Federated Learning Presentation

Paper: ESync: Accelerating Intra-Domain Federated Learning in Heterogeneous Data Centers

Publisher Info: TSC20, Journal Impact 5.69

Reporter: Lee Sen.J

Date: 17th / October / 2024

Outline

1. Background & Related Works
2. Architecture Design
3. Modeling & Algorithm
4. Convergence Analysis
5. Experimental Evaluation
6. Appreciation

Background

Today's Federated Learning

- McMahan *et al*. proposed FL to realize privacy-preserving collaborative learning.
- Cross-device FL & Cross-Silo FL
  - Focus on scenarios where clients and parties communicate ML models with the central server over bandwidth-limited **inter-domain** networks (cross WAN),
  - Isolated parties (scale at hundreds or thousands),
  - Uneven and non-i.i.d data distribution.
- Novel scenarios: **intra-domain FL** in heterogeneous data centers
  - Sufficient bandwidth,
  - Highly heterogeneous computing power.
    - ***We need to make improvements!***

Existing Literatures' Methods

- **Sync**hronous methods:
  - **FedAvg**: poor convergence rate(1400:36), straggler problem, non-i.i.d problem.
  - **FedDrop**: time window,
  - **TiFL**: grouped into multi-tiers, straggler less possible to contribute,
  - **SSGD**: (traditional DML) free of communication bottleneck, but bottlenecked by strong computational heterogeneity, has the same convergence rate on non-i.i.d. data as on i.i.d. data
- **Async**hronous methods:
  - **ASGD**: natural adaptable to computaional heterogeneity, but poor accuracy,
  - **FedAsync**: update the global model immediately by weighted averaging, introduces a mixing hyperparameter to mitigate the error caused by staleness.
  - **DC-ASGD**: compensate stale gradients using Talor expansion.

Goals of the Work

To address the stragglers in the strongly heterogeneous *intra-domain FL*, while maintaining accuracy without loss, adaptively avoid blocking caused by stragglers.

- A novel **scheduler**.
- An efficient **synchronization algorithm** (based on SSGD).
- Analyze the **trade-off** between convergence accuracy and communication efficiency.
- Extensive experimental evaluation on real-world datasets.

Architecture Design

Overview

- **State Server**: monitor the status and progress, coordinate the synchronization pace of all workers.
- **Parameter Server**: average the updates and synchronize the averaged updates to all workers.
- **Workers**: perform local updates and send their updates to the parameter server.
  - up to State Server's instruction.

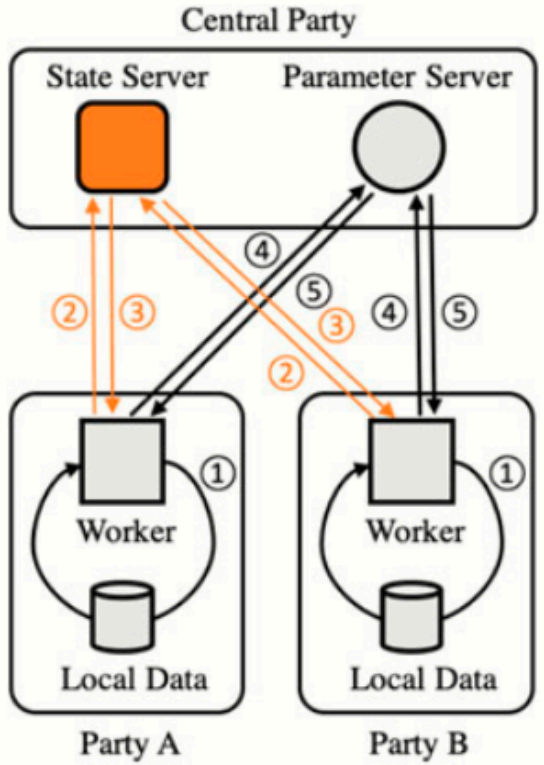

Fig. 2. Overview of the proposed architecture. ① The worker trains a local model on local data. ② The worker queries the next action from the state server. ③ The state server responds with the next action. ④ The worker pushes an update to the parameter server. ⑤ The worker pulls the averaged update from the parameter server.

State Server

State Server contains

- a **message receiver**, a **message sender**,
- a FIFO **message queue**,
  - buffer the received messages.
- a **message router**,
  - forward to the handlers accoring to `msg_type` field.
- a **state database**,
  - lock-free state table.
- and a set of **message handlers**.

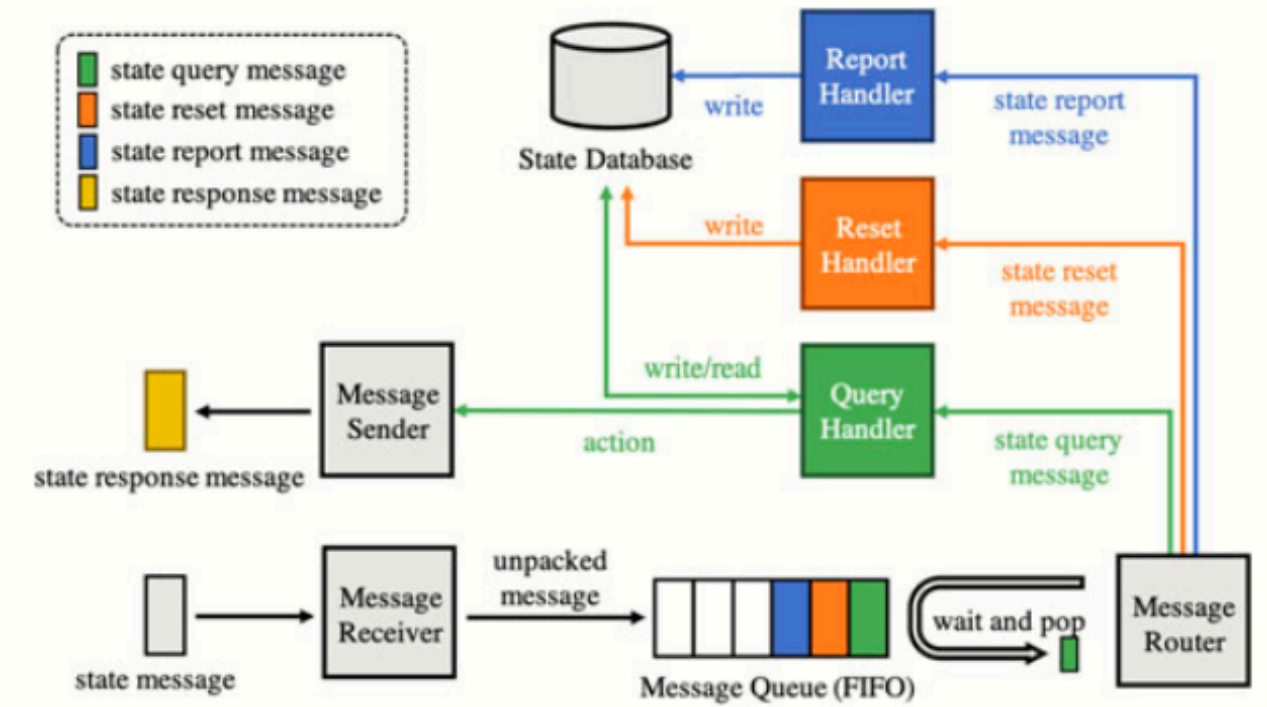

Fig. 3. Implementation of state server.

Message Format & Message Types

- **state reset message**: initiated by the Parameter Server, to initialize the records in the state database.
- **~ report msg**: initiated by the worker to synchronize its status and progress to State Server.
- **~ query msg**: initiated by the worker to query State Server for the next action.
- **~ response msg**: used by State Server to inform the querying worker of the next action.

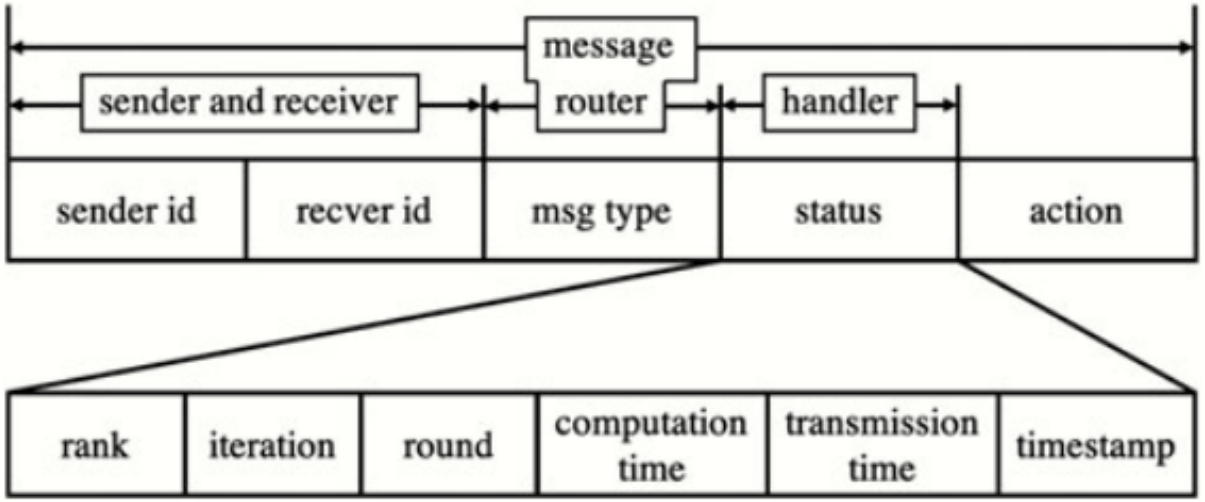

Fig. 4. The structure of message.

State Database

State database contains

- a **task queue**,
  - buffer the received tasks from handlers.
- a **task engine**,
  - multi-threaded asynchronous task processor,
  - process the tasks in the queue according to state table.
- a **state table**,
  - record the status and next action of all workers.

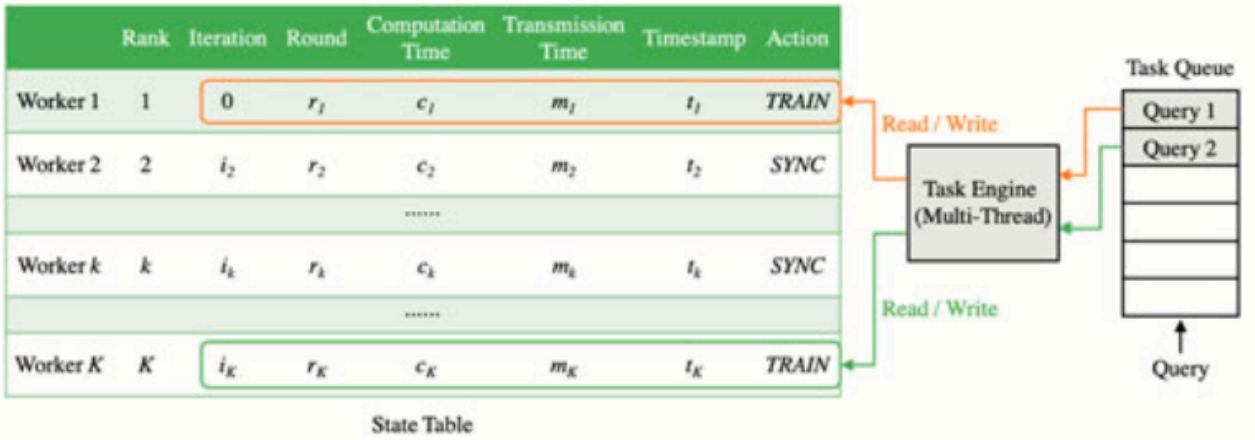

Fig. 5. Implementation of the state database.

How to find stragglers and determine the number of local iteration?

Suppose the rank of straggler is $s = 1$, and define delay $d_k = c_k + m_k$. To decide the next action for the querying worker k , the task engine uses the status $(k, i_k, r_k, c_k, m_k, t_k)$ attached to the msg to update the state table, traverse it to find the straggler:

$$s = \arg\max_k d_k, \quad (k \in [1, K]), \tag{1}$$

return the status $(s, i_s, r_s, c_s, m_s, t_s)$ to the query handler, it will predict the moment $\hat{t}_s$ when the straggler s submits its update: $\hat{t}_s = t_s + d_s$,
Then, checks whether the querying worker k can perform another iteration before $\hat{t}_s$:

$$t_c + d_k \leq \hat{t}_s, \text{ where } t_c \text{ is the current timestamp.} \tag{3}$$

Modeling and Algorithm

Problem Modeling

The goal of this paper is to minimize (a) **the global loss** and (b) **the blocking time**.

Minimize the global loss

Define a function f that maps input x to a probability vector $\tilde{y}$ using weights w and a classification c probability f_c using cross-entropy loss l as the loss function:

$$l(w, x, y) = \sum_{c=1}^{C} p(y = c)[\log f_c(x, w)]. \tag{4}$$

$l(w, x, y)$ can be extend to local loss $l_k(w_k, x_k, y_k)$ of worker k, and global loss $l_g(w, x, y)$.

Therefore, the first goal (a) is to minimize the global loss $l_g(w, x, y)$, and find the optimal weights w^*:

$$w^* = \arg\min_w l_g(w, \mathcal{X}, \mathcal{Y}) \tag{7}$$

Minimize the blocking time

For party k, the delay d_k is the sum of its computation and transmission time $d_k = c_k + m_k$. Define the computational heterogeneity as $h_c = \frac{c_{max}}{c_{min}}$ and the communication heterogeneity as $h_m = \frac{m_{max}}{m_{min}}$. Then, the blocking time of the party k can be defined as:

$$d_k^{\text{wait}} = c_s + q_s + m_s - i_k(c_k + q_k) - m_k, \tag{8}$$

Since the data center network has a strong computational heterogeneity ($h_c \leq 300$), weak communication heterogeneity ($h_m \approx 1$), and sufficient bandwidth, we simplify (8) by $q_s = q_k \approx 0$ and $m_s = m_k$.

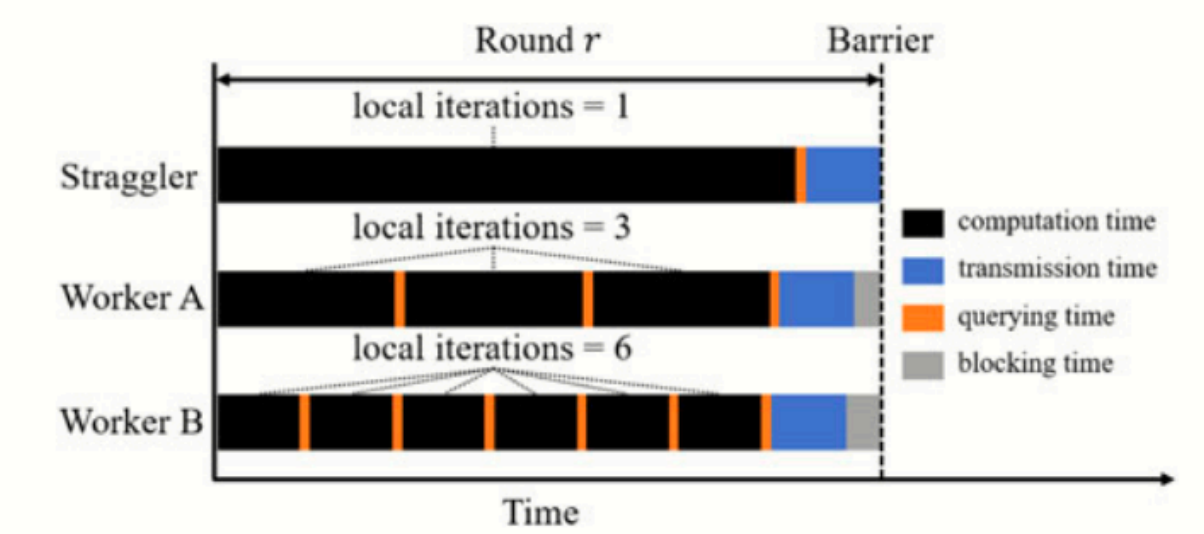

Fig. 7. ESync timeline with adaptive local iterations.

Fig. 7 shows the timeline of the proposed algorithm ESync. The core idea is to make use of the blocking time to train more iterations.

Therefore, the second goal (b) is to find a strategy π to coordinate the number of local iterations $i_k(\pi)$ for each worker to minimize the overall blocking time d^{wait}:

$$\underset{\pi}{\text{minimize}} \quad d^{\text{wait}} = \sum_{k \in [1,K], k \neq s} (c_s - i_k(\pi) c_k). \tag{9}$$

Algorithm Design

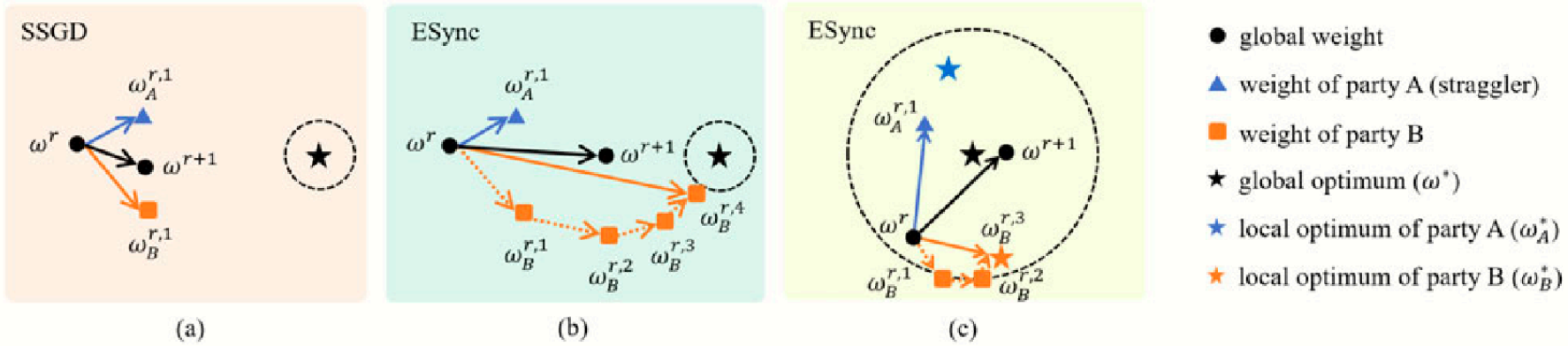

Fig. 6. Illustration of (a) SSGD and (b) ESync in the early training, and (c) ESync in the later training.

As shown in Fig 6(a), SSGD forces all workers to synchronize their local models in each iteration, and the powerful workers are blocked until the straggler uploads its update.

ESync allows the powerful workers to train as many iterations as possible without waiting for the straggler, where the number of local iterations i_k is adaptively coordinated by State Server.

Fig 6(b) shows a round $r(r \to 0)$ in the early training, where the powerful party B explores a local weight $w_B^{r,4}$ that is closer to the global optimum w^* after multiple local iterations, and pushes the local weight $w_A^{r,1}$ of the straggler party A closer to the global optimum w^* , thus speeding up convergence and avoiding blocking.

Meanwhile, ESync will not fall into the local optimum of the powerful party in the later training ($r \to R$), as shown in Fig 6(c), the local model $w_B^{r,3}$ of the powerful party B has reached its local optimal w_B^* after several rounds of iterations, but the straggler party A is still slowly pushing the global model w^{r+1} leaves the local optimum of B (because the $|w^r - w_A^*|$ gap is too large and clearly erroneous), and eventually the global model w^R reaches a dynamic equilibrium near the global optimal model w^*.

Pseudo-code

MyBlog:

https://senjlearning.space/4-Scholar/1-Data-Center/Federated-Learning/Esync：notes

Convergence Analysis

Algorithm 1. *(State Server) StateQuerying*

Input: The state table $S = [(p, i_p, r_p, c_p, m_p, t_p, a_p)]_{\forall p \in [1,K]}$; The status $(k, i_k, r_k, c_k, m_k, t_k)$ attached in the state query message from the querying worker k.

Update the record of worker k in the state table S, $S[k].iteration = i_k, S[k].computation\_time = c_k, S[k].transmission\_time = m_t, S[k].timestamp = t_k$;

Find the straggler s by Eq. (1);

// The worker k has just entered the new round, or the straggler s has not yet entered the new round.

if $i_k == 0$ **or** $r_k > r_s$:

  return $a_k = TRAIN$;

$t_c = CurrentTime()$;

// The straggler s has completed the local training, or the worker k cannot perform another iteration before the straggler s completed the local training.

if $k == s$ **or** $i_s == 1$ **or** $a_s == SYNC$ **or** ineq. (3) does not hold:

  Update the action of worker k, $S[k].action = SYNC$;

  return $a_k = SYNC$;

return $a_k = TRAIN$;

Output: A state response message with action a_k.

Experimental Evaluation

Experimental Setup

- **Environment Setup**: in a shared data center with a 1 Gbps bandwidth LAN network, use 3 machines to simulate 12 isolated parties, each machines have 2 GTX 1080TI GPUs and 2 Intel E5-2650v4 CPUs, run 12 docker containers for 12 parties respectively, each party contributes 1 worker. Computational heterogeneity default to be $h_c = 150$.
- **Models and Datasets**: AlexNet, ResNet, Inception, CIFAR10, FashionMNist.
- **Benchmark Algorithms**: traditional DML (SSGD, ASGD, DC-ASGD), cross-silo FL (FedAvg, FedDrop, TiFL, FedAsync).
- **Training Hyperparameters**: use SGD optimizer for local training and set the local learning rate $\eta = 0.001$, the global learning rate $\epsilon = 1$, the batch size $b = 32$, and the number of local epochs $E = 1$ by default in the following experiments.

Numerical Results

ESync vs. Traditional DML

Training Efficiency.

> ESync can achieve greater acceleration on more complex tasks, among which the training efficiency of ESync far exceeds SSGD, ASGD, and DC-ASGD.

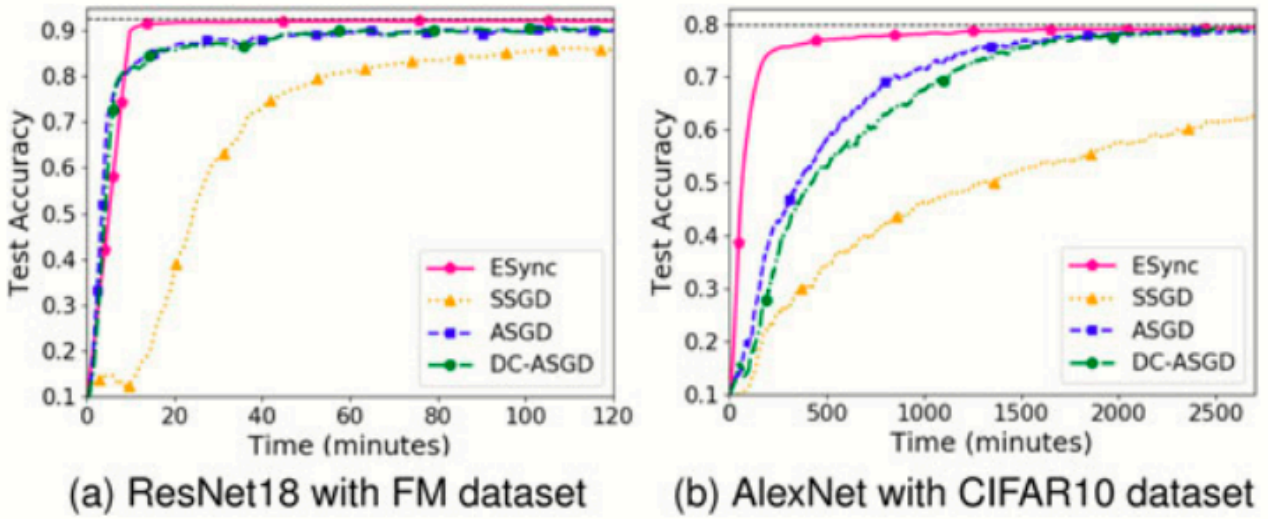

Fig. 9. Test accuracy of ESync, SSGD, ASGD, DC-ASGD on (a) ResNet18 with i.i.d. Fashion-MNIST dataset and (b) AlexNet with i.i.d. CIFAR10 dataset.

Time Reduction.

- ESync can achieve up to 95% time reduction compared to SSGD on AlexNet.
- ESync uses blocking time to process more samples, which helps to explore more accurate models, and therefore converges faster than SSGD.

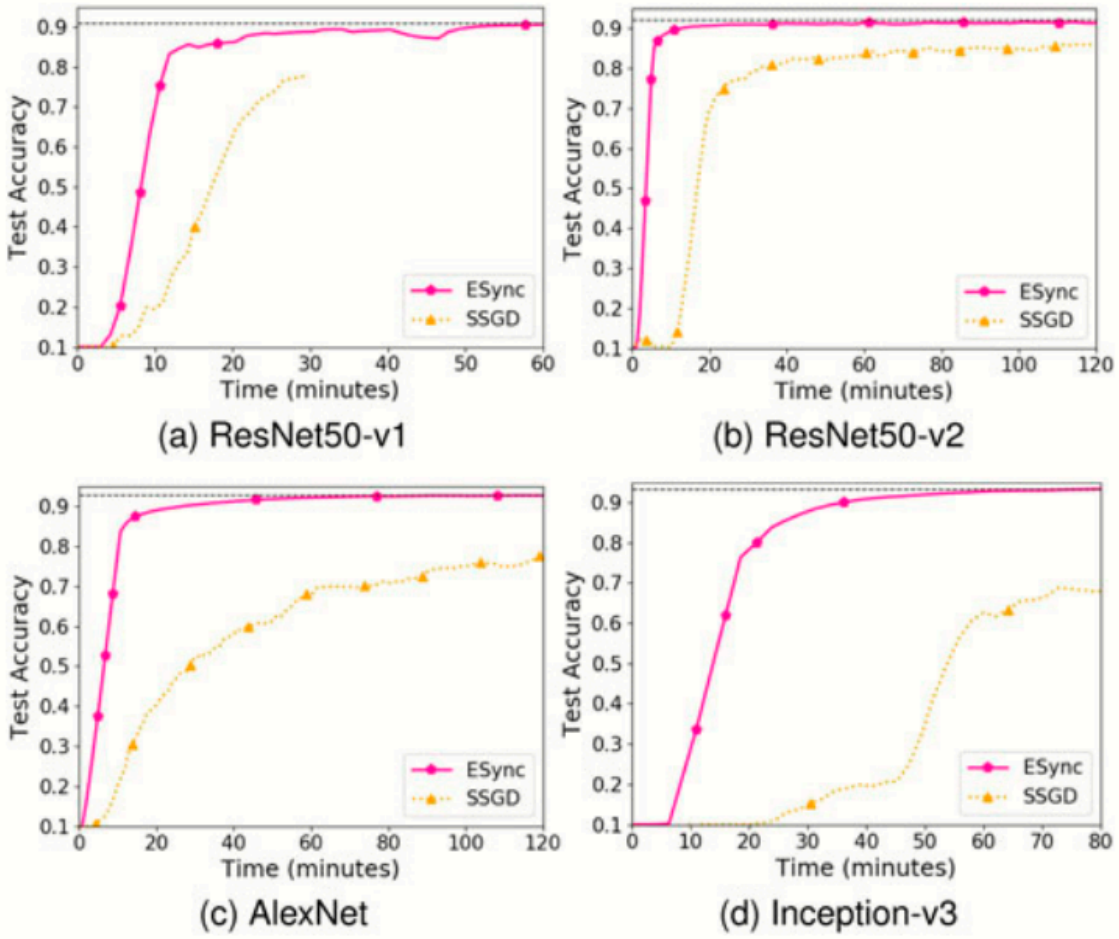

Fig. 10. Test accuracy of ESync compared with SSGD on (a) ResNet50-v1, (b) ResNet50-v2, (c) AlexNet, and (b) Inception-v3.

Converged Accuracy.

- ESync does not impair the converged accuracy, and surprisingly, ESync even achieves higher converged accuracy than SSGD.
- The reason may be that the weight divergence of ESync introduces a moderate amount of noise which helps the global model escape from the sharp minimum to the flat minimum, where the flat minimum tends to have better generalization.

TABLE 3
Time Reduction and Converged Accuracy on Different Models

Model	Time to Reach 0.8 (min)		Converged Accuracy	
	SSGD	ESync	CSGD	ESync
AlexNet	148	**6 (-96%)**	0.928	**0.932**
Inception-v3	208	**21 (-90%)**	0.932	**0.940**
ResNet18-v1	53	**8 (-85%)**	0.919	**0.926**
ResNet50-v2	33	**5 (-85%)**	0.918	0.918
ResNet50-v1	32	**9 (-72%)**	0.910	**0.914**

Non-i.i.d. Performance.

> The non-i.i.d. data magnifies the impact of delayed gradients of ASGD and DC-ASGD, and leads to a sharp drop in training efficiency. Although ASGD and DC-ASGD still have high efficiency in the early training, their accuracy is overtaken by SSGD soon, thus losing the advantage of efficiency.

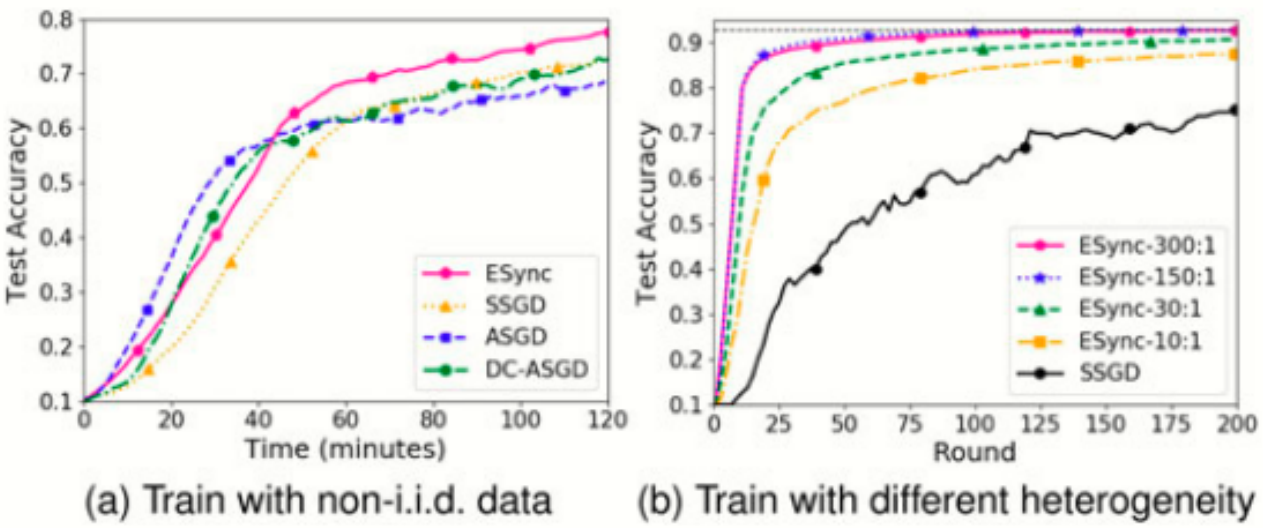

Fig. 11. Test accuracy of ESync with (a) non-i.i.d. Fashion-MNIST data-set and (b) different computational heterogeneity.

Different Heterogeneity.

> Fig 11(b) shows the curve of test accuracy over rounds on ResNet18 model with i.i.d. Fashion-MNIST dataset, where h_c increases from 10 to 300. The results show that the training efficiency of ESync continues to improve with the increase of h_c, and finally stabilizes at the optimal efficiency. We find that ESync does not bring accuracy loss even in the setting of $h_c = 300 : 1$.

ESync vs. Cross-Silo FL Algorihtms

ESync vs. FedAsync.

> FedAsync avoids blocking through asynchronous updating, and achieves higher efficiency than FedAvg in the early training. However, the strong computational heterogeneity $h_c = 150 : 1$ exacerbates the impact of delayed updates and severely destroys the accuracy, ultimately making FedAsync overtaken by FedAvg.

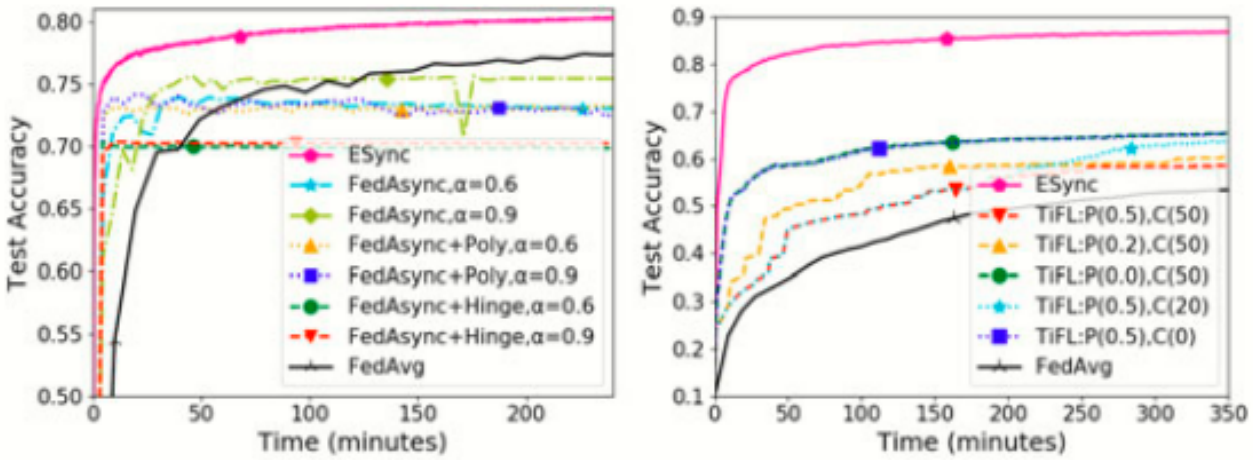

Fig. 12. Test accuracy of ESync compared with (a) FedAsync, (b) TiFL.

> Instead, ESync is superior to FedAvg and FedAsync in terms of both efficiency and accuracy under all recommended settings, because ESync is a synchronous algorithm with no delayed updates.

ESync vs. TiFL.

> ESync maximizes the overlap time between stragglers and powerful workers, and the computation time of stragglers can be considered as not included in the overall training time, so ESync can achieve higher training efficiency than TiFL.

ESync vs. FedDrop.

- FedDrop can effectively speed up early training by dropping the stragglers, but it prematurely converges to a low accuracy.
- Instead, ESync is still more efficient and accurate than FedDrop, because it does not drop workers and has better convergence accuracy than FedAvg.

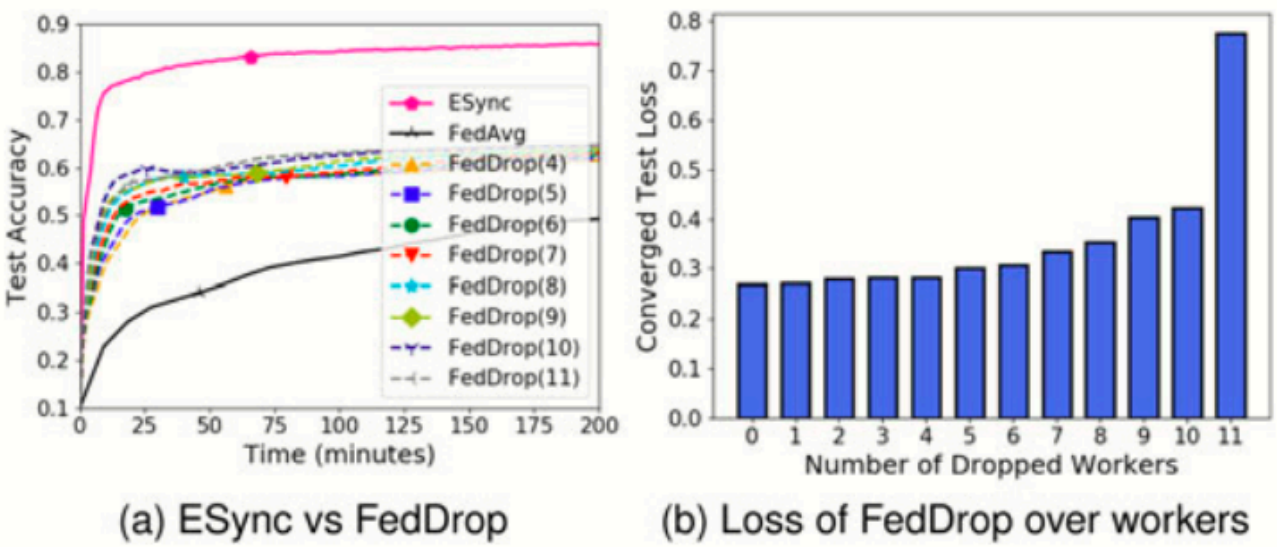

Fig. 13. Comparison of (a) ESync and FedDrop and (b) the test loss of FedDrop over the number of dropped workers.

Thanks for your attention!