowest device is ready. The pseudo-code of *QUERY-STATE* on State Server is shown in Algorithm 2. Assume the computing capabilities (time to process b samples) of p devices are $\{c_1, c_2, ..., c_p\} (c_1 \geq c_2 \geq ... \geq c_p > 0)$, and ε is a small constant to avoid boundary error. We guarantee that the blocking time of aggregation operation is less than c_p (0.03 seconds in our experiments), which is negligible compared to the slowest device (3.5 seconds in our experiments).

Algorithm 2 *State-Server*

```
1: procedure QUERY-STATE($r, k, c, t_e$):
2:     // Record Format: $(rank, local\_iterations, capability, last\_finish\_time, ready, round)$.
3:     Update record $current$ (rank $r$) to $(r, k, c, t_e, -1, -1)$, where $-1$ means unchanged;
4:     $slowest \leftarrow$ FIND-SLOWEST-DEVICE();
5:     if $k$ == 0 or $slowest$ has not complete the $PULL$ operation yet:
6:         return $false$;
7:     $current\_time \leftarrow$ GET-CURRENT-TIME();
8:     $rest\_time \leftarrow slowest$.get_capability() $- current\_time + slowest$.get_last_finish_time();
9:     if $current$ == $slowest$ or $slowest$ is ready or $current$.get_capability() $+\varepsilon > rest\_time$:
10:        $current$.set_ready($true$);
11:        return $true$;
12:    return $false$;
```

4.2 Non-Blocking Mode with Delayed ACK

Considering that the clusters may be in a network with high communication delay, the state query operation will block the execution of training locally and aggregation for a long time, and forces the devices to wait until the State Server responds a signal *READY* or *NOT-READY*, which causes great waste of computational resources. Besides, the available network resources may be heterogeneous and dynamic, and the difference of communication delay introduces noise to the State Server.

We redefine capability c_i as the time to process b samples and the RTT of state query operation on device i to address both the difference of computing and communication capabilities, and allow devices to continue training locally while waiting for the response from the State Server to avoid waste of computational resources. We assume that implicit *NOT-READY* ACKs are received by default after sending state query requests, and devices continue training locally until an explicit *READY* ACK is received. When *READY* ACK of the k-th iteration is received, each device aborts the current $(k + \delta)$-th iteration immediately and starts to perform aggregation based on iterations 1 to $k + \delta - 1$, where $\delta - 1$ is the number of local iterations that can be completed in RTT time.

5 Experiments

The experimental platform is built on MXNet[4], a flexible and efficient library for deep learning, and supports distributed training by Parameter Server. We implement the aggregation operation on Parameter Server and an open source communication library PS-LITE. In this implementation, devices push model updates to the centralized KVStore to perform aggregation, and then pull aggregated model updates to local memory to update the model. DC-ASGD is selected for comparison because it proved to outperform both SSGD and ASGD and nearly approaches the performance of sequential SGD. We evaluate *ESync* on a heterogeneous cluster with 2 Intel E5-2650v4 CPUs and 4 GTX 1080TI GPUs, and compare its speedup, test accuracy, data throughput, traffic load, computing time ratio with SSGD, ASGD and DC-ASGD.

Note that SSGD is only suitable for moderate scale optimization problems in heterogeneous clusters due to the training efficiency of SSGD is severely limited by CPUs, which prohibits the system evaluation of larger scale datasets, such as ImageNet. Therefore, for fair comparison, we selected the classic moderate scale Fashion-MNSIT[28] dataset, which consists of a training set of 60,000 samples and a test set of 10,000 samples. We distributed the Fashion-MNIST dataset to each device in i.i.d. (uniform random sampling) and non-i.i.d. (assign samples of several specific classes to one device) settings. Without loss of generality, we set the learning rate to 0.0005 and batch size to 64 in all experiments, and trained ResNet[10] models on different settings respectively. Furthermore, we define the speedup as the time ratio to achieve test accuracy 0.8, that is close to the convergent accuracy, and the computing time ratio as proportion of total computing time to total training time.

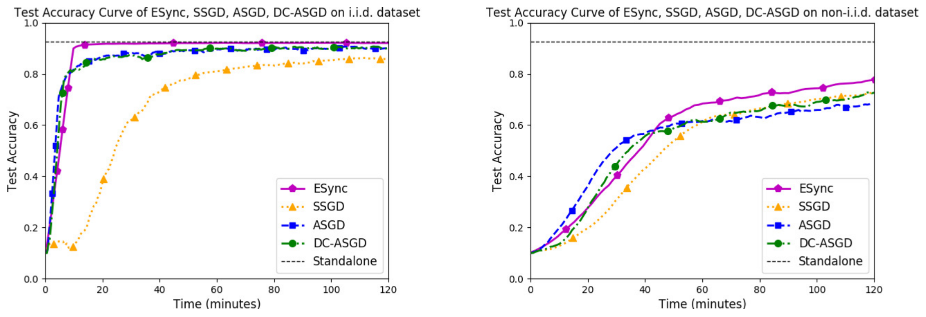

Figure 3: Test accuracy curve on i.i.d and non-i.i.d. Fashion-MNIST dataset.

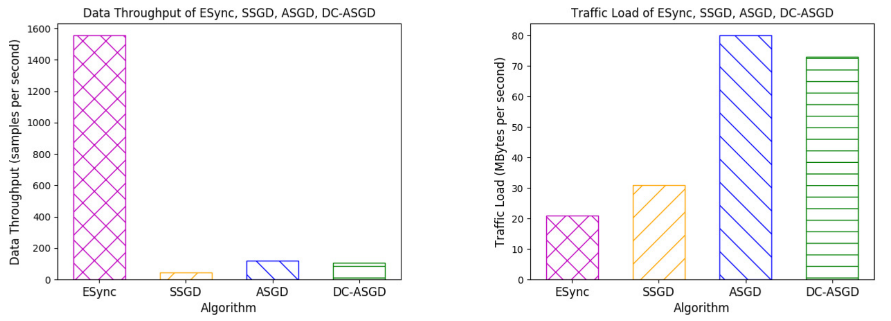

Figure 4: Data throughput and traffic load per second.

Test accuracy and training speed. *ESync* increases training speed greatly without loss of accuracy and achieves the state-of-the-art performance in both settings. As shown in Figure 3, while training on i.i.d. dataset (left), *ESync* achieves similar training speed as ASGD and DC-ASGD and 7× speedup to SSGD, meanwhile achieves the test accuracy of sequential SGD in the shortest time. Besides, while training on non-i.i.d. dataset (right), *ESync* trains faster than SSGD without loss of accuracy while the accuracy of ASGD is impaired, and DC-ASGD barely achieves the accuracy of SSGD.

Data throughput and traffic load. *ESync* enables heterogeneous clusters to process data more efficiently, meanwhile minimizes traffic load in network. As shown in Figure 4, *ESync* trains 33× more samples than SSGD and 13× more samples than ASGD per second, however, only 1/4 of the

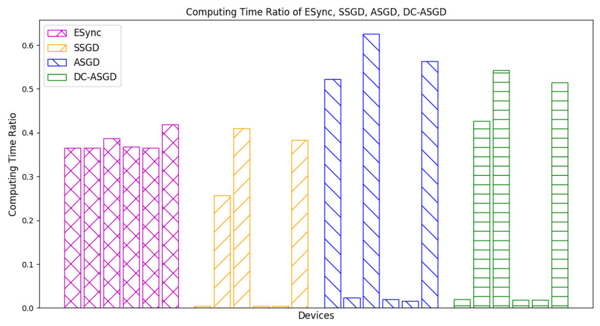

Figure 5: Computing time ratio on each device.

data need to be transmitted per second compared to ASGD. The reason is that *ESync* performs multiple iterations locally, which increases data throughput and reduces the number of communication.

Computing time. *ESync* improves the utilization rate of computing power on devices with strong computing capabilities significantly. As shown in Figure 5, SSGD takes too much time waiting for straggling devices, ASGD takes little time on computing but always in communication, and DC-ASGD takes much time in compensating delayed gradient on the parameter servers (which results in throughput, traffic load and computing time reduction compared with ASGD). Instead, *ESync* realizes a good balance of computing time among devices, and enables devices with strong computing capabilities to make full use of their computational resources while waiting for the straggling devices.

We compare the speedup of *ESync* to SSGD and the test accuracy with sequential SGD on several classic models[11][17][26][13]. Each group of experiments was repeated for five times, and the $mean$ and $stddev$ of the test accuracy are given in the form of $mean \pm stddev$, as shown in Table 1. Results show that *ESync* accelerates the training process greatly and achieves almost the same (sometimes higher) accuracy as sequential SGD.

Table 1: Speedup and test accuracy of *ESync* on different models.

Model	Speedup	Test Accuracy ($mean \pm stddev$)	
		ESync	Sequential SGD
AlexNet	27×	**0.932** ± **0.003**	0.928 ± 0.006
Inception-v3	7×	**0.940** ± **0.003**	0.932 ± 0.004
ResNet-18-v1	7×	**0.926** ± **0.002**	0.919 ± 0.004
ResNet-50-v1	4×	**0.914** ± **0.001**	0.910 ± 0.003
ResNet-50-v2	6×	0.917 ± 0.001	**0.918** ± **0.002**
MobileNet-v1	2×	0.905 ± 0.002	**0.906** ± **0.003**

6 Conclusion

This paper proposes *ESync*, an efficient synchronous parallel algorithm which allows devices in heterogeneous clusters to take full use of their computing capabilities while waiting for the straggling devices, meanwhile achieves the accuracy of SSGD and the training speed of ASGD with lowest traffic load. We introduce State Server to assign the number of local iterations for each device automatically and dynamically, aim to solve the "Short Board" problem in heterogeneous and dynamic clusters. Specially, *ESync* allows aggregation operations to be performed in a synchronous manner in heterogeneous clusters, and provides users with flexibility in selecting different collective algorithms and communication topologies according to the characteristics of tasks and network.

In future work, we will expand *ESync* to large-scale clusters for more experimental results. Further more, we are going to design a mechanism and algorithms based on *ESync* that supports automatic communication mode design based on the characteristics of clusters (e.g. number of devices, computing resources, communication resources), and realize seamless switching of communication modes while detecting dramatic changes in the clusters to better adapt to heterogeneous and dynamic clusters, such as multi-tenant clusters and edge devices in Edge AI.

References

[1] Alekh Agarwal and John C Duchi. Distributed delayed stochastic optimization. In *Advances in Neural Information Processing Systems*, pages 873–881, 2011.

[2] Haim Avron, Alex Druinsky, and Anshul Gupta. Revisiting asynchronous linear solvers: Provable convergence rate through randomization. *Journal of the ACM (JACM)*, 62(6):51, 2015.

[3] Jianmin Chen, Xinghao Pan, Rajat Monga, Samy Bengio, and Rafal Jozefowicz. Revisiting distributed synchronous sgd. *arXiv preprint arXiv:1604.00981*, 2016.

[4] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274*, 2015.

[5] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. Large scale distributed deep networks. In *Advances in neural information processing systems*, pages 1223–1231, 2012.

[6] Ofer Dekel, Ran Gilad-Bachrach, Ohad Shamir, and Lin Xiao. Optimal distributed online prediction using mini-batches. *Journal of Machine Learning Research*, 13(Jan):165–202, 2012.

[7] Alexandros V Gerbessiotis and Leslie G Valiant. Direct bulk-synchronous parallel algorithms. *Journal of parallel and distributed computing*, 22(2):251–267, 1994.

[8] Andrew Gibiansky. Bringing hpc techniques to deep learning. `http://research.baidu.com/bringing-hpc-techniques-deep-learning/`, 2017.

[9] William D Gropp, William Gropp, Ewing Lusk, and Anthony Skjellum. *Using MPI: portable parallel programming with the message-passing interface*, volume 1. MIT press, 1999.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016.

[12] Q Ho, J Cipar, H. Cui, J. K. Kim, S Lee, P. B. Gibbons, G. A. Gibson, G. R. Ganger, and E. P. Xing. More effective distributed ml via a stale synchronous parallel parameter server. *Advances in Neural Information Processing Systems*, 2013(2013):1223, 2013.

[13] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[14] Xianyan Jia, Shutao Song, Wei He, Yangzihao Wang, Haidong Rong, Feihu Zhou, Liqiang Xie, Zhenyu Guo, Yuanzhou Yang, Liwei Yu, et al. Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes. *arXiv preprint arXiv:1807.11205*, 2018.

[15] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

[16] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *Computer Science*, 2014.

[17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[18] Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. Scaling distributed machine learning with the parameter server. In *11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14)*, pages 583–598, 2014.

[19] Xiangru Lian, Yijun Huang, Yuncheng Li, and Ji Liu. Asynchronous parallel stochastic gradient for nonconvex optimization. In *Advances in Neural Information Processing Systems*, pages 2737–2745, 2015.

[20] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William J Dally. Deep gradient compression: Reducing the communication bandwidth for distributed training. *arXiv preprint arXiv:1712.01887*, 2017.

[21] Brendan McMahan and Matthew Streeter. Delay-tolerant algorithms for asynchronous distributed online learning. In *Advances in Neural Information Processing Systems*, pages 2915–2923, 2014.

[22] Rolf Rabenseifner. Optimization of collective reduction operations. In *International Conference on Computational Science*, pages 1–9. Springer, 2004.

[23] Frank Seide, Hao Fu, Jasha Droppo, Gang Li, and Dong Yu. 1-bit stochastic gradient descent and its application to data-parallel distributed training of speech dnns. In *Fifteenth Annual Conference of the International Speech Communication Association*, 2014.

[24] Suvrit Sra, Adams Wei Yu, Mu Li, and Alexander J. Smola. Adadelay: Delay adaptive distributed stochastic convex optimization. *Mathematics*, 2016.

[25] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147, 2013.

[26] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[27] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

[28] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[29] Shuxin Zheng, Qi Meng, Taifeng Wang, Wei Chen, Nenghai Yu, Zhi-Ming Ma, and Tie-Yan Liu. Asynchronous stochastic gradient descent with delay compensation. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 4120–4129. JMLR.org, 2017.