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

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 and synchronizing model updates from devices in large-scale isomorphic clusters, however, these algorithms are not suitable in heterogeneous clusters, which consist of computing devices with different computing capabilities. This paper proposes *ESync*, an efficient synchronous parallel algorithm which takes both the accuracy of SSGD and training speed of ASGD, meanwhile takes full advantage of the computing capabilities of the heterogeneous clusters with lowest traffic load. We evaluate *ESync* on a heterogeneous cluster with several Intel E5-2650v4 CPUs and GTX 1080TI GPUs, and show that *ESync* 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, Xianyan, et al.[1] trained ResNet-50 on ImageNet dataset with 2048 Tesla P40 GPUs and achieved 75.8% top-1 test accuracy in only 6.6 minutes.

The training process of distributed machine learning can be divided into three steps: (1) Each computing device calculates model updates with local data independently; (2) Aggregate all the model updates and broadcast to all computing devices; (3) Update model parameters with aggregated model updates. Synchronous SGD (SSGD)[2] requires all the devices to aggregate their model updates at the same pace, and devices with strong computing capabilities are blocked to wait for other straggling devices. Therefore, SSGD is suitable for isomorphic, stable and reliable clusters, in which computing devices have similar computing capabilities.

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. Clusters are usually equipped with computing devices of different types and manufacturers, which will lead to differences in computing capabilities among devices. Besides,

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

Federated Learning[3] 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 heterogeneity of clusters severely limits the training efficiency of SSGD to the weakest device, which causes the "Short Board" problem and great waste of computational resources. A natural idea to take full advantage of the computing capabilities of the 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[4]. Our experiments find that simply increase batch size according to 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", which uses outdated model updates to update the latest model parameters, 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" problem to the convergence[6][7][8] and tried to improve ASGD from different perspectives to suppress the effect of delayed gradient on model accuracy, such as penalizing delay gradient by well-designed learning rate[9][10], compensate delay gradient[11], and control the difference of iterations[12]. However, these approaches have not solved "Delayed Gradient" problem fundamentally, and they may be difficult to meet the training or convergence requirements in clusters with strong heterogeneity.

This paper introduces *ESync*, an efficient synchronous parallel algorithm designed for distributed machine learning tasks in heterogeneous clusters. The key idea of *ESync* is to produce higher quality model updates that have a long-term vision while waiting for other devices. In each communication round, each device holds the same global model parameters, saves an additional replica of the global model parameters, and trains the local model replica for n iterations via SGD (or other optimization algorithms) using local data independently, where the number of local iterations n is different across devices and dynamically assigned by State Server according to the computing capability. *ESync* introduces State Server to ensure that, when the slowest device completes computations, other devices have completed local iterations as many times as possible. When all devices are ready, each device calculates model updates based on difference between local model replica and global model, and sends updates to other devices to perform aggregation. Finally, the aggregated model updates are averaged and added to the global model parameters.

We evaluate *ESync* on a heterogeneous cluster with 2 Intel E5-2650v4 CPUs and 4 GTX 1080TI GPUs. Results show that *ESync* achieves $7\times$ the acceleration of SSGD without loss of accuracy, processes $33\times$ more samples per second than SSGD, transmits $15\times$ less data per second than ASGD, and achieves the state-of-the-art performance in both i.i.d. and non-i.i.d. datasets.

Contributions. To the best of authors' knowledge, we propose the first efficient synchronous parallel algorithm to solve the "Short Board" problem in heterogeneous ML clusters. To sum up, the main contributions of this paper are as follow: (1) We introduce *ESync*, an efficient synchronous parallel algorithm that solves both the "Short Board" and "Delayed Gradient" problems, and greatly increases training speed without loss of accuracy; (2) We introduce State Server to assign the number of local iterations for each device automatically and dynamically, which provides fine-grained control over devices in dynamic clusters; (3) We provide a solution to take full advantage of the computing capabilities of the heterogeneous clusters with lowest traffic load; (4) We provide a solution that allows aggregation operations to be performed in a synchronous manner in heterogeneous clusters, and supports many efficient collective communication algorithms (e.g. Ring Allreduce[13], Butterfly[14]).

The organization of this paper is as follows. In Section 2, we discuss the problems of "Short Board" and "Delayed Gradient", review the corresponding solutions, and reveal their limitations in heterogeneous clusters. In Section 3, we describe the *ESync* algorithm and analyze its convergence. In Section 4, we discuss the design of State Server and propose a design scheme for harsh communication environment. In Section 5, we evaluate *ESync* and compare its training speed, accuracy, data throughput, traffic load, communication ratio with SSGD and ASGD. Finally, we conclude this paper and delineate several promising directions for future research in Section 6.

2 Related Work

This paper addresses both the problems of "Short Board" and "Delayed Gradient", and strives to take full advantage of the computing capabilities of the heterogeneous clusters without loss of accuracy.

Short Board Problem. In Synchronous SGD (SSGD)[2], each device calculates model updates independently with local data, and pushes these 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, Jeffrey, 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 of devices is large in non-i.i.d. data distribution.

Delayed Gradient Problem. ASGD updates model $w_{t+\tau}$ with outdated model updates $g(w_t)$, resulting in a mismatch between the model updates and the model, and causes unexpected turbulence to the training trajectory. Ho, Qirong, et al.[12] propose Stale Synchronous Parallel (SSP) to make a tradeoff between accuracy and training speed by allowing fastest device to be ahead of the slowest device by up to a bounded number of iterations τ. However, staleness still exists during the training process, and SSP will degenerate into Bulk Synchronous Parallel (BSP)[15] with staleness in heterogeneous clusters, eventually make the "Short Board" and "Delayed Gradient" problems worse. McMahan[9] propose AdaptiveRevision and Sra, Suvrit, et al.[10] propose Adadelay to penalize delay gradient by well-designed learning rate, but these approaches reduce the contribution of the gradient and limit the training efficiency when the difference of steps τ is large ($\tau \approx 150$ in our experiments). Zheng, Shuxin, et al.[11] propose DC-ASGD, which leverages the first-order term of Taylor expansion of the gradient function and approximates Hessian matrix of the loss function to compensate delayed gradient, but its convergence requirements will be difficult to satisfy in clusters with strong heterogeneity (the value of the high-order term $O((w_{t+\tau} - w_t)^2)I$ may be large and cannot be ignored).

Instead of continuing to alleviate the negative effect of delayed gradient, *ESync* tackles the challenge of "Short Board" problem based on synchronous parallel algorithms directly, to avoid the introduction of delayed gradient and reduce the blocking time of the aggregation operations.

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 at 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 example for simplicity, and it can be easily replaced by other optimization algorithms.

3.1 Algorithm Modeling

Distributed machine learning tasks can be considered as a treasure hunt game with multiple explorers. In standard synchronous parallel algorithms such as SSGD, each explorer shares one discovery in

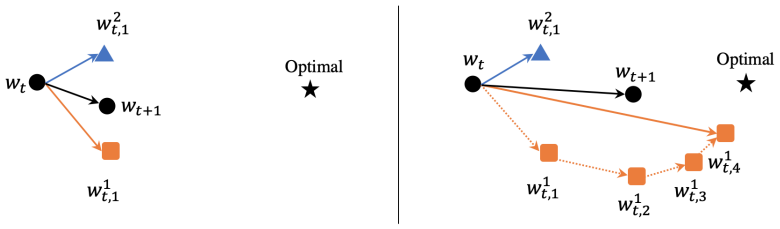

Figure 1: Overview of SSGD (left) and *ESync* (right) within a communication round.

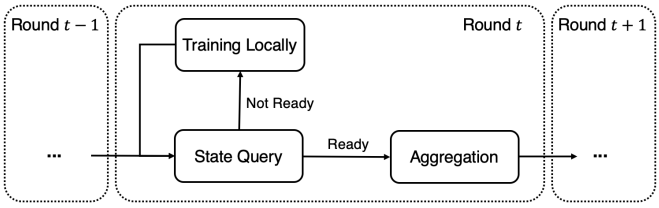

Figure 2: Training process of *ESync*.

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. An illustration of SSGD and *ESync* within a communication round is shown in Figure 1, and the training process of *ESync* is shown in Figure 2.

We define the global model parameters 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 parameters 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.

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, RMSProp, Adam) by modifying Formula (5).

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

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

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

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

Algorithm 1 *ESync* (D_i, T, p, η, 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 η; Batch size b; Device id r; Global learning rate ϵ (optional, the default value is 1).

1: *RANDOM-INITIALIZE-MODEL-PARAMETERS*(w_0, r);
2: *INITIALIZE-STATE-SERVER*(p, ε);
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: $\quad ready \leftarrow$ *QUERY-STATE*(r, k, c, t_e);
9: $\quad$ if not $ready$:
10: $\quad\quad t_s \leftarrow$ *GET-CURRENT-TIME()*;
11: $\quad\quad data \leftarrow$ *GET-NEXT-RANDOM-MINIBATCH*(b);
12: $\quad\quad gradients \leftarrow$ *COMPUTE-AVERAGE-GRADIENTS*($w^r_{t,k}, data$);
13: $\quad\quad w^r_{t,k+1} \leftarrow w^r_{t,k} - \eta \cdot gradients$;
14: $\quad\quad t_e \leftarrow$ *GET-CURRENT-TIME()*;
15: $\quad\quad c \leftarrow t_e - t_s$;
16: $\quad\quad k \leftarrow k + 1$;
17: $\quad$ else:
18: $\quad\quad \Delta w^r_t \leftarrow w^r_{t,k} - w_t$;
19: $\quad\quad aggregated\_updates \leftarrow$ *GLOBAL-SUM-UPDATES*($r, \Delta w^r_t$);
20: $\quad\quad w_{t+1} \leftarrow w_t + \epsilon \cdot aggregated\_updates/p$;
21: $\quad\quad t \leftarrow t + 1$;
22: $\quad\quad w^r_{t,0} \leftarrow w_t$;
23: $\quad\quad k \leftarrow 0$;
24: $\quad\quad t_e \leftarrow$ *GET-CURRENT-TIME*();
25: $\quad\quad$ *RESET-STATE-SERVER*(r, t, t_e);

Output: Trained model parameters w_t.

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 or MPI Allreduce, such as Ring Allreduce, Butterfly, Recursive Doubling, Three-Phase Allreduce, Pipeline etc.

3.2 Convergence Analysis

In this section, we analyze the convergence rate of *ESync* and compare with the standard synchronous mini-batch SGD[16]. 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.

Proposition 1. *Let $L(w, z)$ be an L-smooth convex loss function in w for each $z \in D$ and assume that the stochastic gradient $\nabla_w L(w, z)$ has σ^2-bounded variance for all $w \in W$. If the update rule used in a serial setting has an expected optimality gap bounded by $\tilde{\psi}(\sigma^2, m) = \frac{1}{m}\psi(\sigma^2, m)$, then the expected optimality gap of Algorithm 1 after processing m samples is at most*

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

specially, if $\psi(\sigma^2, m) = 2D^2L + 2D\sigma\sqrt{m}$, the expected optimality gap is bounded by

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

Proof of Proposition 1 is given in Appendix A. We can see that our bound is asymptotically equivalent to the bound of standard synchronous mini-batch SGD $O(\frac{1}{m} + \frac{1}{\sqrt{m}})$, and *ESync* processes $\bar{n}$ times more samples during the same time, makes it train faster and finally achieve the same accuracy. Besides, *ESync* alleviates the "Large Batch Training" problem (larger batch size often leads to lower test accuracy)[4] by expanding the variance of the stochastic gradients by n_i times on each device.

4 State Server

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

4.1 State Query

Considering that computing capability 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, such as multi-tenant clusters and edge devices in Edge AI.

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 aggregate. Therefore, the "Short Board" problem is not solved because other devices still need to wait until A finishes its new iteration.

A better way is to perform 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 can guarantee that the blocking time of the 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*

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

4.2 Non-Blocking Mode with Delayed ACK

Considering that the clusters may be in a network with high communication delay, and the blocked state query operation forces the devices to wait until the State Server responds signal *READY* or *NOT-READY*, which causes great waste of computational resources. Besides, the available network resources may be heterogeneous and unstable, 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[17], which is a flexible and efficient library for deep learning, and supports distributed deep learning by Parameter Server. We evaluate *ESync* on a heterogeneous cluster with 2 Intel E5-2650v4 CPUs and 4 GTX 1080TI GPUs, and compare its training speed, test accuracy, data throughput, traffic load, communication ratio with standard synchronous parallel algorithm SSGD (Sync) and asynchronous parallel algorithm ASGD (Async).

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. We can also select MPI Allreduce algorithms to perform aggregation according to the number of devices, physical topology and size of the model. Besides, quantization and sparsification techniques can be used to accelerate communication, such as 1-bit SGD[18] and DGC[19].

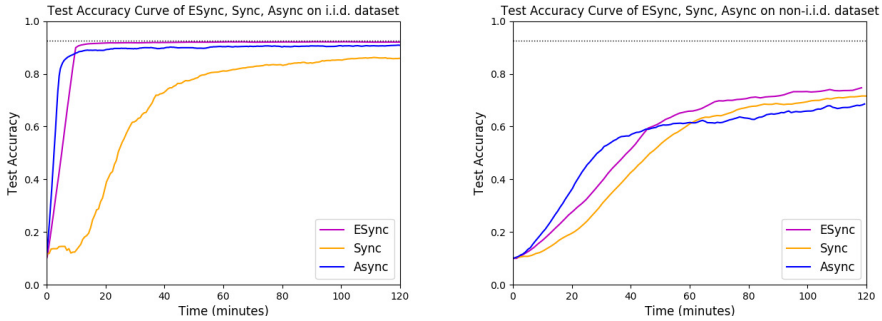

Figure 3: Test accuracy curve on i.i.d and non-i.i.d. Fashion-MNIST dataset.

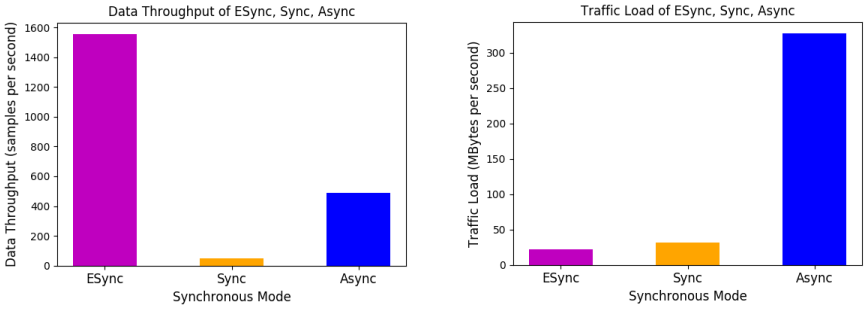

Figure 4: Data throughput and traffic load.

We set learning rate to 0.001 and batch size to 64, and apply mini-batch SGD for all experiments. We plot the curves of test accuracy over time, count the number of samples processed and the amount of data transmitted per second, and count the total time of computing and communication. Besides, we define the speedup as the time ratio to achieve test accuracy 0.8, and the communication time ratio as the proportion of total communication time to total training time.

Test accuracy and training speed. *ESync* greatly increase training speed without loss of accuracy. We distribute the classic dataset Fashion-MNIST[20] to each device in i.i.d. and non-i.i.d. data distributions and train ResNet[21] respectively. Results show that *ESync* achieves the state-of-the-art performance in both settings. As shown in Figure 3, while training on i.i.d. dataset, *ESync* achieves almost the same training speed as Async, which is about 7$\times$ faster than Sync, and achieves the test accuracy of standalone training in the shortest time. Furthermore, while training on non-i.i.d. dataset, *ESync* converges faster than Sync without loss of accuracy while the accuracy of Async is impaired.

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$\times$ more samples than Sync and 3x more samples than Async per second, however, only 1/15 of the data need to be transmitted per second compared to Async. The reason is that *ESync* performs multiple iterations locally, which increases data throughput and reduces the number of communication.

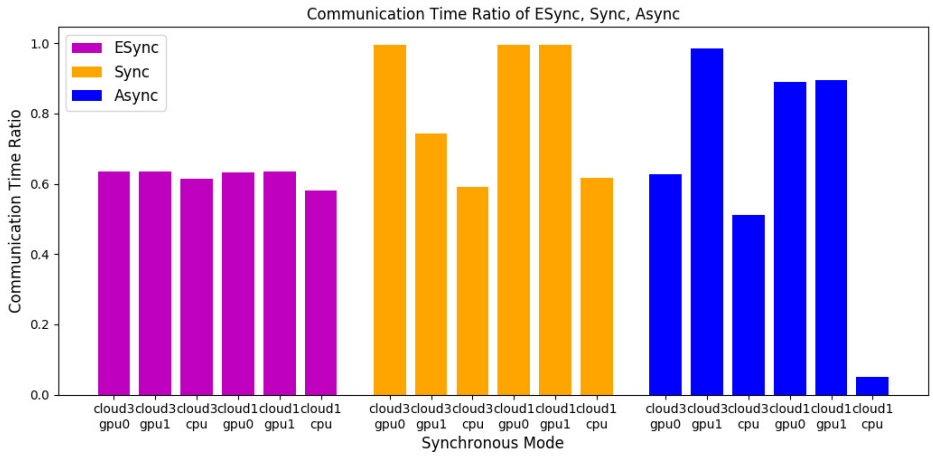

Figure 5: Communication time ratio on each device.

Communication time. *ESync* reduces the time ratio of communication and takes full advantage of the computing capabilities of the heterogeneous clusters. As shown in Figure 5, Sync spends too much time waiting for straggling devices, and Async spends little time on computing but always in communication. *ESync* effectively balances the computation and communication on each device, makes its training process more efficient in clusters with heterogeneous communication resources.

We compare the speedup of *ESync* to Sync and the test accuracy with the standalone training on several classic models, as shown in Table 1. The results show that *ESync* can greatly accelerate the training process and finally achieve the same (sometimes higher) accuracy as the standalone training.

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

Model	Speedup	Test Accuracy	
		ESync	Standalone
AlexNet	27×	**0.929**	0.928
Inception-v3	7×	**0.937**	0.932
ResNet-18-v1	7×	0.926	0.926
ResNet-50-v1	4×	**0.914**	0.910
ResNet-50-v2	6×	0.918	**0.920**
MobileNet-v1	2×	**0.903**	0.902

6 Conclusion

This paper proposes *ESync*, an efficient synchronous parallel algorithm designed for heterogeneous clusters. *ESync* greatly increases training speed without loss of accuracy, and takes full advantage of the computing capabilities of the heterogeneous clusters with lowest traffic load. We also introduce State Server to assign the number of local iterations for each device automatically and dynamically, aim to solve both the "Short Board" and "Delayed Gradient" problems fundamentally in heterogeneous and dynamic clusters. In addition, *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 (e.g. star topology of Parameter Server, ring topology of MPI Ring Allreduce, hierarchical hybrid topologies, and even user-defined topologies).

In future work, we are going to design a communication mechanism based on *ESync* that supports user-defined communication mode (including communication topologies and algorithms), and propose an algorithm for 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] Jia, Xianyan, et al. "Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes." arXiv preprint arXiv:1807.11205, 2018.

[2] Chen, Jianmin, et al. "Revisiting distributed synchronous SGD." arXiv preprint arXiv:1604.00981 (2016).

[3] Konecny, Jakub, et al. "Federated learning: Strategies for improving communication efficiency." arXiv preprint arXiv:1610.05492 (2016).

[4] Keskar, Nitish Shirish, et al. "On large-batch training for deep learning: Generalization gap and sharp minima." arXiv preprint arXiv:1609.04836 (2016).

[5] Dean, Jeffrey, et al. "Large scale distributed deep networks." Advances in neural information processing systems. 2012.

[6] Lian, Xiangru, et al. "Asynchronous parallel stochastic gradient for nonconvex optimization." Advances in Neural Information Processing Systems. 2015.

[7] Avron, Haim, Alex Druinsky, and Anshul Gupta. "Revisiting asynchronous linear solvers: Provable convergence rate through randomization." Journal of the ACM (JACM) 62.6 (2015): 51.

[8] Agarwal, Alekh, and John C. Duchi. "Distributed delayed stochastic optimization." Advances in Neural Information Processing Systems. 2011.

[9] McMahan, Brendan, and Matthew Streeter. "Delay-tolerant algorithms for asynchronous distributed online learning." Advances in Neural Information Processing Systems. 2014.

[10] Sra, Suvrit, et al. "Adadelay: Delay adaptive distributed stochastic convex optimization." arXiv preprint arXiv:1508.05003 (2015).

[11] Zheng, Shuxin, et al. "Asynchronous stochastic gradient descent with delay compensation." Proceedings of the 34th International Conference on Machine Learning-Volume 70. JMLR. org, 2017.

[12] Ho, Qirong, et al. "More effective distributed ml via a stale synchronous parallel parameter server." Advances in neural information processing systems. 2013.

[13] Gibiansky, Andrew. "Bringing HPC techniques to deep learning". http://research. baidu.com/bringing-hpc-techniques-deep-learning/, 2017.2.21.

[14] Rabenseifner, Rolf. "Optimization of collective reduction operations." International Conference on Computational Science, Krakow, Poland, 2004.6.6-6.9.

[15] Gerbessiotis, Alexandros V., and Leslie G. Valiant. "Direct bulk-synchronous parallel algorithms." Journal of parallel and distributed computing 22.2 (1994): 251-267.

[16] Dekel, Ofer, et al. "Optimal distributed online prediction using mini-batches." Journal of Machine Learning Research 13.Jan (2012): 165-202.

[17] Chen, Tianqi, et al. "Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems." arXiv preprint arXiv:1512.01274 (2015).

[18] Seide, Frank, et al. "1-bit stochastic gradient descent and its application to data-parallel distributed training of speech dnns." Fifteenth Annual Conference of the International Speech Communication Association. 2014.

[19] Lin, Yujun, et al. "Deep gradient compression: Reducing the communication bandwidth for distributed training." arXiv preprint arXiv:1712.01887 (2017).

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

[21] He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

[22] He, Kaiming, et al. "Identity mappings in deep residual networks." European conference on computer vision. Springer, Cham, 2016.

[23] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012.

[24] Szegedy, Christian, et al. "Rethinking the inception architecture for computer vision." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

[25] Howard, Andrew G., et al. "Mobilenets: Efficient convolutional neural networks for mobile vision applications." arXiv preprint arXiv:1704.04861 (2017).