Ordered Local Momentum for Asynchronous Distributed Learning under Arbitrary Delays

Chang-Wei Shi[1], **Shi-Shang Wang**[1], **Wu-Jun Li**[1,2*]

[1] National Key Laboratory for Novel Software Technology, School of Computer Science, Nanjing University
[2] Pazhou Laboratory (Huangpu), Guangdong, China
{shicw, wangs}@smail.nju.edu.cn, liwujun@nju.edu.cn

Abstract

Momentum SGD (MSGD) serves as a foundational optimizer in training deep models due to momentum's key role in accelerating convergence and enhancing generalization. Meanwhile, asynchronous distributed learning is crucial for training large-scale deep models, especially when the computing capabilities of the workers in the cluster are heterogeneous. To reduce communication frequency, local updates are widely adopted in distributed learning. However, how to implement asynchronous distributed MSGD with local updates remains unexplored. To solve this problem, we propose a novel method, called ordered local momentum (OrLoMo), for asynchronous distributed learning. In OrLoMo, each worker runs MSGD locally. Then the local momentum from each worker will be aggregated by the server in order based on its global iteration index. To the best of our knowledge, OrLoMo is the first method to implement asynchronous distributed MSGD with local updates. We prove the convergence of OrLoMo for non-convex problems under arbitrary delays. Experiments validate that OrLoMo can outperform its synchronous counterpart and other asynchronous methods.

Introduction

Stochastic gradient descent (SGD) (Robbins and Monro 1951) and its variants (Polyak 1964; Kingma and Ba 2015) are commonly used in training deep models. At each iteration, SGD uses a stochastic gradient as an approximation of the full gradient to update the model parameter. In practice, momentum SGD (MSGD) (Polyak 1964) is widely adopted by incorporating momentum into SGD to accelerate convergence and enhance generalization. Furthermore, momentum has been a core component in many popular adaptive optimizers, such as Adam (Kingma and Ba 2015) and AMSgrad (Reddi, Kale, and Kumar 2018).

As both models and datasets in deep learning continue to grow rapidly, the demand of training computation has surpassed the capacity of single devices. Distributed learning (Yang et al. 2019; Verbraeken et al. 2020) addresses this challenge by distributing computation across multiple workers. Existing distributed learning methods primarily fall into two categories: synchronous distributed learning (SDL) methods and asynchronous distributed learning (ADL) methods. SDL methods (Lin et al. 2018; Vogels, Karimireddy, and Jaggi 2019; Yu, Jin, and Yang 2019; Wang et al. 2020) require all workers to synchronize at each communication round. Thus, SDL methods face the challenge of idling, as the fast workers must wait for the slow workers. This challenge is especially severe in clusters with heterogeneous computing capabilities. ADL methods (Agarwal and Duchi 2011; Lian et al. 2018; Yang and Li 2021; Shi, Yang, and Li 2024) mitigate this by allowing workers to proceed independently since only information from a subset of workers is required to be aggregated per round. Representative ADL methods include asynchronous SGD (ASGD) and its variants (Agarwal and Duchi 2011; Lian et al. 2015; Zheng et al. 2017; Arjevani, Shamir, and Srebro 2020; Maranjyan, Tyurin, and Richtárik 2025). While ADL methods avoid idle time, they introduce convergence challenges due to the delay of updates. Existing convergence analyses for ADL methods typically rely on additional assumptions about the delays (e.g., maximum delay bounds), which can't fully capture the actual behavior of ADL methods in practice. The convergence proof for ADL methods under arbitrary delays was a long-standing open problem until recently. The works in (Koloskova, Stich, and Jaggi 2022; Mishchenko et al. 2022) establish a convergence guarantee for ASGD under arbitrary delays and prove that ASGD achieves the same convergence rate as synchronous SGD in terms of gradient computations. The work in (Shi, Yang, and Li 2024) proposes a momentum variant of ASGD called OrMo-DA, and provides a convergence proof for OrMo-DA under arbitrary delays.

To reduce communication frequency, local updates are widely adopted in distributed learning. The most representative methods using local updates are local SGD (Lin et al. 2019; Stich 2019) and its variants (Basu et al. 2019; Karimireddy et al. 2020; Crawshaw and Liu 2024). Local SGD is also the key component of FedAvg in federated learning (McMahan et al. 2017). Local SGD enables each worker to execute multiple SGD iterations instead of one iteration before communicating with the server or the other workers, thereby reducing communication frequency. Several works in (Lin et al. 2019; Stich 2019; Haddadpour et al. 2019; Yu, Yang, and Zhu 2019) establish the convergence guarantee for local SGD. The work in (Yu, Jin,

*Corresponding author.

and Yang 2019) proposes parallel restarted SGD with momentum (PRSGDm) as a momentum variant of local SGD and proves its convergence with a linear speedup property. Most existing works about local SGD and its variants focus on SDL methods, which face the challenge of idling while waiting for slow workers. Several works (Xie, Koyejo, and Gupta 2019; Nguyen et al. 2022; Wang et al. 2025) have studied asynchronous local SGD (AL-SGD), which implements asynchronous distributed SGD with local updates. While momentum is widely recognized as crucial for deep model training, no research has studied how to implement asynchronous distributed MSGD with local updates. Furthermore, the convergence guarantee of asynchronous distributed MSGD with local updates remains an open problem, particularly under arbitrary delays caused by the heterogeneous computing capabilities in the cluster.

In this paper, we propose a novel method, called ordered local momentum (OrLoMo), for asynchronous distributed learning. Our contributions are outlined as follows:

- In OrLoMo, each worker runs MSGD locally. Then the local momentum from each worker will be aggregated by the server in order based on its global iteration index. To the best of our knowledge, OrLoMo is the first method to implement asynchronous distributed MSGD with local updates.
- We prove the convergence of OrLoMo for non-convex problems under arbitrary delays. Our convergence analysis doesn't rely on the maximum delay or bounded gradient assumptions which are commonly used in existing analyses of ADL methods.
- Experiments validate that OrLoMo can outperform its synchronous counterpart and other asynchronous methods.

Preliminary

In this section, we introduce the problem formulation and then present two classic algorithms, including momentum SGD (MSGD) and asynchronous local SGD (AL-SGD).

Problem Formulation

Many machine learning model training tasks can be formulated as the following optimization problem:

$$\min_{\mathbf{w}\in\mathbb{R}^d} F(\mathbf{w}) = \mathbb{E}_{\xi\sim\mathcal{D}} f(\mathbf{w};\xi), \tag{1}$$

where $\mathbf{w}$ is the d-dimensional model parameter, ξ denotes a training instance sampled from distribution $\mathcal{D}$ and $f(\mathbf{w};\xi)$ is the loss function evaluated at parameter $\mathbf{w}$ for instance ξ. $\nabla f(\mathbf{w};\xi)$ denotes the stochastic gradient. $\|\cdot\|$ denotes the L_2 norm. $[n]$ denotes $\{0,1,\ldots,n-1\}$, where $n\in\mathbb{N}^*$.

In this paper, we focus on the widely used parameter server (PS) framework (Li et al. 2014). In the PS framework, the workers are responsible for sampling training instances and computing stochastic gradients, and the server is responsible for storing and updating the model parameters. Without loss of generality, we assume that each worker performs stochastic gradient computation by sampling a single training instance per iteration. The analysis of mini-batch sampling follows a similar approach.

Algorithm 1: AL-SGD

Input: number of workers K, number of global iterations T, number of local iterations S, global learning rate η_t, local learning rate γ;
Server:
Initialization: global parameter $\mathbf{w}_0$;
Send $\mathbf{w}_0$ to all workers;
for $t=0$ **to** $T-1$ **do**
  Receive $\Delta\mathbf{w}^{k_t}_{ite(t,k_t)}$ from some worker k_t;
  $\mathbf{w}_{t+1}=\mathbf{w}_t-\eta_t\Delta\mathbf{w}^{k_t}_{ite(t,k_t)}$;
  Send $\mathbf{w}_{t+1}$ back to worker k_t;
end for
Notify all workers to stop;
Worker k **:** $(k\in[K])$
repeat
  Wait until receiving $\mathbf{w}_{t'}$ from the server;
  Initialize local variables $\tilde{\mathbf{w}}^k_{t',0}=\mathbf{w}_{t'}$;
  for $s=0$ **to** $S-1$ **do**
    Randomly sample $\xi\sim\mathcal{D}$ and then compute the stochastic gradient $\mathbf{g}^k_{t',s}=\nabla f(\tilde{\mathbf{w}}^k_{t',s};\xi)$;
    $\tilde{\mathbf{w}}^k_{t',s+1}=\tilde{\mathbf{w}}^k_{t',s}-\gamma\mathbf{g}^k_{t',s}$;
  end for
  $\Delta\mathbf{w}^k_{t'}=\tilde{\mathbf{w}}^k_{t',0}-\tilde{\mathbf{w}}^k_{t',S}$;
  Send $\Delta\mathbf{w}^k_{t'}$ to the server;
until receive server's notification to stop

Momentum SGD

The widely used momentum SGD (MSGD) (Polyak 1964) can be formulated as follows:

$$\begin{aligned}\mathbf{u}_{t+1}&=\beta\mathbf{u}_t+\gamma\mathbf{g}_t,\\ \mathbf{w}_{t+1}&=\mathbf{w}_t-\mathbf{u}_{t+1},\end{aligned} \tag{2}$$

where $\mathbf{g}_t=\nabla f(\mathbf{w}_t;\xi),\mathbf{u}_0=\mathbf{0},\beta\in[0,1),t\in[T]$ and T denotes the number of iterations. β is the momentum coefficient. γ is the learning rate. When $\beta=0$, MSGD degenerates to vanilla SGD. $\mathbf{u}_t$ represents Polyak's momentum, which is a weighted sum of gradients in a strictly ordered manner. Specifically, gradients with smaller iteration indexes are assigned progressively smaller weights. For example, $\mathbf{u}_4=\beta^3\gamma\mathbf{g}_0+\beta^2\gamma\mathbf{g}_1+\beta^1\gamma\mathbf{g}_2+\beta^0\gamma\mathbf{g}_3$.

Asynchronous Local SGD

Algorithm 1 outlines the asynchronous local SGD (AL-SGD) method. AL-SGD serves as the foundational case of FedBuff (Nguyen et al. 2022) without federated learning components, including the buffer, partial client participation, and differential privacy mechanism. Our work focuses on distributed learning in data-center settings. When $S=1$, AL-SGD degenerates to ASGD (Mishchenko et al. 2022).

For each worker k ($k\in[K]$), after receiving the latest global parameter from the server, the worker conducts S SGD iterations locally, and then sends the local parameter update $\Delta\mathbf{w}^k_{t'}$ to the server. For the server, after receiving the local parameter update $\Delta\mathbf{w}^{k_t}_{ite(t,k_t)}$ from some worker k_t, the server will update the global parameter using $\Delta\mathbf{w}^{k_t}_{ite(t,k_t)}$

and then send the latest global parameter $\mathbf{w}_{t+1}$ back to the worker k_t immediately. To distinguish them from worker-side local iterations, the server-side iterations are termed global iterations. Here, k_t denotes the index of the worker from which the local parameter update is used at global iteration t, and $ite(t, k_t)$ denotes the global iteration index of $\Delta\mathbf{w}^{k_t}_{ite(t,k_t)}$. The function $ite(t, k)$ denotes the global iteration index of the latest parameter sent to worker k before global iteration t. $ite(t, k)$ can be formulated as follows:

$$ite(t+1,k) = \begin{cases} t+1 & k = k_t, \\ ite(t,k) & k \neq k_t, \end{cases}$$

where $k \in [K]$, $t \in [T]$ and $ite(0, k) = 0, \forall k \in [K]$. We define $\tau_t = t - ite(t, k_t)$ as the delay of $\Delta\mathbf{w}^{k_t}_{ite(t,k_t)}$.

In AL-SGD, upon receiving a local parameter update from any worker, the server immediately updates the global parameter and then sends it back to the worker. This asynchronous nature avoids straggler bottlenecks by not waiting for slower workers. In contrast, synchronous local SGD (Stich 2019) requires the server to aggregate the information from all workers before performing global parameter updates and broadcasting. Consequently, its training speed is constrained by the slowest worker in the cluster.

Methodology

In this section, we introduce our proposed method OrLoMo, including the algorithm details and convergence analysis.

OrLoMo

The details of OrLoMo are presented in Algorithm 2.

For each worker, upon receiving the latest global parameter $\mathbf{w}_{t'}$ from the server, the local parameter $\tilde{\mathbf{w}}^k_{t',0}$ and local momentum $\tilde{\mathbf{u}}^k_{t',0}$ are initialized as $\tilde{\mathbf{w}}^k_{t',0} = \mathbf{w}_{t'}, \tilde{\mathbf{u}}^k_{t',0} = \mathbf{0}$. Then, the worker performs S MSGD updates locally. After completing S local updates, the worker sends both local momentum $\Delta\mathbf{u}^k_{t'} = \tilde{\mathbf{u}}^k_{t',S}$ and local parameter update $\Delta\mathbf{w}^k_{t'} = \tilde{\mathbf{w}}^k_{t',0} - \tilde{\mathbf{w}}^k_{t',S}$ back to the server. Here, t' denotes the global iteration index of both $\Delta\mathbf{u}^k_{t'}$ and $\Delta\mathbf{w}^k_{t'}$.

All the local momentums computed in OrLoMo can be given by:

$$\Delta\mathbf{u}^0_0, \Delta\mathbf{u}^1_0, \cdots, \Delta\mathbf{u}^{K-1}_0, \Delta\mathbf{u}^{k_0}_1, \Delta\mathbf{u}^{k_1}_2, \Delta\mathbf{u}^{k_2}_3, \cdots.$$

We group these local momentums in order based on their global iteration indexes. Each group contains K local momentums. The i-th group ($i \geq 1$) in OrLoMo is defined as:

$$\left\{\Delta\mathbf{u}^{k_{(i-1)K}}_{(i-1)K+1}, \Delta\mathbf{u}^{k_{(i-1)K+1}}_{(i-1)K+2}, \cdots, \Delta\mathbf{u}^{k_{iK-1}}_{iK}\right\},$$

and the 0-th group is defined as $\left\{\Delta\mathbf{u}^0_0, \Delta\mathbf{u}^1_0, \cdots, \Delta\mathbf{u}^{K-1}_0\right\}$. Note that local momentums typically arrive at the server out of order due to hardware factors (e.g., workers' computing capabilities). The core of OrLoMo is that the server aggregates local momentums into the global momentum in order based on their global iteration indexes.

The global momentum in OrLoMo is the weighted sum of local momentums received by the server, scaled by their corresponding global learning rates. For simplicity, we refer to local momentum scaled by its global learning rate as scaled local momentum. The weights of scaled local momentums in the global momentum are determined by their global iteration indexes. Before global iteration t, the global parameter $\mathbf{w}_t$ has been sent to some worker. Thus, the global iteration index of the latest local momentum in $\mathbf{u}_{t+1}$ can be t at most. The weight of the scaled local momentum with global iteration index t is defined to be β^0 in $\mathbf{u}_{t+1}$, which belongs to the $\lceil \frac{t}{K} \rceil$-th group. The scaled local momentums from the i-th group is defined to weight $\beta^{\lceil \frac{t}{K} \rceil - i}$ in $\mathbf{u}_{t+1}$, where $0 \leq i \leq \lceil \frac{t}{K} \rceil$. The scaled local momentums from one group share the same weight in the global momentum.

At global iteration t, the server receives local information from some worker k_t, including the local momentum

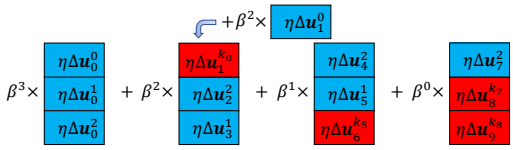
Algorithm 2: OrLoMo

Input: number of workers K, number of global iterations T, number of local iterations S, global learning rate η_t, local learning rate γ, momentum coefficient $\beta \in [0, 1)$;
Server:
Initialization: global parameter $\mathbf{w}_0$, global momentum $\mathbf{u}_0 = \mathbf{0}$;
Send $\mathbf{w}_0$ to all workers;
for $t = 0$ **to** $T - 1$ **do**
  if $\lceil \frac{t}{K} \rceil > \lceil \frac{t-1}{K} \rceil$ **then**
    $\mathbf{w}_{t+\frac{1}{2}} = \mathbf{w}_t - \beta\mathbf{u}_t$, $\mathbf{u}_{t+\frac{1}{2}} = \beta\mathbf{u}_t$;
  else
    $\mathbf{w}_{t+\frac{1}{2}} = \mathbf{w}_t$, $\mathbf{u}_{t+\frac{1}{2}} = \mathbf{u}_t$;
  end if
  Receive $\Delta\mathbf{u}^{k_t}_{ite(t,k_t)}$ and $\Delta\mathbf{w}^{k_t}_{ite(t,k_t)}$ from some worker k_t;
  $\mathbf{u}_{t+1} = \mathbf{u}_{t+\frac{1}{2}} + \beta^{\lceil \frac{t}{K} \rceil - \lceil \frac{ite(t,k_t)}{K} \rceil}\eta_t\Delta\mathbf{u}^{k_t}_{ite(t,k_t)}$;
  $\mathbf{w}_{t+1} = \mathbf{w}_{t+\frac{1}{2}} - \eta_t\Delta\mathbf{w}^{k_t}_{ite(t,k_t)} - \frac{\beta(1-\beta^{\lceil \frac{t}{K} \rceil - \lceil \frac{ite(t,k_t)}{K} \rceil})}{1-\beta}\eta_t\Delta\mathbf{u}^{k_t}_{ite(t,k_t)}$;
  Send $\mathbf{w}_{t+1}$ back to worker k_t;
end for
Notify all workers to stop;
Worker k : ($k \in [K]$)
repeat
  Wait until receiving $\mathbf{w}_{t'}$ from the server;
  Initialize local variables $\tilde{\mathbf{u}}^k_{t',0} = \mathbf{0}, \tilde{\mathbf{w}}^k_{t',0} = \mathbf{w}_{t'}$;
  for $s = 0$ **to** $S - 1$ **do**
    Randomly sample $\xi \sim \mathcal{D}$ and then compute the stochastic gradient $\mathbf{g}^k_{t',s} = \nabla f(\tilde{\mathbf{w}}^k_{t',s}; \xi)$;
    $\tilde{\mathbf{u}}^k_{t',s+1} = \beta\tilde{\mathbf{u}}^k_{t',s} + \gamma\mathbf{g}^k_{t',s}$;
    $\tilde{\mathbf{w}}^k_{t',s+1} = \tilde{\mathbf{w}}^k_{t',s} - \tilde{\mathbf{u}}^k_{t',s+1}$;
  end for
  $\Delta\mathbf{u}^k_{t'} = \tilde{\mathbf{u}}^k_{t',S}$;
  $\Delta\mathbf{w}^k_{t'} = \tilde{\mathbf{w}}^k_{t',0} - \tilde{\mathbf{w}}^k_{t',S}$;
  Send $\Delta\mathbf{u}^k_{t'}$ and $\Delta\mathbf{w}^k_{t'}$ to the server;
until receive server's notification to stop

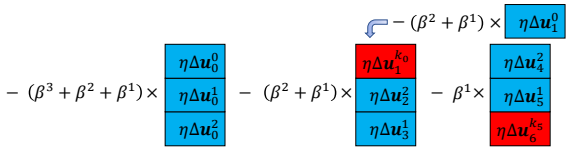
$\Delta\mathbf{u}_{ite(t,k_t)}^{k_t}$ and local parameter update $\Delta\mathbf{w}_{ite(t,k_t)}^{k_t}$. We explain the update rules on the server at iteration t.

- $\mathbf{w}_{t+\frac{1}{2}} = \mathbf{w}_t - \beta\mathbf{u}_t$, $\mathbf{u}_{t+\frac{1}{2}} = \beta\mathbf{u}_t$, when $\lceil \frac{t}{K} \rceil > \lceil \frac{t-1}{K} \rceil$.
  Since the global parameter $\mathbf{w}_t$ has been sent to worker k_{t-1}, $\Delta\mathbf{u}_t^{k_{t-1}}$ may arrive at the server at iteration t. If $\lceil \frac{t}{K} \rceil > \lceil \frac{t-1}{K} \rceil$, the global momentum is multiplied by β and used to update the global parameter. In this way, the global momentum gets ready to accommodate $\Delta\mathbf{u}_t^{k_{t-1}}$, which belongs to the new ($\lceil \frac{t}{K} \rceil$-th) group.
- Update the global momentum: $\mathbf{u}_{t+1} = \mathbf{u}_{t+\frac{1}{2}} + \beta^{\lceil \frac{t}{K} \rceil - \lceil \frac{ite(t,k_t)}{K} \rceil}\eta_t\Delta\mathbf{u}_{ite(t,k_t)}^{k_t}$.
  Since the scaled local momentum from the $\lceil \frac{t}{K} \rceil$-th group weights β^0 in the global momentum, the weight of the scaled local momentum $\eta_t\Delta\mathbf{u}_{ite(t,k_t)}^{k_t}$ (belonging to the $\lceil \frac{ite(t,k_t)}{K} \rceil$-th group) should be $\beta^{\lceil \frac{t}{K} \rceil - \lceil \frac{ite(t,k_t)}{K} \rceil}$. OrLoMo updates the global momentum by adding $\eta_t\Delta\mathbf{u}_{ite(t,k_t)}^{k_t}$ with this weight.
- Update the global parameter: $\mathbf{w}_{t+1} = \mathbf{w}_{t+\frac{1}{2}} - \eta_t\Delta\mathbf{w}_{ite(t,k_t)}^{k_t} - \frac{\beta(1-\beta^{\lceil \frac{t}{K} \rceil - \lceil \frac{ite(t,k_t)}{K} \rceil})}{1-\beta}\eta_t\Delta\mathbf{u}_{ite(t,k_t)}^{k_t}$.
  We analyze the contributions of the newly arrived $\Delta\mathbf{u}_{ite(t,k_t)}^{k_t}$ and $\Delta\mathbf{w}_{ite(t,k_t)}^{k_t}$ to the global parameter update, separately.
  - $\Delta\mathbf{w}_{ite(t,k_t)}^{k_t}$ is scaled by the global learning rate η_t and directly applied to update the global parameter.
  - Local momentums from the same group as $\Delta\mathbf{u}_{ite(t,k_t)}^{k_t}$ (received by the server at previous iterations) have contributed to the global parameter update with a total coefficient $-(\beta^1 + \beta^2 + \cdots + \beta^{\lceil \frac{t}{K} \rceil - \lceil \frac{ite(t,k_t)}{K} \rceil}) = -\frac{\beta(1-\beta^{\lceil \frac{t}{K} \rceil - \lceil \frac{ite(t,k_t)}{K} \rceil})}{1-\beta}$, scaled by their corresponding global learning rates. To ensure consistency, the coefficient of $\Delta\mathbf{u}_{ite(t,k_t)}^{k_t}$ for the global parameter update is set as $-\frac{\beta(1-\beta^{\lceil \frac{t}{K} \rceil - \lceil \frac{ite(t,k_t)}{K} \rceil})}{1-\beta}$, scaled by its corresponding global learning rate η_t.

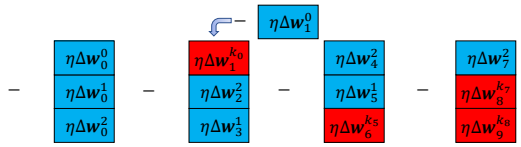
We use an example shown in Figure 1 to illustrate the update process of OrLoMo at iteration 8 when $K = 3$. For simplicity, we consider the global learning rate as constant here, i.e., $\eta_t \equiv \eta$. The local momentums or local parameter updates shown in red indicate those that have not arrived at the server. In this case, $k_0 = 0, k_2 = k_4 = k_5 = 1$ and $k_1 = k_3 = k_6 = k_7 = 2$. The global iteration indexes of the server's received information at global iterations $0, 1, \cdots, 7$ are 0, 0, 0, 2, 3, 5, 4, 7. At iteration 8, the local momentum $\Delta\mathbf{u}_1^0$ and the local parameter update $\Delta\mathbf{w}_1^0$ from worker 0 arrive at the server, i.e., $k_8 = 0$ and $ite(8, k_8) = 1$. The scaled local momentum $\eta\Delta\mathbf{u}_1^0$ is added into the global momentum with the weight of $\beta^{\lceil \frac{8}{3} \rceil - \lceil \frac{1}{3} \rceil} = \beta^2$ as shown in Figure 1(a). Then we get that $\mathbf{u}_9 = \beta^3 \times (\eta\Delta\mathbf{u}_0^0 + \eta\Delta\mathbf{u}_0^1 + \eta\Delta\mathbf{u}_0^2) + \beta^2 \times (\eta\Delta\mathbf{u}_1^0 + \eta\Delta\mathbf{u}_2^2 + \eta\Delta\mathbf{u}_3^1) + \beta^1 \times (\eta\Delta\mathbf{u}_4^2 + \eta\Delta\mathbf{u}_5^1) + \beta^0 \times \eta\Delta\mathbf{u}_7^2$. The local momentum $\Delta\mathbf{u}_1^0$ belongs to the same group as $\Delta\mathbf{u}_2^2$, which was received by the server at iteration 3. $\eta\Delta\mathbf{u}_2^2$ has contributed to the parameter update with the coefficient $-(\beta^2 + \beta^1)$. Thus, we set the coefficient of $\eta\Delta\mathbf{u}_1^0$ as $-(\beta^2 + \beta^1)$ when updating the global parameter as shown in Figure 1(b). The local parameter update $\Delta\mathbf{w}_1^0$ is scaled by the learning rate η and then directly used to update the global parameter as shown in Figure 1(c).



(a) The global momentum $\mathbf{u}_{t+1}$.



(b) $\mathbf{w}_{t+1} - \mathbf{w}_0$ contributed by the local momentums.



(c) $\mathbf{w}_{t+1} - \mathbf{w}_0$ contributed by the local parameter updates.

Figure 1: Update demonstration when $t = 8, K = 3$.

Remark 1. *When $\beta = 0$, OrLoMo degenerates to AL-SGD in Algorithm 1. Compared with AL-SGD, OrLoMo requires workers to send not only local parameter updates but also local momentums to the server. This inclusion of local momentums is typical in synchronous local momentum methods, e.g., (Yu, Jin, and Yang 2019). Since OrLoMo operates with low communication frequency, the extra overhead doesn't significantly impact the training speed, which is validated in our experiments.*

Remark 2. *When $S = 1$, OrLoMo degenerates to OrMo-DA in (Shi, Yang, and Li 2024). In OrLoMo, each worker runs S MSGD steps locally. For comparison, we implement a baseline method named local OrMo-DA in the experiments, where workers execute S SGD steps locally instead. Experimental results confirm that local momentum in OrLoMo significantly improves convergence performance, especially when the number of local iterations S is large.*

Convergence Analysis

In this subsection, we prove the convergence of OrLoMo for non-convex problems under arbitrary delays. The proof

details are included in the appendix.

We adopt the following standard assumptions in distributed learning. Notably, our convergence analysis doesn't require any assumptions about the delays, e.g., the maximum delay $\tau_t = t - ite(t,k_t) \leq \tau_{max}, \forall t \in [T]$, making our results valid for arbitrary delays. Furthermore, unlike existing ADL methods (Yang and Li 2021; Wang et al. 2024), we do not require the bounded gradient assumption, i.e., $\mathbb{E}_{\xi\sim\mathcal{D}}\|\nabla f(\mathbf{w};\xi)\|^2 \leq G^2, \forall \mathbf{w} \in \mathbb{R}^d$.

Assumption 1. *The stochastic gradient $\nabla f(\mathbf{w};\xi)$ has unbiased expectation and bounded variance for $\forall \mathbf{w} \in \mathbb{R}^d$:*

$$\mathbb{E}_{\xi\sim\mathcal{D}}\left[\nabla f(\mathbf{w};\xi)\right] = \nabla F(\mathbf{w}),$$
$$\mathbb{E}_{\xi\sim\mathcal{D}}\|\nabla f(\mathbf{w};\xi) - \nabla F(\mathbf{w})\|^2 \leq \sigma^2.$$

Assumption 1 indicates that the data across all workers are independent and identically distributed (i.i.d.). It's commonly satisfied in the data-center scenario (Dean et al. 2012), where all workers have access to the full dataset.

Assumption 2. *$F(\mathbf{w})$ is L-smooth ($L > 0$):*

$$F(\mathbf{w}) \leq F(\mathbf{w}') + \nabla F(\mathbf{w}')^T(\mathbf{w} - \mathbf{w}') + \frac{L}{2}\|\mathbf{w} - \mathbf{w}'\|^2,$$

holds for $\forall \mathbf{w}, \mathbf{w}' \in \mathbb{R}^d$.

Assumption 3. $F(\mathbf{w}) \geq F^*, \forall \mathbf{w} \in \mathbb{R}^d$.

Firstly, we define two notations $next(t,k)$ and $\hat{\eta}_{t,k}$. $next(t,k)$ denotes the index of the next global iteration at which the server will receive the information from worker k after global iteration t (including t). $\hat{\eta}_{t,k}$ is the global learning rate at iteration $next(t,k)$. They are formulated as:

$$next(t,k) = \min\{j \geq t : k_j = k\},$$
$$\hat{\eta}_{t,k} = \eta_{next(t,k)},$$

where $k \in [K]$, $t \in [T]$.

Secondly, we introduce two auxiliary variables $\hat{\mathbf{u}}_t$ and $\hat{\mathbf{w}}_t$ corresponding to $\mathbf{u}_t$ and $\mathbf{w}_t$ respectively, where $t \geq 1$:

$$\hat{\mathbf{u}}_{t+1} = \begin{cases} \beta\hat{\mathbf{u}}_t + \hat{\eta}_{t,k_{t-1}}\Delta\mathbf{u}_t^{k_{t-1}} & K \mid (t-1), \\ \hat{\mathbf{u}}_t + \hat{\eta}_{t,k_{t-1}}\Delta\mathbf{u}_t^{k_{t-1}} & K \nmid (t-1), \end{cases}$$

for $t \geq 1$ and $\hat{\mathbf{u}}_1 = \sum_{k\in[K]}\hat{\eta}_{0,k}\Delta\mathbf{u}_0^k$,

$$\hat{\mathbf{w}}_{t+1} = \begin{cases} \hat{\mathbf{w}}_t - \beta\hat{\mathbf{u}}_t - \hat{\eta}_{t,k_{t-1}}\Delta\mathbf{w}_t^{k_{t-1}} & K \mid (t-1), \\ \hat{\mathbf{w}}_t - \hat{\eta}_{t,k_{t-1}}\Delta\mathbf{w}_t^{k_{t-1}} & K \nmid (t-1), \end{cases}$$

for $t \geq 1$ and $\hat{\mathbf{w}}_1 = \mathbf{w}_0 - \sum_{k\in[K]}\hat{\eta}_{0,k}\Delta\mathbf{w}_0^k$.

For simplicity, we define $\Delta\mathbf{h}_0^k = \gamma\sum_{s=0}^{S-1}\mathbf{g}_{0,s}^k$ for $k \in [K]$, and $\Delta\mathbf{h}_t^{k_{t-1}} = \gamma\sum_{s=0}^{S-1}\mathbf{g}_{t,s}^{k_{t-1}}$ for $t \geq 1$. It's easy to verify that: $\frac{1}{1-\beta}\Delta\mathbf{h}_0^k = \frac{\beta}{1-\beta}\Delta\mathbf{u}_0^k + \Delta\mathbf{w}_0^k$ for $k \in [K]$, and $\frac{1}{1-\beta}\Delta\mathbf{h}_t^{k_{t-1}} = \frac{\beta}{1-\beta}\Delta\mathbf{u}_t^{k_{t-1}} + \Delta\mathbf{w}_t^{k_{t-1}}$ for $t \geq 1$.

Then, we define another auxiliary variable $\hat{\mathbf{y}}_t$ corresponding to $\hat{\mathbf{w}}_t$: $\hat{\mathbf{y}}_1 = \mathbf{w}_0 - \frac{1}{1-\beta}\sum_{k\in[K]}\hat{\eta}_{0,k}\Delta\mathbf{h}_0^k$, and

$$\hat{\mathbf{y}}_{t+1} = \hat{\mathbf{y}}_t - \frac{\hat{\eta}_{t,k_{t-1}}}{1-\beta}\Delta\mathbf{h}_t^{k_{t-1}}, t \geq 1. \tag{3}$$

Our proof begins with rigorously tracking the differences between these variables in Lemmas 1, 2, and 3.

Lemma 1. *For any $t \geq 0$, the difference between $\mathbf{u}_{t+1}$ and $\hat{\mathbf{u}}_{t+1}$ can be expressed as follows:*

$$\hat{\mathbf{u}}_{t+1} - \mathbf{u}_{t+1} = \sum_{k\in[K],k\neq k_t}\beta^{\lceil\frac{t}{K}\rceil - \lceil\frac{ite(t,k)}{K}\rceil}\hat{\eta}_{ite(t,k),k}\Delta\mathbf{u}_{ite(t,k)}^k. \tag{4}$$

Lemma 2. *For any $t \geq 0$, the difference between $\mathbf{w}_{t+1}$ and $\hat{\mathbf{w}}_{t+1}$ can be expressed as follows:*

$$\hat{\mathbf{w}}_{t+1} - \mathbf{w}_{t+1} = -\sum_{k\in[K],k\neq k_t}\Bigg[\hat{\eta}_{ite(t,k),k}\Delta\mathbf{w}_{ite(t,k)}^k + \frac{\beta(1-\beta^{\lceil\frac{t}{K}\rceil - \lceil\frac{ite(t,k)}{K}\rceil})}{1-\beta}\hat{\eta}_{ite(t,k),k}\Delta\mathbf{u}_{ite(t,k)}^k\Bigg]. \tag{5}$$

Lemma 3. *For any $t \geq 1$, the difference between $\hat{\mathbf{y}}_t$ and $\hat{\mathbf{w}}_t$ can be expressed as: $\hat{\mathbf{y}}_t - \hat{\mathbf{w}}_t = -\frac{\beta}{1-\beta}\hat{\mathbf{u}}_t$.*

Lemma 4. *With Assumption 1, the gap between the local and global parameters can be bounded for $t \geq 1$:*

$$\sum_{s=0}^{S-1}\mathbb{E}\left\|\tilde{\mathbf{w}}_{t,s}^{k_{t-1}} - \mathbf{w}_t\right\|^2 \leq \frac{\gamma^2 S(S-1)\sigma^2}{(1-\beta)^2} + \frac{\gamma^2 S(S-1)}{(1-\beta)^2}\sum_{s=0}^{S-1}\mathbb{E}\left\|\nabla F(\tilde{\mathbf{w}}_{t,s}^{k_{t-1}})\right\|^2.$$

Theorem 1. *With Assumptions 1, 2 and 3, letting $16LS\gamma \leq (1-\beta)^2$ and*

$$\eta_t = \begin{cases} \frac{1}{K} & \tau_t \leq 2K, \\ \frac{1}{\tau_t} & \tau_t > 2K, \end{cases} \tag{6}$$

OrLoMo in Algorithm 2 has the following convergence rate:

$$\mathbb{E}\left\|\nabla F(\bar{\mathbf{w}}_T)\right\|^2 \leq \frac{4K(1-\beta)\left(F(\mathbf{w}_0) - F^*\right)}{\gamma ST} + \frac{4\gamma L\sigma^2}{K(1-\beta)^2} + \frac{\gamma^2 L^2(S-1)\sigma^2}{(1-\beta)^2},$$

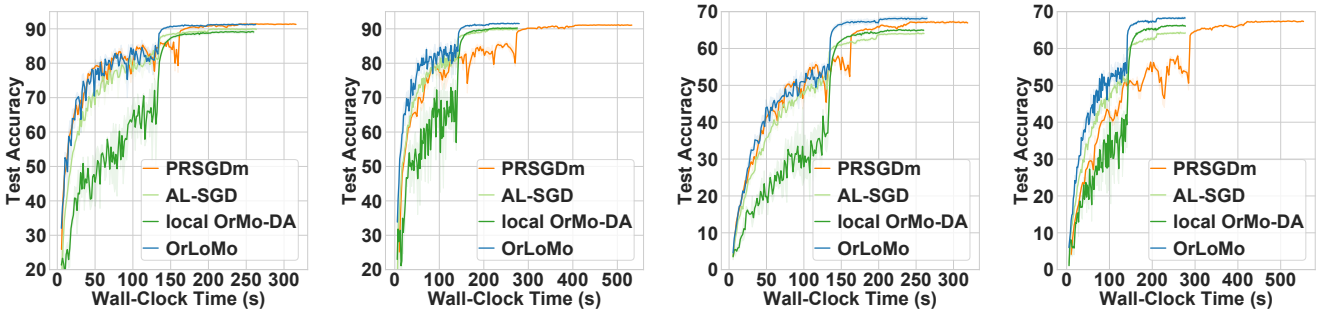
where $\mathbb{E}\left\|\nabla F(\bar{\mathbf{w}}_T)\right\|^2 = \frac{\sum_{k\in[K]}\hat{\eta}_{0,k}\|\nabla F(\mathbf{w}_0)\|^2}{\sum_{k\in[K]}\hat{\eta}_{0,k} + \sum_{t=1}^{T-1}\hat{\eta}_{t,k_{t-1}}} + \frac{\sum_{t=1}^{T-1}\hat{\eta}_{t,k_{t-1}}\mathbb{E}\|\nabla F(\mathbf{w}_t)\|^2}{\sum_{k\in[K]}\hat{\eta}_{0,k} + \sum_{t=1}^{T-1}\hat{\eta}_{t,k_{t-1}}}$.

Remark 3. *The global learning rate in (6) implements a mechanism that penalizes both the local momentums and the local parameter updates with extremely large delays ($\tau_t > 2K$). This mechanism is critical for ensuring convergence of asynchronous methods under arbitrary delays, as shown in (Mishchenko et al. 2022; Shi, Yang, and Li 2024).*

Remark 4. *Theorem 1 in (Yu, Jin, and Yang 2019) establishes the convergence rate of PRSGDm as $\mathcal{O}(\frac{1}{\gamma\tilde{T}} + \frac{\gamma}{K} + \gamma^2 S)$, with $\tilde{T}$ denoting its number of iterations. For fair comparison between OrLoMo and PRSGDm, we unify the metric to total gradient computations C. PRSGDm's convergence rate is $\mathcal{O}(\frac{K}{\gamma C} + \frac{\gamma}{K} + \gamma^2 S)$ since $C = K\tilde{T}$. OrLoMo's convergence rate is $\mathcal{O}(\frac{K}{\gamma C} + \frac{\gamma}{K} + \gamma^2 S)$ since $C = ST$. Our analysis proves identical gradient computation complexity between OrLoMo and its synchronous counterpart PRSGDm.*

	Datasets		CIFAR10				CIFAR100			
	Workers	Local Iterations	PRSGDm	AL-SGD	local OrMo-DA	OrLoMo	PRSGDm	AL-SGD	local OrMo-DA	OrLoMo
homogeneous	8	8	**92.18±0.10**	91.46±0.33	92.02±0.23	92.11±0.08	**69.89±0.08**	67.46±0.48	69.04±0.41	69.43±0.23
		16	91.67±0.14	91.01±0.09	91.36±0.20	**92.11±0.26**	69.09±0.31	67.05±0.32	68.29±0.46	**69.49±0.23**
		32	91.37±0.19	90.82±0.16	90.02±0.09	**91.85±0.12**	68.29±0.24	67.17±0.33	65.15±0.20	**68.95±0.12**
	16	8	**92.00±0.12**	90.26±0.16	91.14±0.07	91.39±0.09	**69.03±0.12**	64.67±0.36	67.30±0.48	68.92±0.36
		16	**91.29±0.13**	89.85±0.28	89.08±0.43	91.27±0.18	66.93±0.18	64.06±0.25	64.94±0.26	**68.20±0.60**
		32	90.10±0.02	89.21±0.20	77.96±0.87	**91.05±0.09**	64.78±0.37	62.15±0.71	49.09±0.60	**65.65±0.25**
heterogeneous	8	8	92.06±0.13	91.13±0.19	92.18±0.13	**92.34±0.10**	69.60±0.23	67.97±0.40	**69.73±0.29**	69.44±0.30
		16	91.66±0.13	91.15±0.14	91.64±0.07	**91.76±0.24**	68.57±0.16	67.38±0.23	68.62±0.31	**69.25±0.21**
		32	91.61±0.35	90.88±0.23	90.83±0.11	**91.96±0.12**	68.40±0.30	67.06±0.43	67.09±0.44	**69.25±0.19**
	16	8	**91.88±0.23**	90.02±0.03	91.45±0.14	91.61±0.20	**69.29±0.18**	64.83±0.30	67.79±0.21	68.65±0.34
		16	91.08±0.05	89.85±0.13	90.23±0.14	**91.50±0.02**	67.34±0.18	64.20±0.27	66.07±0.14	**68.26±0.27**
		32	90.41±0.19	89.31±0.34	83.45±1.14	**90.73±0.29**	64.92±0.40	61.99±0.24	55.42±0.51	**66.19±0.24**

Table 1: Test accuracy results of SqueezeNet model.



(a) homogeneous (CIFAR10) (b) heterogeneous (CIFAR10) (c) homogeneous (CIFAR100) (d) heterogeneous (CIFAR100)

Figure 2: Test accuracy curves of SqueezeNet model when $K = 16, S = 16$.

Remark 5. *If the local learning rate* γ *is set as* $\min\{\frac{(1-\beta)^2}{16LS}, \frac{K(1-\beta)^{\frac{3}{2}}(F(\mathbf{w}_0)-F^*)^{\frac{1}{2}}}{(LST)^{\frac{1}{2}}\sigma}, \frac{(1-\beta)[4K(F(\mathbf{w}_0)-F^*)]^{\frac{1}{3}}}{[S(S-1)T]^{\frac{1}{3}}(L\sigma)^{\frac{2}{3}}}\}$, *the convergence rate in Theorem 1 becomes* $\mathcal{O}\left(\sqrt{\frac{L\sigma^2}{ST}} + \frac{(KL\sigma)^{\frac{2}{3}}(S-1)^{\frac{1}{3}}}{(ST)^{\frac{2}{3}}} + \frac{KL}{T}\right)$. *By letting* $S = 1$ *and* $\gamma = \min\{\frac{(1-\beta)^2}{16L}, \frac{K(1-\beta)^{\frac{3}{2}}(F(\mathbf{w}_0)-F^*)^{\frac{1}{2}}}{(LT)^{\frac{1}{2}}\sigma}\}$ *in Theorem 1, we get the convergence rate* $\mathcal{O}\left(\sqrt{\frac{L\sigma^2}{T}} + \frac{KL}{T}\right)$ *for OrLoMo, which matches the convergence results for OrMo-DA in Theorem 2 in (Shi, Yang, and Li 2024).*

Experiments

In this section, we evaluate the performance of OrLoMo and other baseline methods. All methods are implemented based on the PS framework. The experiments are conducted using NVIDIA RTX 2080 Ti GPUs. The distributed platform is implemented using Docker containerization.

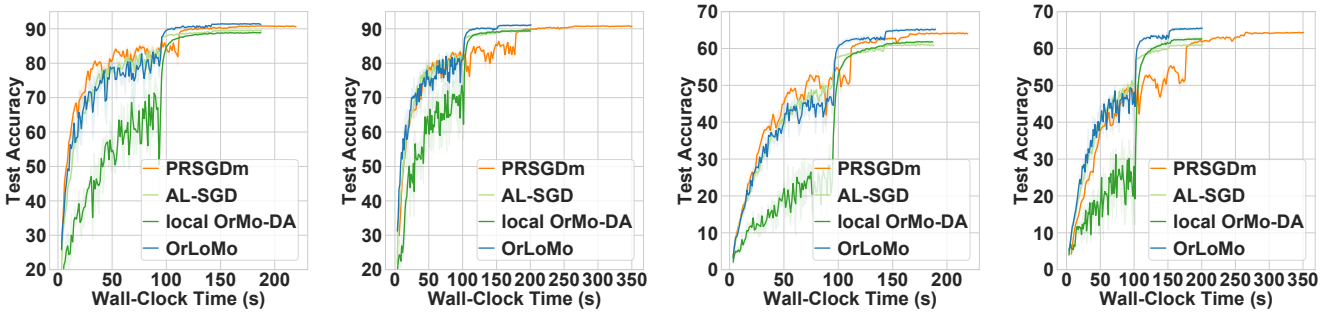
The baseline methods include PRSGDm (Yu, Jin, and Yang 2019), AL-SGD, and local OrMo-DA. In local OrMo-DA, each worker performs SGD steps locally, while the server follows the same update rule as OrMo-DA (Shi, Yang, and Li 2024), as detailed in the appendix. AL-SGD, local OrMo-DA, and OrLoMo are ADL methods, while PRSGDm is an SDL method. We evaluate these methods by training SqueezeNet (Han et al. 2015) and ResNet20 (He et al. 2016) models on CIFAR10 and CIFAR100 datasets (Krizhevsky, Hinton et al. 2009). Additional experimental results of ResNet32 on Tiny-ImageNet are provided in the appendix. The models are trained with 160 epochs. The number of workers K is set to 8 and 16. The batch size for each local iteration is set to 64. The number of local iterations S is set to 8, 16, and 32. The local learning rate is multiplied by 0.1 at the 80-th and 120-th epochs. The momentum coefficient is set to 0.9. The weight decay is set to 0.001. Each experiment is repeated 3 times. Two scenarios are considered in our experiments:

- Homogeneous: All the workers have similar computational capabilities.
- Heterogeneous: 25% of all workers are designated as slow workers, with their average gradient computation time being twice as long as that of the other workers.

Table 1 and Table 2 show the test accuracy results. Figure 2 and Figure 3 present the test accuracy curves with respect to wall-clock time. Experimental results on training

	Datasets		CIFAR10				CIFAR100			
	Workers	Local Iterations	PRSGDm	AL-SGD	local OrMo-DA	OrLoMo	PRSGDm	AL-SGD	local OrMo-DA	OrLoMo
homogeneous	8	8	91.65±0.11	90.99±0.11	**91.89±0.05**	91.76±0.19	**67.78±0.08**	65.20±0.22	67.27±0.12	67.58±0.21
		16	**91.68±0.17**	90.72±0.18	90.90±0.11	91.49±0.14	**66.73±0.32**	64.82±0.21	66.30±0.09	66.32±0.23
		32	91.14±0.15	90.69±0.06	88.64±0.26	**91.26±0.13**	65.43±0.06	64.74±0.34	63.45±0.18	**66.08±0.35**
	16	8	91.36±0.17	89.58±0.10	90.71±0.23	**91.37±0.13**	65.80±0.12	62.26±0.47	66.00±0.16	**66.18±0.10**
		16	90.64±0.17	89.43±0.18	88.69±0.02	**91.32±0.14**	64.00±0.38	60.80±0.24	61.78±0.15	**65.23±0.43**
		32	89.72±0.12	88.81±0.17	76.42±3.15	**90.13±0.08**	61.91±0.24	59.65±0.44	48.48±0.65	**62.07±0.19**
heterogeneous	8	8	91.90±0.09	91.08±0.28	91.88±0.05	**91.97±0.31**	67.23±0.24	65.02±0.04	**67.83±0.12**	67.62±0.42
		16	91.61±0.13	90.84±0.06	91.33±0.11	**91.76±0.15**	66.45±0.33	64.71±0.44	66.40±0.16	**66.48±0.14**
		32	90.99±0.02	90.94±0.24	89.61±0.38	**91.44±0.16**	65.43±0.06	64.74±0.34	63.45±0.18	**66.08±0.35**
	16	8	**91.28±0.33**	89.58±0.13	91.11±0.10	91.27±0.14	65.80±0.12	62.26±0.47	66.00±0.16	**66.18±0.10**
		16	90.72±0.06	89.28±0.09	89.44±0.30	**91.12±0.15**	64.32±0.08	61.02±0.15	62.56±0.64	**65.44±0.30**
		32	89.76±0.20	89.20±0.48	82.26±0.95	**90.15±0.23**	61.73±0.23	59.60±0.24	51.27±0.78	**61.88±0.07**

Table 2: Test accuracy results of ResNet20 model.



(a) homogeneous (CIFAR10) (b) heterogeneous (CIFAR10) (c) homogeneous (CIFAR100) (d) heterogeneous (CIFAR100)

Figure 3: Test accuracy curves of ResNet20 model when $K = 16, S = 16$.

Slow/Fast	1	2	5	10	50
Maximum Delay	34	207	714	1905	9449
Test Accuracy	91.38	91.42	91.23	91.62	91.44

Table 3: Test accuracy results of OrLoMo under different heterogeneous scenarios on ResNet20 model and CIFAR10 dataset with $K = 16, S = 8$.

loss are presented in the appendix. Compared with AL-SGD and local OrMo-DA, OrLoMo can achieve better test accuracy. OrLoMo's communication overhead introduced by local momentums incurs negligible training time impact, maintaining comparable efficiency to both AL-SGD and local OrMo-DA. As shown in Figure 2 and Figure 3, OrLoMo can be 2 times faster than PRSGDm in the heterogeneous scenario since the training speed of PRSGDm is hindered by slow workers. In contrast, slow workers have a limited impact on OrLoMo's training speed. Even in the homogeneous scenario, OrLoMo remains faster than PRSGDm. This advantage arises because the computation time of each worker varies within a certain range, and some workers must wait for the others to complete their gradient computations in PRSGDm.

To assess OrLoMo's robustness to arbitrary delays (particularly extreme maximum delays), we conduct experiments under different heterogeneous scenarios. We designate one worker as the slow worker while keeping the others as fast workers. Table 3 shows test accuracy results of OrLoMo when $K = 16$, $S = 8$. The *Slow/Fast* ratio quantifies the disparity in average gradient computation time between the slow worker and the fast workers, with higher values indicating more severe maximum delays. Our experimental results confirm that OrLoMo sustains robust performance under extremely large delays. This aligns with our theoretical analysis, which remains valid under arbitrary delays.

Conclusion

In this paper, we propose a novel method, called OrLoMo, for asynchronous distributed learning. To the best of our knowledge, OrLoMo is the first method to implement asynchronous distributed MSGD with local updates. We prove the convergence of OrLoMo for non-convex problems under arbitrary delays. Empirical results demonstrate that OrLoMo can achieve state-of-the-art performance.

Acknowledgments

This work is supported by National Key R&D Program of China (No.2020YFA0713900), the Key Major Project of Pengcheng Laboratory (No.PCL2024A06) and NSFC Project (No.12326615).

References

Agarwal, A.; and Duchi, J. C. 2011. Distributed Delayed Stochastic Optimization. In *Advances in Neural Information Processing Systems*, 873–881.

Arjevani, Y.; Shamir, O.; and Srebro, N. 2020. A Tight Convergence Analysis for Stochastic Gradient Descent with Delayed Updates. In *Proceedings of the International Conference on Algorithmic Learning Theory*, 111–132.

Basu, D.; Data, D.; Karakus, C.; and Diggavi, S. N. 2019. Qsparse-local-SGD: Distributed SGD with Quantization, Sparsification and Local Computations. In *Advances in Neural Information Processing Systems*, 14668–14679.

Crawshaw, M.; and Liu, M. 2024. Federated Learning under Periodic Client Participation and Heterogeneous Data: A New Communication-Efficient Algorithm and Analysis. In *Advances in Neural Information Processing Systems*, 8240–8299.

Dean, J.; Corrado, G.; Monga, R.; Chen, K.; Devin, M.; Le, Q. V.; Mao, M. Z.; Ranzato, M.; Senior, A. W.; Tucker, P. A.; Yang, K.; and Ng, A. Y. 2012. Large Scale Distributed Deep Networks. In *Advances in Neural Information Processing Systems*, 1232–1240.

Haddadpour, F.; Kamani, M. M.; Mahdavi, M.; and Cadambe, V. R. 2019. Local SGD with Periodic Averaging: Tighter Analysis and Adaptive Synchronization. In *Advances in Neural Information Processing Systems*, 11080–11092.

Han, S.; Pool, J.; Tran, J.; and Dally, W. 2015. Learning both Weights and Connections for Efficient Neural Network. In *Advances in Neural Information Processing Systems*, 1135–1143.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 770–778.

Karimireddy, S. P.; Kale, S.; Mohri, M.; Reddi, S. J.; Stich, S. U.; and Suresh, A. T. 2020. SCAFFOLD: Stochastic Controlled Averaging for Federated Learning. In *Proceedings of the International Conference on Machine Learning*, 5132–5143.

Kingma, D. P.; and Ba, J. 2015. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations*.

Koloskova, A.; Stich, S. U.; and Jaggi, M. 2022. Sharper Convergence Guarantees for Asynchronous SGD for Distributed and Federated Learning. In *Advances in Neural Information Processing Systems*, 17202–17215.

Krizhevsky, A.; Hinton, G.; et al. 2009. Learning Multiple Layers of Features from Tiny Images. *Technical report*.

Li, M.; Andersen, D. G.; Smola, A. J.; and Yu, K. 2014. Communication Efficient Distributed Machine Learning with the Parameter Server. In *Advances in Neural Information Processing Systems*, 19–27.

Lian, X.; Huang, Y.; Li, Y.; and Liu, J. 2015. Asynchronous Parallel Stochastic Gradient for Nonconvex Optimization. In *Advances in Neural Information Processing Systems*, 2737–2745.

Lian, X.; Zhang, W.; Zhang, C.; and Liu, J. 2018. Asynchronous Decentralized Parallel Stochastic Gradient Descent. In *Proceedings of the International Conference on Machine Learning*, 3043–3052.

Lin, T.; Stich, S. U.; Patel, K. K.; and Jaggi, M. 2019. Don't Use Large Mini-Batches, Use Local SGD. In *International Conference on Learning Representations*.

Lin, Y.; Han, S.; Mao, H.; Wang, Y.; and Dally, B. 2018. Deep Gradient Compression: Reducing the Communication Bandwidth for Distributed Training. In *International Conference on Learning Representations*.

Maranjyan, A.; Tyurin, A.; and Richtárik, P. 2025. Ringmaster ASGD: The First Asynchronous SGD with Optimal Time Complexity. In *Proceedings of the International Conference on Machine Learning*, 43120–43139.

McMahan, B.; Moore, E.; Ramage, D.; Hampson, S.; and y Arcas, B. A. 2017. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, 1273–1282.

Mishchenko, K.; Bach, F. R.; Even, M.; and Woodworth, B. E. 2022. Asynchronous SGD Beats Minibatch SGD Under Arbitrary Delays. In *Advances in Neural Information Processing Systems*, 420–433.

Nguyen, J.; Malik, K.; Zhan, H.; Yousefpour, A.; Rabbat, M.; Malek, M.; and Huba, D. 2022. Federated Learning with Buffered Asynchronous Aggregation. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, 3581–3607.

Polyak, B. 1964. Some Methods of Speeding Up the Convergence of Iteration Methods. *Ussr Computational Mathematics and Mathematical Physics*, 4: 1–17.

Reddi, S. J.; Kale, S.; and Kumar, S. 2018. On the Convergence of Adam and Beyond. In *International Conference on Learning Representations*.

Robbins, H.; and Monro, S. 1951. A Stochastic Approximation Method. *Annals of Mathematical Statistics*, 22: 400–407.

Shi, C.-W.; Yang, Y.-R.; and Li, W.-J. 2024. Ordered Momentum for Asynchronous SGD. In *Advances in Neural Information Processing Systems*, 16943–16982.

Stich, S. U. 2019. Local SGD Converges Fast and Communicates Little. In *International Conference on Learning Representations*.

Verbraeken, J.; Wolting, M.; Katzy, J.; Kloppenburg, J.; Verbelen, T.; and Rellermeyer, J. S. 2020. A Survey on Distributed Machine Learning. *ACM Computing Surveys*, 53(2): 1–33.

Vogels, T.; Karimireddy, S. P.; and Jaggi, M. 2019. PowerSGD: Practical Low-Rank Gradient Compression for Distributed Optimization. In *Advances in Neural Information Processing Systems*, 14236–14245.

Wang, J.; Tantia, V.; Ballas, N.; and Rabbat, M. G. 2020. SlowMo: Improving Communication-Efficient Distributed SGD with Slow Momentum. In *International Conference on Learning Representations*.

Wang, X.; Li, Z.; Jin, S.; and Zhang, J. 2025. Achieving Linear Speedup in Asynchronous Federated Learning With Heterogeneous Clients. *IEEE Transactions on Mobile Computing*, 24(1): 435–448.

Wang, Y.; Wang, S.; Lu, S.; and Chen, J. 2024. FADAS: Towards Federated Adaptive Asynchronous Optimization. In *Proceedings of the International Conference on Machine Learning*, 51701–51733.

Xie, C.; Koyejo, S.; and Gupta, I. 2019. Asynchronous Federated Optimization. *arXiv preprint arXiv:1903.03934*.

Yang, T.; Yi, X.; Wu, J.; Yuan, Y.; Wu, D.; Meng, Z.; Hong, Y.; Wang, H.; Lin, Z.; and Johansson, K. H. 2019. A Survey of Distributed Optimization. *Annual Reviews in Control*, 47: 278–305.

Yang, Y.; and Li, W. 2021. BASGD: Buffered Asynchronous SGD for Byzantine Learning. In *Proceedings of the International Conference on Machine Learning*, 11751–11761.

Yu, H.; Jin, R.; and Yang, S. 2019. On the Linear Speedup Analysis of Communication Efficient Momentum SGD for Distributed Non-Convex Optimization. In *Proceedings of the International Conference on Machine Learning*, 7184–7193.

Yu, H.; Yang, S.; and Zhu, S. 2019. Parallel Restarted SGD with Faster Convergence and Less Communication: Demystifying Why Model Averaging Works for Deep Learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 5693–5700.

Zheng, S.; Meng, Q.; Wang, T.; Chen, W.; Yu, N.; Ma, Z.; and Liu, T. 2017. Asynchronous Stochastic Gradient Descent with Delay Compensation. In *Proceedings of the International Conference on Machine Learning*, 4120–4129.