Distributed Training Across the World

Anonymous Authors[1]

Abstract

Conventional distributed training is usually performed inside a data center. But in many applications, training data is distributed across many geographical locations (e.g., federated learning, collaborative learning). The long physical distance leads to unavoidable high latency and traditional distributed algorithms cannot scale well under poor networking conditions. In this work, we aim to **scale distributed learning across the world**. We propose Delayed Synchronous SGD (DS-SGD) that delays the synchronization barrier to a later iteration. We compensate the gradients to overcome the staleness issue. Such a delayed update enables synchronous training to tolerate extreme network conditions without compromising accuracy. We *theoretically* verify that the convergence rate of our algorithm is no slower than vanilla SGD. We also *empirically* evaluate that there is no accuracy drop on vision (ImageNet) and NLP (Penn Treebank) tasks. Moreover, we deploy servers across three continents in the world: London (Europe), Tokyo (Asia), Oregon (North America) and Ohio (North America). Under such challenging settings, our algorithm achieves $90\times$ speedup over traditional methods without loss of accuracy.

1. Introduction

Deep neural networks have demonstrated great success in solving large-scale machine learning problems (Krizhevsky et al., 2012; Simonyan & Zisserman, 2015; He et al., 2016). With the employment of larger and larger architectures, the training time of deep neural networks increases significantly. In order to speed up the training process, a common technique to perform is distributed training (Dean et al., 2012; Recht et al., 2011; Li et al., 2014; Goyal et al., 2017). A network infrastructure with *low latency* and *high bandwidth* is essential for most modern distributed algorithms. Existing distributed frameworks such as MXNet (Chen et al., 2015), TensorFlow (Abadi et al., 2015) and Horovod (Sergeev & Del Balso, 2018) are all based on high-end networking infrastructures with high bandwidth (100 Gbps) and low latency (1 µs).

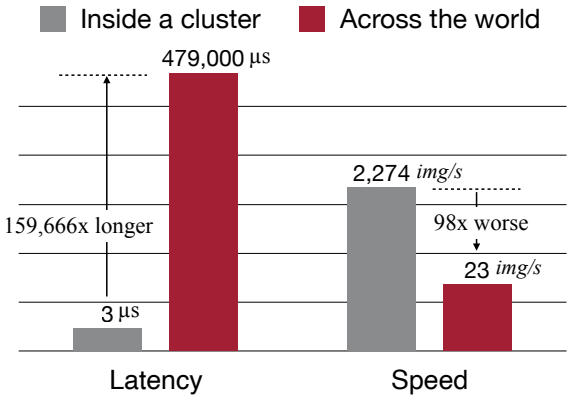


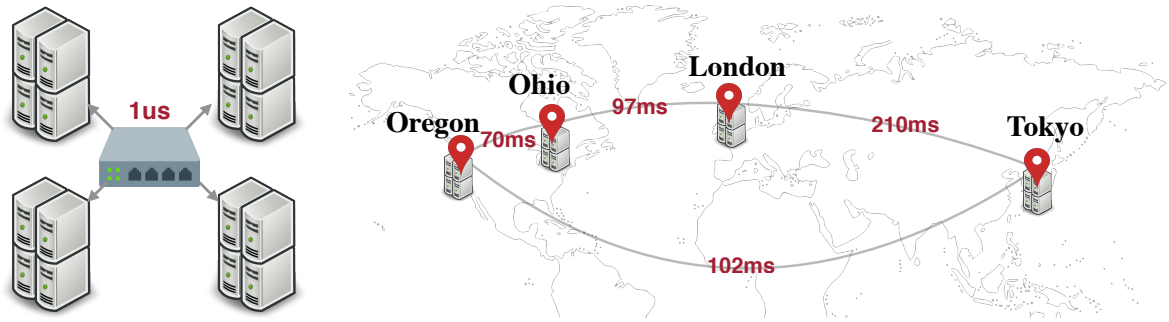
Figure 1. The networking latency across the world is five orders of magnitude longer than inside a cluster, which greatly degrades the training speed. We tackle this issue by designing new algorithms.

However, centralizing the data to a data center is not always possible. In many scenarios, such as personal medical information (Jochems et al., 2016; 2017) and keyboard input history (McMahan et al., 2017; Konečný et al., 2016), data can not be centralized within a data center because of security and privacy concerns (GDPR, 2016). This raises a new set of problems. When datasets are distributed across different geographical locations, the networking condition is much worse (high latency, low bandwidth). As shown in Figure 1, the high networking latency greatly hurts the training speed.

While as the bandwidth is easy to increase by hardware upgrade (e.g., stacking more lanes), the latency is about physical limits and hard to improve. For example, consider two servers located in Shanghai and Boston respectively, even with the speed of light (3×10^8m/s) and direct air distance ($11,725$km), it will still take 78ms to send and receive a packet. In real-world, the latency can be only worse (around 700ms) due to indirect routing between inter-

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.



(a) Distributed training inside a cluster (b) Distributed training across the world

Figure 2. Comparison between distributed training inside cluster v.s. across the world. Different from data center where high-end network infrastructure is available, long-distance distributed training suffers from inevitable high latency, which proposes a severe challenge to scale across the world.

net service providers (ISPs) and queuing delay in switches. Such high latency leads to severe training speed drop for traditional distributed training algorithms.

The focus in this work is to enable **scalable distributed training across different geographical locations** with long physical distance and high latency network connection. Our main contributions include:

- We propose an algorithm named **D**elayed **S**ynchronous **SGD** (DS-SGD) and provide extensions to allow momentum update and periodic synchronization.
- We theoretically justify the convergence of DS-SGD. We show that under reasonable delay interval, it shares the same convergence rate as Synchronous SGD.
- We empirically evaluate our algorithms on large scale settings and show that DS-SGD can be scaled across the world with servers and data distributed in four different countries. We can train ResNet-50 on ImageNet without loss of accuracy and maintain the scalability. To our best knowledge, our algorithm is the first work that can achieve scalable distributed training under such high latency setting.

2. Background and Related Work

The key of the efficiency in distributed learning systems is the communication-to-computation (C/C) ratio. The communication cost is usually modeled as

$$T_{\text{communication}} = T_{\text{latency}} + \frac{D}{B}$$

which is determined by latency T_{latency}, data size D and bandwidth B. In the standard in-cluster training, the latency is negligible, $T_{\text{latency}} << \frac{D}{B}$, hence the existing literature mainly focus on leveraging the bandwidth and data size. Fancy hardware as infinity band has been designed to increase the bandwidth and researches on gradients quantization (Seide et al., 2014) and compression (Lin et al., 2018) have been proposed to reduce the bits to be transferred.

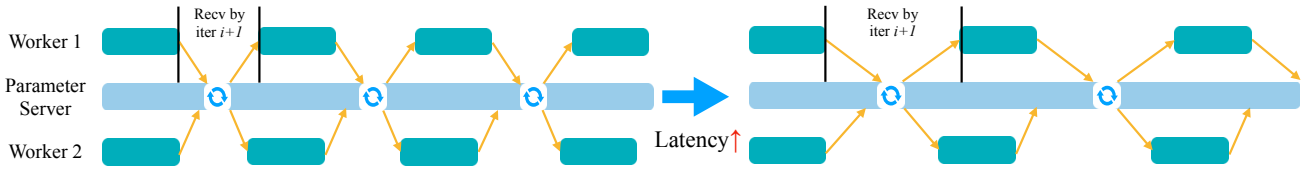
However, the latency becomes inevitably high when we would like to perform cross-region distributed training over the wide area networks (WANs). This setting has become increasingly popular recently as the growing awareness of data privacy (GDPR, 2016). For example, Federated Learning (Konečný et al., 2016) aims to jointly train a model without centralizing the data and Gaia (Hsieh et al., 2017) targets to develop an efficient geo-distributed ML system. Different from local area networks (LANs), communicating over WANs can significantly degrade the system performance because of the limited bandwidth and high latency.

While as the bandwidth can always be increased by upgrading hardware or by performing gradients compression/quantization, the latency is hard or even impossible to improve due to physical limit like the speed of light. Therefore, in this work, we focus on the **high latency issue** and assume **the bandwidth is sufficient**. Let us quickly go through the existing approaches and illustrate their limitations in the high latency setting.

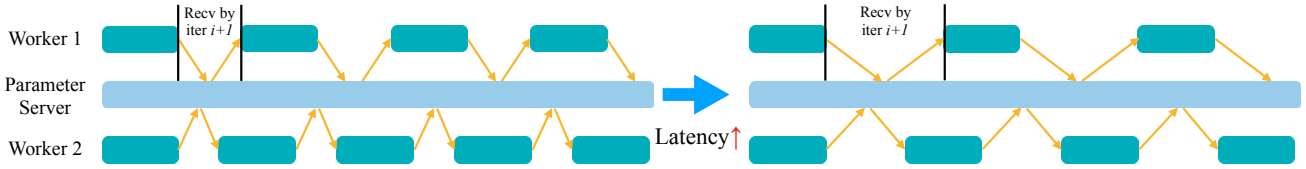
Synchronous SGD (SSGD) is the simplest solution for parallelizing SGD (Valiant, 1990; Zinkevich et al., 2010), where the gradients produced on each worker are synchronized at every iteration:

W :pull parameters $\rightarrow$ local evaluation $\rightarrow$ push gradients
PS :collect & synchronize gradients $\rightarrow$ update parameters

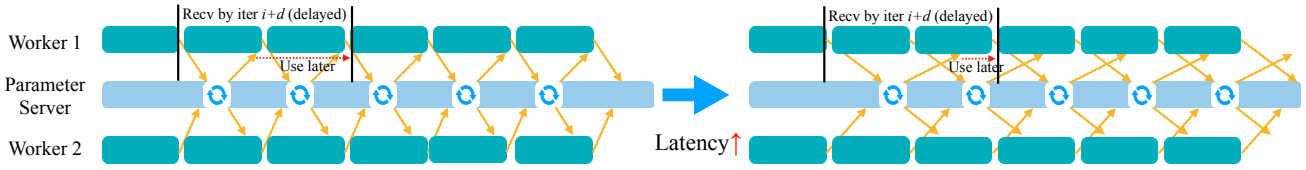
where W and PS indicate the worker and parameter server respectively. In particular, the worker needs to wait until receiving weights from the parameter server before next round's computation. Therefore it suffers a communication cost of $2T_{\text{latency}} + T_{\text{sync}}$ per iteration. The high latency will slow the training process significantly.



(a) Synchronous SGD synchronizes current gradients before next round of computation, thus not scalable when the latency increases.



(b) Asynchronous SGD has no synchronization barrier but each worker still needs to wait a long time when pulling latest weights.



(c) Delayed Synchronous SGD allows more time for communication by putting off the synchronization barrier to d steps later.

Figure 3. The visualization of popular distributed optimization algorithms (SSGD and ASGD) and our proposed DS-SGD($d = 3$). When conventional approaches slow by the high latency as the gradients required to be received by next round of computing, our DSSGD demonstrates great robustness by delaying the timing of receiving.

Asynchronous SGD (ASGD) is derived from (Tsitsiklis et al., 1986) and preferred in heterogeneous computing. In contrast to SSGD, there is no synchronization lock and the update on the parameter server is on the fly. ASGD has powered many successful applications such as HOGWILD! (Recht et al., 2011), BUCKWILD! (De Sa et al., 2015) and Dist-belief (Dean et al., 2012). The main complaint about ASGD is that training on inconsistent models deteriorates the accuracy, even theory shows that ASGD converges as good as SSGD (Lian et al., 2015).

One may expect that without the synchronization barrier, ASGD could handle the high latency setting. Unfortunately, this is not the case. Even though the fast workers do not need to wait for slow workers, they still need to pull weights from and push gradients to the parameter server, which will encounter the latency (Fig. 3(b)). Therefore it still suffers a communication cost of at least $2T_{\text{latency}}$.

Periodic Synchronization is a possible way to amortize the latency by performing multiple local steps on each machine before pushing the gradients to the parameter server like FedAvg (Konečný et al., 2016), RestartedSGD (Yu et al., 2019) and SlowMo (Wang et al., 2020).

W : pull parameters $\rightarrow$ local evaluation$\times k \rightarrow$ push gradients.

This could amortize the latency to k iterations and the communication cost would be $2T_{\text{latency}}/k$ for ASGD or $(2T_{\text{latency}} + T_{\text{sync}})/k$ for SSGD. But still, the dependency on the latency is linear in the number of iterations and it cannot scale up when latency goes larger.

Decentralized Training is an orthogonal exploration. In SSGD and ASGD, the parameter server communicates with all servers and contains all gradients information. In decentralized learning, such as AD-PSGD (Lian et al., 2017) and D^2 (Tang et al., 2018), each node only communicates with the nearest neighbor and it takes several iterations to broadcast to all nodes. However, it still faces the latency problem when servers are located in different locations.

The ineffectiveness of the existing approaches to deal with the high latency setting motivates us to ask the following question: Is it possible to design an algorithm that could scale up under high latency settings without sacrificing the accuracy? We provide a positive answer by designing Delayed Synchronous SGD (DS-SGD).

3. Methodology

In this section, we introduce our method and provide theoretical guarantee in the smooth non-convex setting. More concretely, we consider the following optimization problem

$$\min_{w} f(w) = \frac{1}{J}\sum_{j=1}^{J} f_j(w),$$

where J denotes the number of workers and each individual function is in the form of $f_j = \mathbb{E}_{\zeta_j}[F_j(w, \zeta_j)]$. In the empirical risk minimization setting, ζ_j corresponds to a mini-batch sampling strategy and f_j could be further expressed as a finite sum of functions $f_j = \frac{1}{K_j}\sum f_{k_j}$.

One of the fundamental concepts that allow our algorithm to tolerate high latency is to pipeline transmission with computation across iterations. In other words, the worker keeps performing local updates and hence not blocked during transmission. This idea comes from the modern implementation of deep learning frameworks (Abadi et al., 2015; Sergeev & Del Balso, 2018), where transmission is pipelined with back-propagation: synchronization of gradients at i^{th} layer can be performed simultaneously with back-propagating at $(i-1)^{th}$ layer. Such a pipeline within a single iteration improves the training speed and does not lead to any stale gradient. However, for a long-distance connection, the effect of stale gradients comes into play.

Let us consider an example that we have one worker located in Boston and the other located in Shanghai. At each local worker, it only takes about 300ms to perform a forward and backward pass on ResNet-50 (He et al., 2016) using recent Nvidia V100 GPU. Yet, transmitting a bit from Boston to Shanghai requires 700ms. In other words, when Boston's worker receives the first gradient from Shanghai's worker, it has already performed 3 local updates. Hence we need to deal with stale gradients.

In our algorithm, we propose a variance reduction type update to incorporate the delayed information:

$$w_{(n+1,j)} = w_{(n,j)} - \eta(g_{(n,j)} - g_{(n-d,j)} + \overline{g_{(n-d)}}), \quad (1)$$

where g refers to a stochastic gradient, $\overline{g}$ is the averaged stale gradients and d is a parameter aiming to measure the relative staleness between communication and computation, e.g.

$$d = \left\lceil \frac{T_{\text{communication}}}{T_{\text{computation}}} \right\rceil.$$

For the generality of the algorithm, we leave d as a hyperparameter which could be specified by the user.

To further understand the update rule, we expand the expression of the iterate in the following way:

$$w_{(n,j)} = \overbrace{w_0 - \eta[\,\overline{g_{(0)}} + \cdots + \overline{g_{(n-d-1)}}]}^{\text{invariant of } j} - \eta\left[\boldsymbol{g_{(n-d,j)}} + g_{(n-d+1,j)} + \cdots g_{(n-1,j)}\right]. \quad (2)$$

The first part of the expression is invariant across all the workers and the second part is the sum of the d most recent local gradients. Then performing the update rule Eq.1 yields

$$w_{(n+1,j)} = \overbrace{w_0 - \eta[\,\overline{g_{(0)}} + \cdots + \overline{g_{(n-d-1)}} + \overline{\boldsymbol{g_{(n-d)}}}]}^{\text{invariant of } j} - \eta\left[g_{(n-d+1,j)} + \cdots g_{(n-1,j)} + \boldsymbol{g_{(n,j)}}\right].$$

In particular, the oldest local gradients $g_{(n-d,j)}$ is replaced by the synchronized gradients $\overline{g_{(n-d)}}$. This is indeed a **delayed synchronization** strategy: at iteration n we synchronize the gradients from iteration $n-d$. In the extreme case when there is no delay, i.e. $d = 0$, we recover the vanilla synchronous SGD.

The delayed synchronization step allows us to take advantage of both synchronization and staleness. A direct consequence is that the parameters on different workers only differ on the most recent d gradients. Under the boundedness assumption on the variance and norm of the stochastic gradient, we can bound the difference between iterates, leading to the convergence analysis.

Algorithm 1 Delayed Synchronous Stochastic gradients Descent (DS-SGD)

Initialize each worker with $w_{0,j} = w_0$ for $j \in [1, J]$.
for $n = 0, 1, \cdots$ **do**
  On local worker j**:**
  Sample stochastic gradients $g_{(n,j)}$ at $w_{(n,j)}$
  Send $g_{(n,j)}$ to all other workers.
  if $n < d$ **then**
    Update with standard SGD rule locally:

$$w_{(n+1,j)} = w_{(n,j)} - \eta g_{(n,j)}.$$

  else
    Update with variance reduction rule:

$$w_{(n+1,j)} = w_{(n,j)} - \eta(g_{(n,j)} - g_{(n-d,j)} + \overline{g_{(n-d)}})$$

    where

$$\overline{g_{(n-d)}} = \frac{1}{J}\sum_{j=1}^{J} g_{(n-d,j)}$$

  end if
end for
Return $\overline{w}_n = \frac{1}{J}\sum_{j=1}^{J} w_{(n,j)}$

3.1. Theoretical Analysis

To start the convergence analysis, we assume that the objective function satisfies the following assumptions.

Assumption 1 (L-smoothness)**.** *Each function $f_j(x)$ is L-smooth, i.e. differentiable with L-Lipschitz gradient:*

$$||\nabla f_j(x) - \nabla f_j(y)|| \leq L||x - y||. \quad \forall x, y \in \mathbb{R}^d$$

Table 1. Comparison with SSGD, ASGD and our DS-SGD. The synchronism guarantees the consistency across workers and usually yields better final accuracy. The staleness improves the efficiency and enables better speed up. With a proper choice of delayed interval d, our algorithm could achieve zero C./C. ratio. Namely, the communication cost is fully covered by the computation cost.

	SYNCHRONISM	STALENESS	COMMUNICATION-TO-COMPUTATION (C./C.) RATIO
SSGD	✓	✗	$(2T_{\text{LATENCY}} + T_{\text{SYNC}})/T_{\text{COMPUTE}}$
ASGD	✗	✓	$2T_{\text{LATENCY}}/T_{\text{COMPUTE}}$
PERIODIC SSGD	✓	✗	$(2T_{\text{LATENCY}} + T_{\text{SYNC}})/pT_{\text{COMPUTE}}$
PERIODIC ASGD	✗	✓	$2T_{\text{LATENCY}}/pT_{\text{COMPUTE}}$
DS-SGD(OURS)	✓	✓	$\max(2T_{\text{LATENCY}} + T_{\text{SYNC}} - d \times T_{\text{COMPUTATION}}, 0)/T_{\text{COMPUTE}}$
PERIODIC DS-SGD(OURS)	✓	✓	$\max(2T_{\text{LATENCY}} + T_{\text{SYNC}} - d \times T_{\text{COMPUTATION}}, 0)/pT_{\text{COMPUTE}}$

Assumption 2 (Bounded gradients and variances)**.** *There exists a positive constant G and σ such that*

$$\mathbb{E}_{\zeta_j}||\nabla F_j(w;\zeta_i)||^2 \leq G^2, \forall w, \forall j.$$

$$\mathbb{E}_{\zeta_j}||\nabla F_j(w;\zeta_j) - \nabla f_j(w)||^2 \leq \sigma^2, \forall w, \forall j.$$

This allows us to upper bound the variance among different workers given the gradients are bounded. Based on the bound, we can derive to the following convergence result:

Theorem 3.1. *Assume that f is differentiable and ∇f is L-Lipschitz. The sequence generated by DS-SGD in Algorithm 1 with stepsize $\eta \leq \frac{1}{L}$ satisfies*

$$\frac{1}{N}\sum_{n=0}^{N-1}\mathbb{E}[\|\nabla f(\overline{w}_n)\|^2] \leq \frac{2}{\eta N}\mathbb{E}[f(w_0) - f(\overline{w}_n)] + 4\eta^2 d^2 G^2 L^2 + \frac{L}{J}\eta\sigma^2.$$

Corollary 3.1.1. *When the function f is lower bounded and $f(w_0) - f^* \leq \Delta$, then let the stepsize $\gamma = \frac{\sqrt{J}}{L\sqrt{N}}$ yields*

$$\frac{1}{N}\sum_{n=0}^{N-1}\mathbb{E}[\|\nabla f(\overline{w}_n)\|^2] = O\left(\frac{\Delta + \sigma^2}{\sqrt{JN}} + \frac{Jd^2}{N}\right).$$

Our proof closely follows the analysis of ASGD in the nonconvex setting (Yu et al., 2019) and we delay it to Appendix A.1. We remark that the convergence rate degrades when the delayed parameter d increases. This is not surprising since a larger d introduces a larger staleness. As long as the delay is in a reasonable range such that $d < O(N^{\frac{1}{4}}J^{-\frac{3}{4}})$, the first term dominates and our algorithm shares the same convergence speed $O(1/\sqrt{JN})$ as the vanilla SSGD.

We would like to emphasize that the delayed synchronization step wisely combines ingredients from synchronization, asynchronous update and variance reduction, but also differs from each of them in a non-trivial way.

Comparison with Synchronous algorithms In the standard synchronized SGD and its variants (Zinkevich et al., 2010; Zhang et al., 2016; Yu et al., 2019; Stich, 2019), whenever the synchronization happens, the parameters on all the workers are unified, i.e. $w_{(n,j)} = w_{(n,k)}$ for any pair of j, k. However, in our algorithm, the delayed synchronization does not affect the recent gradients. Hence, at a fixed iteration $n > 0$, the iterates $w_{(n,j)}$ are in general distinct for different j. Nevertheless, the difference between $w_{(n,j)}$ and $w_{(n,k)}$ is bounded, which is one of the keys for obtaining convergence guarantee.

Comparison with Asynchronous algorithms In the asynchronous SGD, the local parameters are updated on the fly. This introduces inconsistency across the workers. In other words, a fast worker could already be at iteration 10 while a slow worker is still evaluating the gradients at iteration 0. Although the inconsistent staleness does not largely affect the convergence rate of an algorithm, it is commonly observed that the accuracy of the model got highly influenced (Chen et al., 2016; Dai et al., 2019). Several techniques have been introduced to reduce this phenomenon (Ho et al., 2013; Konečnỳ et al., 2016) but the performance is still unstable. In our algorithm, the inconsistency is prevented by the delayed synchronization. This allows our algorithm to achieve no accuracy drop with high speedup. (See experiments in Section 4)

Comparison with variance reduction methods (SVRG) Variance reduction is an important technique that enables a fast convergence rate in convex optimization. One of the most popular methods is SVRG (Johnson & Zhang, 2013), which enjoys the following update

$$w_{n+1} = w_n - \eta\left[\nabla f_i(w_n) - \nabla f_i(\tilde{w}) + \nabla f(\tilde{w})\right],$$

where $\tilde{w}$ is a snapshot reference point and $\nabla f(\tilde{w})$ is the full gradient. The introduction of the reference point reduces the variance of the stochastic gradient, leading to faster convergence analysis than vanilla SGD. Extension of SVRG to synchronous and asynchronous distributed setting has been studied in (Lee et al., 2017; Reddi et al., 2015).

Although our update rule (restated as follows) seems very similar to SVRG,

$$w_{(n+1,j)} = w_{(n,j)} - \eta(g_{(n,j)} - g_{(n-d,j)} + \overline{g_{(n-d)}}),$$

there are two intrinsic differences lies in the choice of the averaged gradients $\overline{g_{(n-d)}}$. First, $\overline{g_{(n-d)}}$ should not be viewed as a full gradient by any mean, because each $g_{(n-d,j)}$ is evaluated at a different location $w_{(n-d,j)}$ (see explanation in comparison with the synchronous algorithms). Second, SVRG uses a fixed reference point $\tilde{w}$ for a non-negligible number of iterations. However, such a strategy is not effective for deep learning (Defazio & Bottou, 2018). In contrast, our averaged gradients $\overline{g_{(n-d)}}$ change along with the iteration, making the "reference point" adaptive.

Last but not least, perhaps the works closest to ours is delayed SGD (Zinkevich et al., 2009; Joulani et al., 2013; Arjevani et al., 2018), where the update rule is given by $w_{n+1} = w_n - \eta g_{(n-d)}$. While such a simple method remains a proof of concept attempt, our algorithm is able to handle complex real-world scenario based on a non-trivial combination of synchronization, stale gradients and variance reduction technique.

In the following, we present several useful variants of the proposed algorithm integrating momentum update and periodic synchronization.

3.2. DS-SGD with momentum update

The momentum update (Polyak, 1964; Qian, 1999) is a widely used technique to accelerate training in modern large scale learning. It incorporates the past gradients into the current update by keeping in memory a descent vector u_k and gradually update it via

$$u_{k+1} = \beta u_k + g_k,$$

where β is the momentum parameter, usually set as 0.9, and $g_{(k)}$ is the most recent gradient. In other words, the momentum update associates different weights to past gradients coming from different iteration. Hence, to perform the delayed synchronization, it suffices to adapt the appropriate coefficients. This leads to the following update rule:

$$u_{(n,j)} = g_{(n,j)} + \beta u_{(n-1,j)} + \beta^d(\overline{g_{(n-d)}} - g_{(n-d,j)})$$
$$w_{(n+1,j)} = w_{(n,j)} - \eta[u_{(n,j)} - \frac{1-\beta^d}{1-\beta}(g_{(n-d,j)} - \overline{g_{(n-d)}}].$$

In order to compensate the momentum update, the variance reduction term is now weighted with a coefficient $(1-\beta^d)/(1-\beta)$. We remark that when $\beta = 0$, we recover the vanilla update of DS-SGD in Algorithm 1.

3.3. DS-SGD with periodic synchronization

One potential problem that DS-SGD may encounter is the congestion. In conventional distributed optimization, only one copy of gradients is transmitted at a given time. In DS-SGD, there can be as many as d copies of gradients transferring over the network at the same time. In the ideal setting, the gradients sent from one worker got received in the same ordering at another worker, e.g., $g_{(n)}$ arrives before $g_{(n+1)}$. However, this might not be the case since the multiple transmission may overwhelm the switch and make copies sent earlier actually arrive later in practice.

In order to overcome this limitation, we integrate a periodic synchronization strategy into DS-SGD. In other words, the gradient transmission (line 5 of Algorithm 1) only executes periodicly when $n \equiv 0 \pmod p$, where p is the period. Moreover, instead of sending one gradient, we send a list of p gradients at a time. Consequently, the delayed synchronization is also performed periodicly, only at iterations n satisfying $n \equiv d \pmod p$. In particular, the oldest p gradients got synchronized simultaneously:

$$w_{(n+1,j)} = w_{(n,j)} - \gamma\left(g_{(n,j)} + v_{(n,j)}\right)$$
$$\text{where} \quad v_{(n,j)} = \sum_{i=n-d-p+1}^{n-d} \overline{g_{(i)}} - \sum_{i=n-d-p+1}^{n-d} g_{(i,j)}$$

This periodic synchronization strategy is similar to the existing model averaging methods (Konečný et al., 2016; Yu et al., 2019; Wang et al., 2020). We emphasize that in these methods, the local worker is locked during the synchronization, hence no computation is allowed before the synchronization ends. In contrast, our algorithm allows local worker to keep performing local updates. In this way, the communication cost is fully covered by the computation.

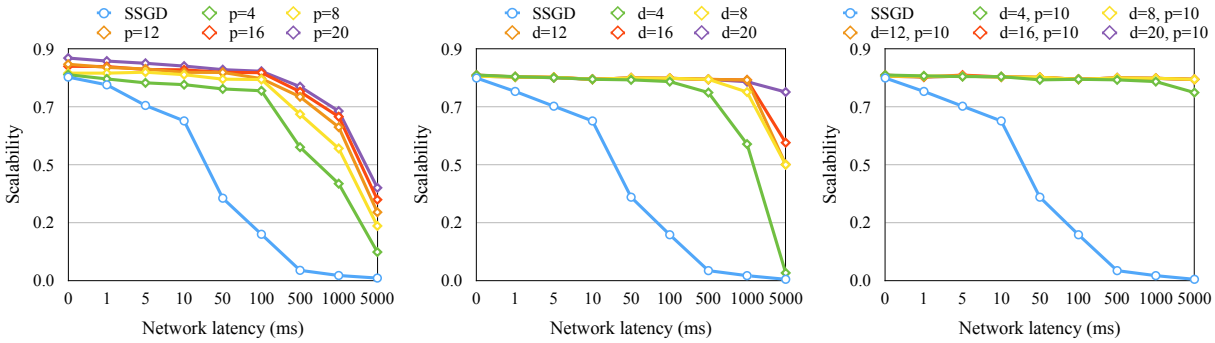
The convergence rate of the periodic synchronization variant is similar to the vanilla DS-SGD, with the delay parameter d replaced by $d+p$. The detail of the algorithm and the proof could be found in Appendex A.2. As we will show in the coming experimental section, the periodic synchronization reduces the congestion issue and allows us to deal with the challenging networking situation.

4. Experiment

Network Our network setup contains two parts: One is inside a cluster where the latency is emulated by *netem*[*]. Another is real-world deployment on AWS EC2 instances where the latency statistics are measured by *qperf*[†]. The emulated setting provides us the flexibility to benchmark on different latency settings, and the real world setting demonstrates the effectiveness of our algorithm.

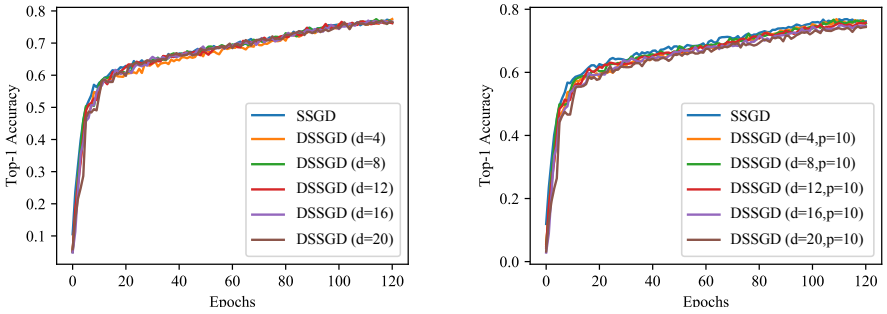
[*] https://wiki.linuxfoundation.org/networking/netem

[†] https://linux.die.net/man/1/qperf



(a) Periodic SSGD (b) DS-SGD (ours) (c) DS-SGD with periodic sync (ours)

Figure 4. The scalability of periodic SSGD, our DS-SGD, and DS-SGD with periodic synchronization.



(a) DS-SGD. (b) DS-SGD with periodic sync.

Figure 5. The top-1 validation accuracy(%) of DS-SGD and DS-SGD with periodic synchronization on ImageNet dataset.

Datasets To evaluate the effectiveness of DS-SGD, we benchmark ResNet-50 (He et al., 2016) on ImageNet (Deng et al., 2009) and LSTM (Hochreiter & Schmidhuber, 1997) on Penn TreeBank (Marcus et al., 1993). ImageNet contains 1.2 million training images and 50,000 validation images in 1000 classes. The Penn Treebank corpus (PTB) dataset consists of 923,000, 73,000 and 82,000 words for training, validation and test respectively.

Implementation We implement DS-SGD in PyTorch framework (Paszke et al., 2017) and adopt Horovod (Sergeev & Del Balso, 2018) as the distributed training backend. We train ImageNet model with momentum SGD using warmup strategy (Goyal et al., 2017) and cosine anneal learning rate decay (Loshchilov & Hutter, 2016) with 120 epochs. Standard data augmentations as random crop and flip are used during training. We set the initial learning rate 0.0125 for which grows linearly w.r.t number of GPUs. For language modeling, we adopt the 2-layer LSTM language model architecture with 1500 hidden units per layer (Press & Wolf, 2016) and train it with gradient clipping and 1 of 40 epochs warm-up period. Our experiments are conducted on 16 standard Amazon *p3.large* EC2 instances with 8 Nvidia Tesla V100 equipped on each. To make results reproducible, we will release the codebase when less anonymous.

Methods We prefer to choose SSGD rather than ASGD as the baseline for two reasons: 1) as shown in Fig. 3, the non-blocking nature of ASGD does not help improving latency issue. 2) synchronous algorithms are widely used in practice as they often attain better accuracy than asynchronous algorithms. We also benchmark its variants periodic SSGD.

4.1. Ablation Studies on Emulated Latency

If a system can achieve M times speedup on N machines, we define its scalability as M/N. The optimal scalability is 1. In practice, the state of the art system can achieve scalability around 0.75-0.8.

As shown in Fig. 4, the scalability of vanilla SSGD degrades drastically when the latency increases. Though periodic SSGD alleviates the issue by amortization, the scalability is still not ideal in high latency settings. DS-SGD, instead of amortizing the communication cost, tries to pipeline transmission with computation as much as possible. By delaying the synchronization, DS-SGD provides much more spare time for transmission and thus scales better than periodic SSGD. As shown in Fig. 4(b), the larger the delayed interval

Table 2. The results of language model on Penn TreeBank dataset.

	Perplexity
Original	72.30
DS-SGD ($d = 4$)	72.28
DS-SGD ($d = 10$)	72.29
DS-SGD ($d = 20$)	72.27

is, the better tolerance towards latency. The scalability keeps stable until maximum tolerance is exceeded. Furthermore, we can integrate periodic sync with DS-SGD to achieve better scalability (Fig. 4(c)).

Besides scalability, we are also interested in the final accuracy. DS-SGD demonstrates robust convergence and comparable performance compared vanilla SSGD even when the delay interval d is as large as 20 (Fig. 5(a)). Considering that ResNet-50 takes 300ms to forward and backward, it means that DS-SGD can tolerate up to 6 seconds latency without deteriorating the accuracy. DS-SGD with periodic synchronization yields a even better scalability (Fig. 4(c)) while the training accuracy is also well-preserved (Fig. 5(b))

Moreover, we also evaluate DS-SGD on Penn TreeBank dataset. Fig. 2 shows the perplexity of the language model while the delay interval gradually increases from 4 to 20. The performance of DS-SGD closely matches the baseline while a high network latency can be tolerated.

4.2. Distributed Training Across the World

To evaluate DS-SGD in real world settings, we deploy eight different servers in four locations across the world: Tokyo(Japan), London (U.K), Oregon (U.S.A) and Ohio(U.S.A). The latency statistics is shown in Fig.2. The real latency across the ring allreduce (Patarasuk & Yuan, 2009) is about 479ms through Ethernet. On V100 GPU, ResNet-50 takes around 300ms to forward and backward. Therefore delayed update with steps more than 4 is already enough to tolerate such a network. As shown in Fig. 5(a), DS-SGD does not suffer from accuracy drop when delayed interval is 4. In our experiments, we set d to 20 to avoid packet loss and unexpected network fluctuation.

Now we are close to our goal of distributed training across the world. However, there is still a bottleneck on bandwidth for cross-continent connections (e.g. Tokyo to London). Though we are optimistic that bandwidth can be always improved by hardware upgrade, unfortunately AWS does not offers such an option. As an economical alternative, we apply deep gradient compression (Lin et al., 2018) to overcome the bandwidth limitation. We set the gradient compression rate to 99% in experiments, which means only 1% percent of the gradients are sent in transmission.

Tab. 3 shows the training accuracy of our proposed approach,

Table 3. Training results of image classification with 8 servers across the world. We aggressively applied DS-SGD with periodic synchronization and gradient compression. This can scale distributed training under very bad network (e.g., cross continent connections) with a negligible drop on accuracy.

Task	Image Classification for ImageNet			
Training Method	d	p	c	Top-1 Accuracy
Baseline	0	0	100	76.63%
DS-SGD	4	4	10	76.15%
DS-SGD	8	8	10	76.32%
DS-SGD	12	8	1	76.18%
DS-SGD	20	10	1	75.81%

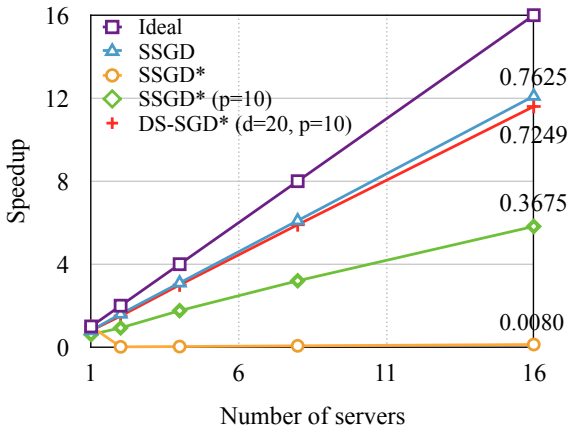


Figure 6. DS-SGD with periodic syncrhonizaiton demonstrates good scalability and accuracy on world-wide high latency training. The superscript $*$ indicates the training speed in the across the world setting.

where d and p and c indicate the delayed steps, periodic synchronization interval and gradient density respectively. On large scale classification task, the accuracy loss of our method is negligible while the training speed is improved by a large margin. Shown in Fig. 6, while conventional algorithms fails to scale up as the latency increases (0.008 for SSGD and 0.36 for periodic SSGD), DS-SGD yields a strong scalability (0.72) under cross-world latency, which is 90× better than SSGD and 2× faster than periodic SSGD. This performance is close to what conventional algorithms achieved inside a data center (0.78).

5. Conclusion

In this paper, we propose the Delayed Synchronous SGD to scale distributed training on servers located at different continents across the world. We demonstrate that our algorithm is capable of enjoying high scalability and preserving accuracy under poor network conditions. We believe that our work will open future avenues for a wide range of cross-region learning applications.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

Arjevani, Y., Shamir, O., and Srebro, N. A tight convergence analysis for stochastic gradient descent with delayed updates. *arXiv preprint arXiv:1806.10188*, 2018.

Chen, J., Pan, X., Monga, R., Bengio, S., and Jozefowicz, R. Revisiting distributed synchronous sgd. *arXiv preprint arXiv:1604.00981*, 2016.

Chen, T., Li, M., Li, Y., Lin, M., Wang, N., Wang, M., Xiao, T., Xu, B., Zhang, C., and Zhang, Z. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274*, 2015.

Dai, W., Zhou, Y., Dong, N., Zhang, H., and Xing, E. Toward understanding the impact of staleness in distributed machine learning. In *International Conference on Learning Representations*, 2019.

De Sa, C. M., Zhang, C., Olukotun, K., and Ré, C. Taming the wild: A unified analysis of hogwild-style algorithms. In *Advances in neural information processing systems (NIPS)*, 2015.

Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., Senior, A., Tucker, P., Yang, K., Le, Q. V., et al. Large scale distributed deep networks. In *Advances in neural information processing systems*, pp. 1223–1231, 2012.

Defazio, A. and Bottou, L. On the ineffectiveness of variance reduced optimization for deep learning. *arXiv preprint arXiv:1812.04529*, 2018.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.

GDPR. The eu general data protection regulation (gdpr). 2016.

Goyal, P., Dollár, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., Tulloch, A., Jia, Y., and He, K. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Ho, Q., Cipar, J., Cui, H., Lee, S., Kim, J. K., Gibbons, P. B., Gibson, G. A., Ganger, G., and Xing, E. P. More effective distributed ml via a stale synchronous parallel parameter server. In *Advances in neural information processing systems*, 2013.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Hsieh, K., Harlap, A., Vijaykumar, N., Konomis, D., Ganger, G. R., Gibbons, P. B., and Mutlu, O. Gaia: Geo-distributed machine learning approaching lan speeds. In *NSDI*, pp. 629–647, 2017.

Jochems, A., Deist, T. M., Van Soest, J., Eble, M., Bulens, P., Coucke, P., Dries, W., Lambin, P., and Dekker, A. Distributed learning: developing a predictive model based on data from multiple hospitals without data leaving the hospital–a real life proof of concept. *Radiotherapy and Oncology*, 121(3):459–467, 2016.

Jochems, A., Deist, T. M., El Naqa, I., Kessler, M., Mayo, C., Reeves, J., Jolly, S., Matuszak, M., Ten Haken, R., van Soest, J., et al. Developing and validating a survival prediction model for nsclc patients through distributed learning across 3 countries. *International Journal of Radiation Oncology, Biology, Physics*, 99(2):344, 2017.

Johnson, R. and Zhang, T. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in neural information processing systems (NIPS)*, 2013.

Joulani, P., Gyorgy, A., and Szepesvári, C. Online learning under delayed feedback. In *International Conference on Machine Learning (ICML)*, 2013.

Konečnỳ, J., McMahan, H. B., Yu, F. X., Richtárik, P., Suresh, A. T., and Bacon, D. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

Konečný, J., McMahan, H. B., Yu, F. X., Richtarik, P., Suresh, A. T., and Bacon, D. Federated learning: Strategies for improving communication efficiency. In *NIPS Workshop on Private Multi-Party Machine Learning*, 2016.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems (NIPS)*, 2012.

Lee, J. D., Lin, Q., Ma, T., and Yang, T. Distributed stochastic variance reduced gradient methods by sampling extra data with replacement. *The Journal of Machine Learning Research*, 18(1):4404–4446, 2017.

Li, M., Andersen, D. G., Park, J. W., Smola, A. J., Ahmed, A., Josifovski, V., Long, J., Shekita, E. J., and Su, B.-Y. Scaling distributed machine learning with the parameter server. In *Operating Systems Design and Implementation (OSDI)*, 2014.

Lian, X., Huang, Y., Li, Y., and Liu, J. Asynchronous parallel stochastic gradient for nonconvex optimization. In *Advances in Neural Information Processing Systems (NIPS)*, 2015.

Lian, X., Zhang, W., Zhang, C., and Liu, J. Asynchronous decentralized parallel stochastic gradient descent. *arXiv preprint arXiv:1710.06952*, 2017.

Lin, Y., Han, S., Mao, H., Wang, Y., and Dally, W. J. Deep gradient compression: Reducing the communication bandwidth for distributed training. In *International Conference on Learning Representations (ICLR)*, 2018.

Loshchilov, I. and Hutter, F. Sgdr: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations (ICLR)*, 2016.

Marcus, M. P., Santorini, B., and Marcinkiewicz, M. A. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, 19(2):313–330, 1993.

McMahan, B., Moore, E., Ramage, D., Hampson, S., and y Arcas, B. A. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*, 2017.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

Patarasuk, P. and Yuan, X. Bandwidth optimal all-reduce algorithms for clusters of workstations. *Journal of Parallel and Distributed Computing*, 69(2):117–124, 2009.

Polyak, B. T. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

Press, O. and Wolf, L. Using the output embedding to improve language models. *arXiv preprint arXiv:1608.05859*, 2016.

Qian, N. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1):145–151, 1999.

Recht, B., Re, C., Wright, S., and Niu, F. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in neural information processing systems*, pp. 693–701, 2011.

Reddi, S. J., Hefny, A., Sra, S., Poczos, B., and Smola, A. J. On variance reduction in stochastic gradient descent and its asynchronous variants. In *Advances in neural information processing systems*, pp. 2647–2655, 2015.

Seide, F., Fu, H., Droppo, J., Li, G., and Yu, D. 1-bit stochastic gradient descent and its application to data-parallel distributed training of speech dnns. In *Fifteenth Annual Conference of the International Speech Communication Association*, 2014.

Sergeev, A. and Del Balso, M. Horovod: fast and easy distributed deep learning in tensorflow. *arXiv preprint arXiv:1802.05799*, 2018.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations (ICLR)*, 2015.

Stich, S. U. Local SGD converges fast and communicates little. In *International Conference on Learning Representations*, 2019.

Tang, H., Lian, X., Yan, M., Zhang, C., and Liu, J. D^2: Decentralized training over decentralized data. In *International Conference on Machine Learning*, 2018.

Tsitsiklis, J., Bertsekas, D., and Athans, M. Distributed asynchronous deterministic and stochastic gradient optimization algorithms. *IEEE transactions on automatic control*, 31(9):803–812, 1986.

Valiant, L. G. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, 1990.

Wang, J., Tantia, V., Ballas, N., and Rabbat, M. Slowmo: Improving communication-efficient distributed sgd with slow momentum. In *International Conference on Learning Representations (ICLR)*, 2020.

Yu, H., Yang, S., and Zhu, S. 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*, 2019.

Zhang, J., De Sa, C., Mitliagkas, I., and Ré, C. Parallel sgd: When does averaging help? *arXiv preprint arXiv:1606.07365*, 2016.

Zinkevich, M., Langford, J., and Smola, A. J. Slow learners are fast. In *Advances in neural information processing systems*, pp. 2331–2339, 2009.

Zinkevich, M., Weimer, M., Li, L., and Smola, A. J. Parallelized stochastic gradient descent. In *Advances in neural information processing systems (NIPS)*, 2010.

A. Supplementary material

A.1. Proof of Theorem 3.1

Lemma A.1. *Let $(w_{(n,j)})$ be the sequence generated by DS-SGD, then under the bounded gradient and variance Assumption 2, for any $j_0 \in [1, J]$*

$$\mathbb{E}\left[\|\overline{w}_n - w_{(n,j_0)}\|^2\right] \leq 4\eta^2 t^2 G^2,$$

where $\overline{w}_n = \frac{1}{J}\sum_{j=1}^{J} w_{(n,j)}$.

Proof. According to the delayed synchronization formula (2), the update up to $n-d$ are invariant across machines. Hence,

$$\begin{aligned}
&\mathbb{E}\left[\|\overline{w}_n - w_{(n,j_0)}\|^2\right] \\
=&\eta^2\mathbb{E}\left[\left\|\frac{1}{J}\sum_{j=1}^{J}\sum_{i=n-d}^{n-1} g_{(i,j)} - \sum_{i=n-d}^{n-1} g_{(i,j_0)}\right\|^2\right] \\
\leq& 2\eta^2\mathbb{E}\left[\left\|\frac{1}{J}\sum_{j=1}^{J}\sum_{i=n-d}^{n-1} g_{(i,j)}\right\|^2\right] + 2\eta^2\mathbb{E}\left[\|\sum_{i=n-d}^{n-1} g_{(i,j_0)}\|^2\right] \\
\text{(Cauchy Schwartz's inequality)} \leq& \frac{2\eta^2 d}{J}\sum_{j=1}^{J}\sum_{i=n-d}^{n-1}\mathbb{E}\left[\left\|g_{(i,j)}\right\|^2\right] + 2\eta^2 d\sum_{i=n-d}^{n-1}\mathbb{E}\left[\|g_{(i,j_0)}\|^2\right] \\
\leq& 4\eta^2 d^2 G^2.
\end{aligned}$$

□

Proof of Theorem 3.1. By smoothness of f, we have

$$\mathbb{E}[f(\overline{g_{(n+1)}})] \leq \mathbb{E}[f(\overline{g_{(n)}}) + \langle \nabla f(\overline{g_{(n)}}), \overline{g_{(n+1)}} - \overline{g_{(n)}}\rangle + \frac{L}{2}\|\overline{g_{(n+1)}} - \overline{g_{(n)}}\|^2].$$

Note that

$$\begin{aligned}
\overline{g_{(n+1)}} &= \frac{1}{J}\sum_{j=1}^{J} w_{(n+1,j)} = \frac{1}{J}\sum_{j=1}^{J}[w_{(n,j)} - \eta(g_{(n,j)} - g_{(n-d,j)} + \overline{g_{(n-d)}})] \\
&= \overline{g_{(n)}} - \frac{\eta}{J}\sum_{j=1}^{J} g_{(n,j)}.
\end{aligned}$$

This implies that

$$\begin{aligned}
&\mathbb{E}[\|\overline{g_{(n+1)}} - \overline{g_{(n)}}\|^2] \\
=&\eta^2\mathbb{E}[\|\frac{1}{J}\sum_{j=1}^{J} g_{(n,j)}\|^2] \\
=&\eta^2\mathbb{E}[\|\frac{1}{J}\sum_{j=1}^{J}(g_{(n,j)} - \nabla f_j(w_{(n,j)})\|^2] + \eta^2\mathbb{E}[\|\frac{1}{J}\sum_{j=1}^{J}\nabla f_j(w_{(n,j)})\|^2] \\
=&\frac{\eta^2}{J^2}\sum_{j=1}^{J}\mathbb{E}[\|(g_{(n,j)} - \nabla f_j(w_{(n,j)})\|^2] + \eta^2\mathbb{E}[\|\frac{1}{J}\sum_{j=1}^{J}\nabla f_j(w_{(n,j)})\|^2] \\
\leq&\frac{\eta^2\sigma^2}{J} + \eta^2\mathbb{E}[\|\frac{1}{J}\sum_{j=1}^{J}\nabla f_j(w_{(n,j)})\|^2].
\end{aligned}$$

Moreover,

$$\begin{aligned}
&\mathbb{E}[\langle \nabla f(\overline{g_{(n)}}), \overline{g_{(n+1)}} - \overline{g_{(n)}}] \\
=& -\eta \mathbb{E}[\langle \nabla f(\overline{g_{(n)}}), \frac{1}{J}\sum_{j=1}^{J} g_{(n,j)}] \\
=& -\eta \mathbb{E}[\langle \nabla f(\overline{g_{(n)}}), \frac{1}{J}\sum_{j=1}^{J} \nabla f_j(w_{(n,j)}] \\
=& \frac{\eta}{2}\mathbb{E}[\|\nabla f(\overline{g_{(n)}}) - \frac{1}{J}\sum_{j=1}^{J}\nabla f_j(w_{(n,j)}\|^2] - \frac{\eta}{2}\mathbb{E}[\|\nabla f(\overline{g_{(n)}})\|^2] - \frac{\eta}{2}\mathbb{E}[\|\frac{1}{J}\sum_{j=1}^{J}\nabla f_j(w_{(n,j)})\|^2]
\end{aligned}$$

Applying Lemma A.1 with the L-smoothness, we have

$$\begin{aligned}
&\mathbb{E}[\|\nabla f(\overline{g_{(n)}}) - \frac{1}{J}\sum_{j=1}^{J}\nabla f_j(w_{(n,j)}\|^2] \\
=&\mathbb{E}[\|\frac{1}{J}\sum_{j=1}^{J}(\nabla f_j(\overline{g_{(n)}}) - \nabla f_j(w_{(n,j)})\|^2] \\
\text{(Jensen's inequality)} \leq& \frac{1}{J}\sum_{j=1}^{J}\mathbb{E}[\|(\nabla f_j(\overline{g_{(n)}}) - \nabla f_j(w_{(n,j)})\|^2] \\
\text{(L-smoothness)} \leq& \frac{L^2}{J}\sum_{j=1}^{J}\|\overline{g_{(n)}} - w_{(n,j)}\|^2 \\
\text{(Lemma A.1)} \leq& 4\eta^2 d^2 L^2 G^2.
\end{aligned}$$

Thus, combining the above inequalities together with $\eta \leq \frac{1}{L}$ yields

$$\begin{aligned}
&\mathbb{E}[f(\overline{g_{(n+1)}})] - \mathbb{E}[f(\overline{g_{(n)}})] \\
\leq& \frac{L\eta^2 - \eta}{2}\mathbb{E}[\|\frac{1}{J}\sum_{j=1}^{J}\nabla f_j(w_{(n,j)})\|^2] - \frac{\eta}{2}\mathbb{E}[\|\nabla f(\overline{g_{(n)}})\|^2] + 2\eta^3 L^2 d^2 G^2 + \frac{L}{2J}\eta^2\sigma^2 \\
\leq& -\frac{\eta}{2}\mathbb{E}[\|\nabla f(\overline{g_{(n)}})\|^2] + 2\eta^3 L^2 d^2 G^2 + \frac{L}{2J}\eta^2\sigma^2.
\end{aligned}$$

Rearrange the terms yields

$$\mathbb{E}[\|\nabla f(\overline{g_{(n)}})\|^2] \leq \frac{2}{\eta}(\mathbb{E}[f(\overline{g_{(n)}})] - \mathbb{E}[f(\overline{g_{(n+1)}})]) + 4\eta^2 L^2 d^2 G^2 + \frac{L}{J}\eta\sigma^2$$

Telescoping from $n = 0, ..N-1$ yields

$$\frac{1}{N}\sum_{n=0}^{N-1}\mathbb{E}[\|\nabla f(\overline{g_{(n)}})\|^2] \leq \frac{2}{\eta N}(\mathbb{E}[f(\overline{g_{(0)}})] - \mathbb{E}[f(\overline{g_{(N)}})]) + 4\eta^2 L^2 d^2 G^2 + \frac{L}{J}\eta\sigma^2$$

□

Corollary A.1.1. *When the number of iteration N surpass the number of machines J, let the stepsize $\eta = \frac{\sqrt{J}}{L\sqrt{N}}$, yields*

$$\begin{aligned}
\frac{1}{N}\sum_{n=0}^{N-1}\mathbb{E}[\|\nabla f(\overline{g_{(n)}})\|^2] &\leq \frac{2L}{\sqrt{JN}}(f(w_0) - f^* + \sigma^2) + \frac{4d^2G^2J}{N}. \\
&= O(\frac{1}{\sqrt{JN}}) + O(\frac{t^2 J}{N})
\end{aligned}$$

A.2. DS-SGD with periodic synchronization

Algorithm 2 DS-SGD with periodic synchronization

Initialize each worker with $w_{0,j} = w_0$ for $j \in [1, J]$.
for $n = 0, 1, \cdots$ **do**
  On local worker j**:**
  Sample a stochastic gradients $g_{(n,j)}$ at $w_{(n,j)}$
  if $n \equiv 0 \pmod p$ **then**
    Send $g_{(n,j)}, g_{(n-1,j)}, \cdots g_{(n-p+1,j)}$ to all other workers.
  end if
  if $n \equiv t \pmod p$ **then**
    Update with variance reduction rule:

$$w_{(n+1,j)} = w_{(n,j)} - \eta \left(g_{(n,j)} - \sum_{i=n-d-p+1}^{n-d} g_{(i,j)} + \sum_{i=n-d-p+1}^{n-d} \overline{g_{(i)}} \right)$$

    where

$$\overline{g_{(i)}} = \frac{1}{J} \sum_{j=1}^{J} g_{(i,j)}$$

  else
    Update with standard SGD rule locally:

$$w_{(n+1,j)} = w_{(n,j)} - \eta g_{(n,j)}.$$

  end if
end for
Return $\overline{w}_n = \frac{1}{J} \sum_{j=1}^{J} w_{(n,j)}$

Notably, the update in the periodic synchronization can be rewritten as

$$w_{(n,j)} = \underbrace{w_0 - \eta \sum_{i=0}^{p\lfloor \frac{n-d}{p} \rfloor} \overline{g_{(i)}}}_{\text{invariant of } j} - \eta \sum_{i=p\lfloor \frac{n-d}{p} \rfloor + 1}^{n-1} g_{(i,j)}, \tag{TS}$$

where the first quantity is commonly shared among all the workers and the second term corresponds to local updates.

Lemma A.2. *Let $(w_{(i,j)})$ be the sequence generated on the j-th worker according to Temporally Sparse Update Algorithm, then for any $j_0 \in [1, J]$*

$$\mathbb{E}\left[\|\overline{g_{(n)}} - w_{(n,j_0)}\|^2\right] \leq 4\eta^2 (d+p)^2 G^2$$

Proof. From the update in periodic synchronization, we have

$$\begin{aligned}
&\mathbb{E}\left[\|\overline{g_{(n)}} - w_{(n,j_0)}\|^2\right] \\
=& \eta^2 \mathbb{E}\left[\left\| \frac{1}{J} \sum_{j=1}^{J} \sum_{i=p\lfloor \frac{n-d}{p} \rfloor + 1}^{n-1} g_{(i,j)} - \sum_{i=p\lfloor \frac{n-d}{p} \rfloor + 1}^{n-1} g_{(i,j_0)} \right\|^2 \right] \\
\leq& 2\eta^2 \mathbb{E}\left[\left\| \frac{1}{J} \sum_{j=1}^{J} \sum_{i=p\lfloor \frac{n-d}{p} \rfloor + 1}^{n-1} g_{(i,j)} \right\|^2 \right] + \mathbb{E}\left[\| \sum_{i=p\lfloor \frac{n-d}{p} \rfloor + 1}^{n-1} g_{(i,j_0)} \|^2 \right] \\
\leq& 4\eta^2 (d+p)^2 G^2.
\end{aligned}$$

□

Theorem A.3. *Consider the sequence generated by the Temporally Sparse Update Algorithm with stepsize $\eta \leq \frac{1}{L}$, then*

$$\frac{1}{N}\sum_{n=0}^{N-1}\mathbb{E}[\|\nabla f(\overline{g_{(n)}})\|^2] \leq \frac{2}{\eta N}(f(w_0)-f^*) + 4\eta^2(d+p)^2G^2L^2 + \frac{L}{J}\eta\sigma^2.$$

Proof. Notice that for both variance reduction update and the SGD update, the following relation holds

$$\overline{g_{(n+1)}} = \overline{g_{(n)}} - \frac{\eta}{J}\sum_{j=1}^{J} g_{(n,j)}.$$

The proof follows the exact same schema as the vanilla DS-SGD algorithm by noticing based on Lemma A.2. □

Corollary A.3.1. *When the number of iteration N surpass the number of machines J, let the stepsize $\eta = \frac{\sqrt{J}}{L\sqrt{N}}$, yields*

$$\frac{1}{N}\sum_{n=0}^{N-1}\mathbb{E}[\|\nabla f(\overline{g_{(n)}})\|^2] = O(\frac{1}{\sqrt{JN}}) + O(\frac{(d+p)^2J}{N})$$

A.3. DS-SGD with momentum

Algorithm 3 DS-SGD with momentum

Initialize each worker with $w_{0,j} = w_0$ and momentum vector $u_{(-1,j)} = 0$ for $j \in [1, J]$.
for $n = 0, 1, \cdots$ **do**
  On local worker j**:**
  Sample a stochastic gradients $g_{(n,j)}$ at $w_{(n,j)}$
  Send $g_{(n,j)}$ to all other workers.
  if $n < t$ **then**
    Update with momentum SGD rule locally:

$$u_{(n,j)} = g_{(n,j)} + \beta u_{(n-1,j)},$$
$$w_{(n+1,j)} = w_{(n,j)} - \eta u_{(n,j)}.$$

  else
    Update with variance reduction rule:

$$u_{(n,j)} = g_{(n,j)} + \beta u_{(n-1,j)} + \beta^d(\overline{g_{(n-d)}} - g_{(n-d,j)})$$
$$w_{(n+1,j)} = w_{(n,j)} - \eta u_{(n,j)} - \eta\frac{1-\beta^d}{1-\beta}(\overline{g_{(n-d)}} - g_{(n-d,j)}).$$

    where

$$\overline{g_{(n-d)}} = \frac{1}{J}\sum_{j=1}^{J} g_{(n-d,j)}$$

  end if
end for
Return $\overline{w}_n = \frac{1}{J}\sum_{j=1}^{J} w_{(n,j)}$