DISTRIBUTED TRAINING ACROSS THE WORLD

Anonymous authors
Paper under double-blind review

ABSTRACT

Traditional synchronous distributed training is performed inside a cluster since it requires high bandwidth and low latency network (*e.g.* 25Gb Ethernet or Infiniband). However, in many application scenarios, training data are often distributed *across many geographic locations*, where physical distance is long and latency is high. Traditional synchronous distributed training cannot scale well under such limited network conditions. In this work, we aim to **scale distributed learning under high-latency network**. To achieve this, we propose **D**elayed and **T**emporally **S**parse (DTS) update that enables synchronous training to tolerate extreme network conditions without compromising accuracy. We benchmark our algorithms on servers deployed across three continents in the world: London (Europe), Tokyo (Asia), Oregon (North America) and Ohio (North America). Under such challenging settings, DTS achieves $90\times$ speedup over traditional methods without loss of accuracy on ImageNet.

1 INTRODUCTION

Deep neural networks have demonstrated much success in solving large-scale machine learning problems (Krizhevsky et al., 2012; Simonyan & Zisserman, 2014; He et al., 2016). However, training deep neural networks may take days or even weeks to converge. In order to enable training in a reasonable time, distributed training is an important technique and gains increasing attention (Li et al., 2014; Dean et al., 2012; Recht et al., 2011; Goyal et al., 2017; Jia et al., 2018). To maintain a good scalability, a *low latency* and *high bandwidth* network is essential for most modern distributed systems. Existing frameworks (Chen et al., 2015; Xing et al., 2015; Moritz et al., 2015; Abadi et al., 2015; Akiba et al., 2017; Paszke et al., 2017; Sergeev & Del Balso, 2018) all require high-end network infrastructure such as 25Gbps Ethernet or Infiniband where bandwidth is as large as 10 to 100 Gbps and latency is as small as 1 us.

Bandwidth is easy to increase (*e.g.* stacking hardware) but latency is hard to improve (physical limits). For example, if we have two servers located at Shanghai and Boston respectively, even at the speed of light and direct air distance, it still takes 78ms* to send and receive a packet. In real world scenario, the latency can be only worse (around 700ms) because indirect routing between internet service providers (ISP) and queuing delay in switches. Such high latency cause severe scalability issue for distributed training. As shown in Fig. 1, traditional distributed training algorithm scales poorly under such large latency.

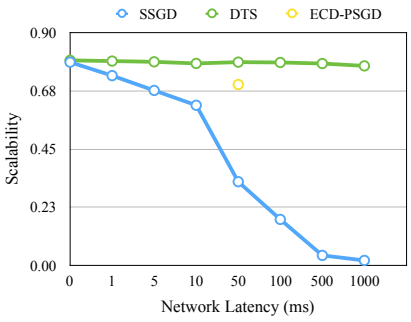


Figure 1: DTS maintains a good scalability when network latency increases, while the performance of conventional algorithms degrades quickly.

In many scenarios, the training data involves privacy-sensitive information, such as personal medical history (Jochems et al., 2016; 2017) and keyboard input history (McMahan et al., 2016; Konen et al., 2016; Bonawitz et al., 2019), thus cannot be centralized to a data center due to security and legacy concerns (GDPR, 2016). When datasets are distributed across many locations across public clouds, private clusters and even edge devices, it is impossible to setup a low latency network under long-distance connections and thereby hurts the scalability of training.

*$11,725\text{km} \times 2/(3 \times 10^8\text{m/s}) = 78.16\text{ms}$. Information collected from Google Maps.

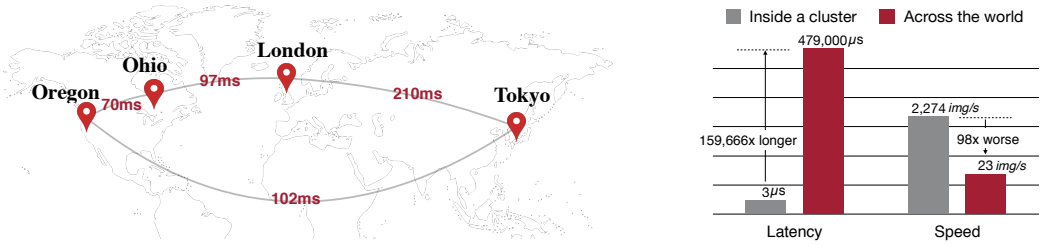


Figure 2: Network conditions across different continents (Left) and the comparison between connections inside a cluster (Right). Different from training inside a data center, long-distance distributed training suffers from a high-latency network, which proposes a severe challenge to scale across the world.

In this work, we enable **scalable distributed training across different geographical locations** with long distance and high latency network connection. We propose *Delayed Update* to tolerate latency by putting off the synchronization barrier to a later iteration; we also propose *Temporally Sparse Update* to amortize the latency and alleviate congestion by reducing the synchronization frequency. To ensure no loss of accuracy, we design a novel error compensation to overcome the staleness for both vanilla and momentum SGD. We focus on the widely adopted synchronous update using data parallelism. As shown in Fig. 1, DTS can maintain high scalability even when latency is as high as 1000ms. This result is also better than existing state of art technologies such as ECD-PSGD (results copied directly from their original paper (Tang et al., 2018a))

We benchmark our DTS under a challenging settings: training a deep neural network on four AWS P3 instances located in different continents of the world (Fig. 2). The measured latency is as large as 277ms (compared to internal latency 2us). In this case, the naive distributed synchronous SGD (SSGD) can only achieve a poor scalability of 0.008. It means in this setting, distributed training with 100 servers is even slower than single machine (0.8 v.s. 1.0). Meanwhile DTS achieves scalability of 0.72 without compromising the accuracy. In conclusion, our contributions are listed below:

- We propose delayed update to tolerate the latency. With our error compensation technique, delayed update preserves the training accuracy while tolerates up to 6 seconds latency.
- We propose temporally sparse update to amortize the latency among multiple steps. It drastically reduced the communication traffic by 20× without loss of accuracy.
- With servers and data distributed in four different countries across the world, we can train ResNet-50 on ImageNet with scalability. To our best knowledge, DTS is the first work that can achieve scalable synchronous distributed training under such high latency.

2 Related Work

Distributed Learning becomes ever more important as the sizes of both datasets and models increase. Many studies have been made to explore the efficiency, both at the algorithm level (Li et al., 2014; Dean et al., 2012; Recht et al., 2011; Goyal et al., 2017; Jia et al., 2018) and at the framework level (Chen et al., 2015; Akiba et al., 2017; Abadi et al., 2015; Paszke et al., 2017; Sergeev & Del Balso, 2018). In most of distributed algorithms, each node performs computation and exchange updates through network. A key component to improve scalability is to reduce the communication-to-computation ratio. The communication cost is determined by latency and bandwidth. Conventional studies focus on reducing bandwidth requirements as the latency of internal connection is usually low.

Gradient quantization / compression has been proposed to reduce the data to be transferred. One-bit gradient gradient (Seide et al., 2014) achieves 10× speedup using 20 GPUs on text-to-speech task. QSGD (Alistarh et al., 2016) and Terngrad (Wen et al., 2017) further improves the trade-off between accuracy and gradient precision. But the theoretical limit of the quantization cannot go beyond 32. To overcome the limitation, gradient sparsification using predefined static threshold (Strom, 2015; Aji & Heafield, 2017) and dynamic compression ratio (Chen et al., 2018) demonstrate that 99% gradients

can be pruned with negligible degradation on model performance. (Sattler et al., 2018) combines both quantization and compression to push the ratio to a new level. DGC (Lin et al., 2017) and DoubleSequeeze (Tang et al., 2019) explore how to compensate the error to preserve the accuracy better. The convergence of quantization and compression is also discussed in (Tang et al., 2019; Jiang & Agrawal, 2018; Alistarh et al., 2018). However, latency remains an issue, especially when servers are not in the same physical location.

Learning on decentralized data has become increasingly popular recently as the growing awareness of data privacy (GDPR, 2016). For example, Federated Learning (Google, 2017) aim to jointly train a model without centralizing the data and have been used to train models for medical treatments across multiple hospitals (Jochems et al., 2016), analyze patient survival situations from various countries (Jochems et al., 2017) and build predictive keyboards to improve typing experience (McMahan et al., 2016; Google, 2017; Bonawitz et al., 2019). However, existing federated learning works do not scale well under high latency network. An orthogonal exploration is Decentralized Training where only partial synchronization is performed in each update, such as AD-PSGD (Lian et al., 2017) and D^2 (Tang et al., 2018b). None of them has been evaluated on large learning tasks yet.

Asynchronous SGD (ASGD) is derived from (Tsitsiklis et al., 1986) and has advantages in unstable latency and fault tolerance. Different from synchronous SGD (SSGD), ASGD relaxes synchronization by allowing training on inconsistent models and powers many successful applications such as HOGWILD! (Recht et al., 2011), BUCKWILD! (De Sa et al., 2015) and dist-belief (Dean et al., 2012). However, most of them are implemented through *parameter server* (Dean et al., 2012; Chilimbi et al., 2014; Li et al., 2014) and leads to problems like communication congestion and resource congestion when scaling up. Moreover, training on inconsistent models leads to different behaviors compared to training on a single device. Though there are studies discussing the convergences (Recht et al., 2011; De Sa et al., 2015) and showing that ASGD converges as good as SSGD (Lian et al., 2015), synchronous SGD (SSGD) is usually preferred in practice (Goyal et al., 2017; Sun et al., 2019) because it has consistent behaviors when increasing the number of machines and therefore is easy to develop and deploy.

3 Approach

To motivate this section, we first review the mechanism of synchronous distributed training: at each step, each node will first compute gradients locally, then they wait for the collective operation to transmit gradients to each other to calculate the average. They use the averaged gradient to update the weight and continue to the next step. As shown in right of Fig. 3a, when the communication increases , the whole training process would be drastically slowed down.

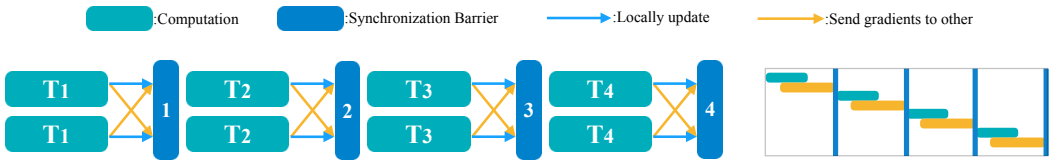
To address the problem, our proposed algorithm contains two parts: *delayed update* (Fig. 3b) and *temporally sparse update* (Fig. 3c). With the delayed update, instead of waiting for average gradients to come back, it puts off the synchronization barrier to steps later to tolerate the latency. With the temporally sparse update, the synchronization frequency is reduced to amortize latency.

Throughout the paper, we use the following notions:

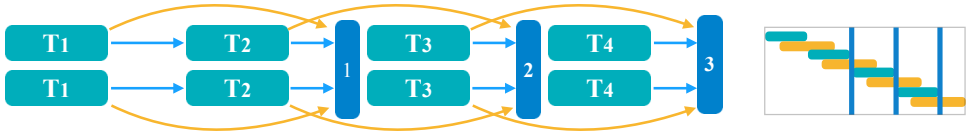
- λ: learning rate.
- t: iteration steps that the synchronization barrier is delayed.
- p: iteration steps between temporally sparse update intervals.
- $u_{(i,j)}$: momentum at iteration i on worker j.
- $u'_{(i,j)}$: error compensated momentum at iteration i on worker j.
- $w_{(i,j)}$: model weights at iteration i on worker j.
- $w'_{(i,j)}$: error compensated weights at iteration i on worker j.
- $\nabla w_{(i,j)}$: local gradients at iteration i on worker j.
- $\overline{\nabla w_{(i)}}$: global averaged gradients at iteration i.

3.1 Delayed Update

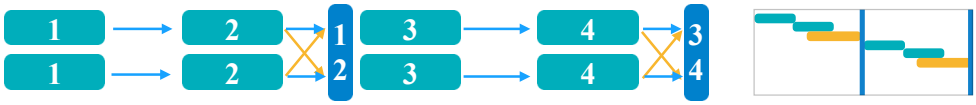
In modern implementations of vanilla SSGD, the transmission is partially pipelined with back-propogration: Synchronizing gradients of n^{th} layer can be performed simultaneously with back-propagating $(n-1)^{\text{th}}$ layer. However, it is not enough to cover communication since latency on



(a) Synchronous distributed learning.



(b) Synchronous distributed learning with delayed update.



(c) Synchronous distributed learning with temporally sparse update.

Figure 3: The visualization of our proposed techniques (*Left*) and the execution pipeline (*Right*). Yellow lines and blocks indicate communication, while green ones indicate computation. Delayed update covers communication by computation and requires the same number of synchronization. Temporally sparse reduces the synchronization frequency and improve the amortized latency.

long-distance connection is usually larger ($\geq$ 100ms) than propagating a layer ($\approx$ 11ms). To relax the limitation, we design delayed update to put off the the synchronization barrier by n steps as shown in Fig. 3b. To ensure minimal loss on accuracy, we modify the gradient update formula with error compensations.

To understand, we start with vanilla SGD: Consider a worker with index j, the weights at timestamp n during distributed training are derived via

$$w_{(n,j)} = w_{(n-1,j)} + \lambda \overline{\nabla w_{(n-1)}} = w_{(0,j)} + \lambda \sum_{i=0}^{n-1} \overline{\nabla w_{(i)}} \tag{1}$$

where $\overline{\nabla w_{(n-1)}}$ is the averaged gradients calculated by gathering all local gradients $\nabla w_{(n-1,j)}$. In SSGD, the synchronization has to finish before $(n+1)^{\text{th}}$ forwarding. In delayed update, we $delay$ the barrier to $(n+t)^{\text{th}}$ iteration and because of the delay, $\overline{\nabla w_{(n-1)}}$ is not received at timestamp n. We use local gradients to update instead.

$$w_{(n,j)} = w_{(n-1,j)} + \lambda \nabla w_{(n-1,j)} \tag{2}$$

Meanwhile, stale gradients $\overline{\nabla w_{(n-t)}}$ arrive. At timestamp $n-t$, we performed the gradient step using local gradients $\nabla w_{(n-t,j)}$, which should be the global averaged $\overline{\nabla w_{(n-t)}}$ in SSGD. We correct the mismatch caused by replacing $\nabla w_{(n-t,j)}$ with $\overline{\nabla w_{(n-t)}}$. From Eq. 1, it adds a compensation term, which is the difference of two gradients.

$$\begin{aligned} w'_{(n,j)} &= w_{(n-1,j)} + \lambda \underbrace{\nabla w_{(n-1,j)}}_{\text{local gradients}} + \lambda (\underbrace{\overline{\nabla w_{(n-t)}}}_{\text{error compensation}} - \nabla w_{(n-t,j)}) \\ &= w_{(n,j)} + \lambda (\overline{\nabla w_{(n-t)}} - \nabla w_{(n-t,j)}) \end{aligned} \tag{3}$$

Next, we consider momentum SGD. With momentum m, the weight at timestamp n is updated by

$$u_{(n,j)} = m u_{(n-1,j)} + \nabla w_{(n,j)} \quad w_{(n,j)} = w_{(n-1,j)} + \lambda u_{(n-1,j)} \tag{4}$$

We rewrite the cumulative form to general form

$$\begin{aligned} u_{(n,j)} &= \sum_{i=0}^{n} m^i \nabla w_{(n-i,j)} \\ &= \sum_{i=0}^{n} m^{n-i} \nabla w_{(i,j)} \end{aligned} \tag{5}$$

$$\begin{aligned} w_{(n,j)} &= w_{(0,j)} + \lambda \sum_{i=0}^{n-1} u_{(i,j)} \\ &= w_{(0,j)} + \lambda \sum_{i=0}^{n-1} \sum_{k=0}^{i} m^{i-k} \nabla w_{(k,j)} \\ &= w_{(0,j)} + \lambda \sum_{i=0}^{n-1} \sum_{k=0}^{n-1-i} m^{k} \nabla w_{(i,j)} \end{aligned} \tag{6}$$

Then, we can derive the compensated momentum

$$\begin{aligned} u'_{(n,j)} &= m u_{(n-1,j)} + \nabla w_{(n-1,j)} + m^t (\overline{\nabla w_{(n-t)}} - \nabla w_{(n-t,j)}) \\ &= u_{(n,j)} + m^t (\overline{\nabla w_{(n-t)}} - \nabla w_{(n-t,j)}) \end{aligned} \tag{7}$$

and weights similarly

$$\begin{aligned} w'_{(n,j)} &= w_{(n-1,j)} + \lambda u_{(n-1,j)} + \lambda \sum_{k=0}^{t-1} m^k (\overline{\nabla w_{(n-t)}} - \nabla w_{(n-t,j)}) \\ &= w_{(n,j)} + \lambda \sum_{k=0}^{t-1} m^k (\overline{\nabla w_{(n-t)}} - \nabla w_{(n-t,j)}) \end{aligned} \tag{8}$$

In original synchronous stochastic gradient descent (SSGD), every worker receives averaged gradients from $n-1$ at timestamp n. This requires collective operation (*e.g.* Allreduce) to execute quickly; otherwise, it will slow down the training process. In delayed update, the barrier of the collective operation is moved from timestamp n to $n+t-1$. It means as long as the gradients are transmitted within $t \times T_{\text{compute}}$, the training is not blocked. To put some real numbers, assume we are training ResNet-50 on Nvidia Tesla V100, each step takes around 300ms. If we delay the update by 20 steps, even with latency as large as 6 seconds, the communication still can be fully covered by computations.

3.2 Temporally Sparse Update

Delayed update provides a good solution to tolerate latency. However, there still remains two issues and network congestion is the first one. In the original SSGD, only one transmission is performed during one iteration. But when using delayed update, there can be up to t transmissions at the same time. Without a special hacking to low-level network driver, gradients sent earlier may arrive later, which is not what we want. The second one is latency variance. In real world network, especially for those long-distance connections, the latency varies in a large range. Even though the average latency is small, some accidentally high latency can slow the system.

To address, we adapt temporally sparse update to reduce the communication frequency and design corresponding error compensation to guarantee the accuracy. As demonstrated in Fig. 3c, temporally sparse update only synchronizes for every p steps. For those steps where synchronization is not performed, each worker updates its model with local gradients. For those steps where synchronization is performed, each worker calculates error compensations based averaged information. As a result, the networking congestion is alleviated and the networking latency is amortized across multiple local update steps.

For vanilla SGD, like the delayed update, we can derive the compensation term for temporally sparse update from Eq. 1, which is the difference between accumulated gradients. Note the compensation in temporally sparse update only applies when $n \pmod p \equiv 0$.

$$w'_{(n,j)} = w_{(n,j)} + \lambda (\sum_{i=n-p}^{n-1} \overline{\nabla w_{(i)}} - \sum_{i=n-p}^{n-1} \nabla w_{(i,j)}) \tag{9}$$

	Bandwidth	Non-computing time
Synchronous SGD	$2\|\|w\|\|/T_{\text{training}}$	$T_{\text{communicate}} - T_{\text{overlap}}$
Delayed Update (t)	$2\|\|w\|\|/T_{\text{training}}$	$max(0, T_{\text{communicate}} - T_{\text{overlap}} - t \times T_{\text{compute}})$
Temporal Sparsity (p)	$4\|\|w\|\|/(p \times T_{\text{training}})$	$(T_{\text{communicate}} - T_{\text{overlap}})/p$
Delayed Update (t) + Temporal Sparsity (p)	$4\|\|w\|\|/(p \times T_{\text{training}})$	$max(0, (T_{\text{communicate}} - T_{\text{overlap}} - t \times T_{\text{compute}}))/p$

Table 1: The networking requirements for different training strategies.

For SGD with momentum acceleration, based on accumulated gradients is not enough to compensate both $\nabla w_{(n,j)}$ and $u_{(n,j)}$. We need to transmit an extra term to correct the momentum:

$$\begin{aligned} u'_{(n,j)} &= u_{(n,j)} + (\sum_{i=n-p}^{n-1} m^{n-i}\overline{\nabla w_{(i)}} - \sum_{i=n-p}^{n-1} m^{n-i}\nabla w_{(i,j)}) \\ w'_{(n,j)} &= w_{(n,j)} + \lambda(\sum_{i=n-p}^{n-1}\sum_{k=0}^{n-1-i} m^{k}\overline{\nabla w_{(i)}} - \sum_{i=n-p}^{n-1}\sum_{k=0}^{n-1-i} m^{k}\nabla w_{(i,j)}) \\ &= w_{(n,j)} + \lambda(\sum_{i=n-p}^{n-1}\sum_{k=n-p}^{i} m^{i-k}\overline{\nabla w_{(k)}} - \sum_{i=n-p}^{n-1}\sum_{k=n-p}^{i} m^{i-k}\nabla w_{(k,j)}) \end{aligned} \tag{10}$$

Let $S_{(i,j)} = \sum_{k=n-p}^{i} m^{i-k}\nabla w_{(k,j)}$. Note $S_{(i+1,j)} = mS_{(i,j)} + \nabla w_{(i+1,j)}$ and $S_{(n-p+1,j)} = \nabla w_{(n-p,j)}$, the formula can be written as:

$$\begin{aligned} u'_{(n,j)} &= u_{(n,j)} + (\overline{S_{(n)}} - S_{(n,j)}) \\ w'_{(n,j)} &= w_{(n,j)} + \lambda(\sum_{i=n-p}^{n-1}\overline{S_{(i)}} - \sum_{i=n-p}^{n-1} S_{(i,j)}) \end{aligned} \tag{11}$$

Note temporally sparse update is different from gradient accumulation (forward and backward N times, accumulate the gradients and optimizer steps only 1 time). Gradient accumulation is equivalent to increasing the batch size and requires to adjust learning rate and the parameters of batch normalization to ensure smooth convergence. Temporally sparsity updates model every iteration and compute error compensation every N steps. It does not require any changes to the hyper-parameters.

3.3 Delayed and temporally sparse update

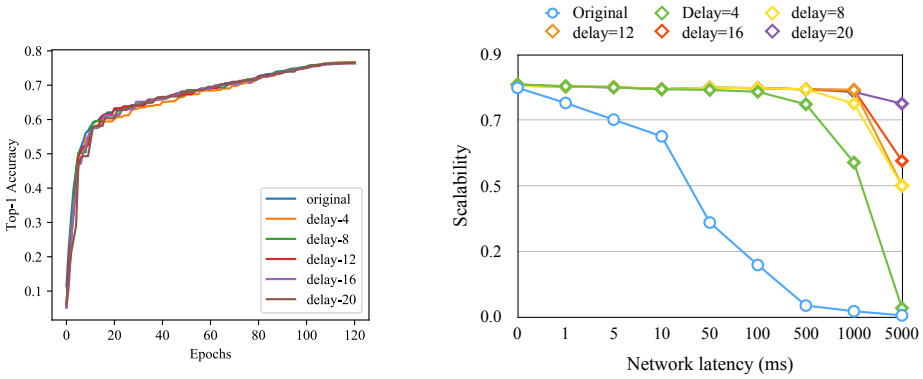
The delayed technique and temporally sparse update can be combined together. The compensation formulas for vanilla SGD is

$$w'_{(n,j)} = w_{(n,j)} + \lambda(\sum_{i=n-t-p}^{n-t-1}\overline{\nabla w_{(i)}} - \sum_{i=n-t-p}^{n-t-1}\nabla w_{(i,j)}) \tag{12}$$

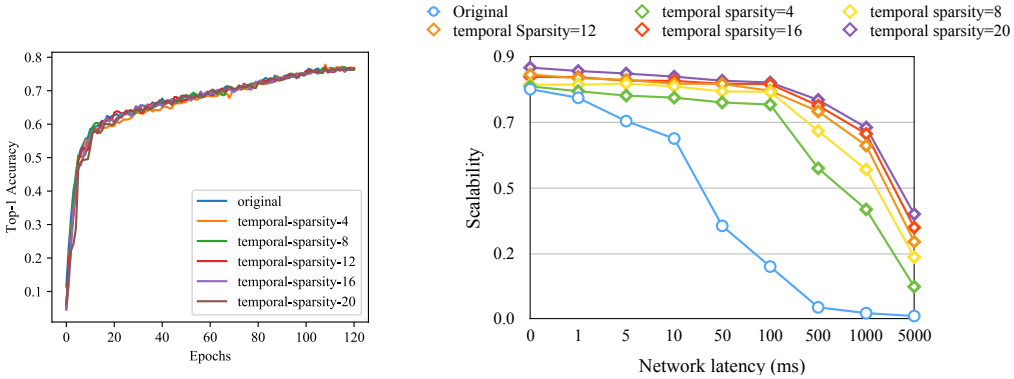
and for momentum SGD is

$$\begin{aligned} u'_{(n,j)} &= u_{(n,j)} + m^{t}(\overline{S_{n-t}} - S_{(n-t,j)}) \\ w'_{(n,j)} &= w_{(n,j)} + \lambda m^{t}(\sum_{i=n-t-p}^{n-t-1}\overline{S_i} - \sum_{i=n-t-p}^{n-t-1} S_{(i,j)}) \end{aligned} \tag{13}$$

Original ring allreduce transfers $2||w||$ packets (Thakur et al., 2005). Though temporally sparse update doubles data transferred, the temporal sparsity t is always larger than 2. As a consequence, combing two methods can further improve scalability under challenging networking as compared in Tab. 1.



(a) The learning curve with different delay steps and scalability under various latency.



(b) The learning curve with different temporal sparsity and scalability under various latency.

Figure 4: Compare the top-1 accuracy and scalability on ImageNet training.

4 Experiment

4.1 Experiment settings

We evaluate our DTS on image classification task. We studies ResNet-50 (He et al., 2016) on ImageNet (Deng et al., 2009). ImageNet contains 1.2 million training images and 50,000 validation images in 1000 classes. 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 random crop and flip are adapted. We set the initial learning rate 0.0125 for which grows linearly w.r.t number of GPUs. For a fair comparison, the training settings are set the same among all experiments.

When performing ablation studies under different latency, we synthesis the network condition using *netem*[†]. When training across the world, the limited network is benchmarked using *qperf*[‡]. We implement DST in PyTorch framework (Paszke et al., 2017) and adapt Horovod (Sergeev & Del Balso, 2018) as the distributed training backend. To make results reproducible, we conduct our experiment on standard Amazon *p.large* EC2 instances with Nvidia Tesla V100 and will release the codebase when less anonymous.

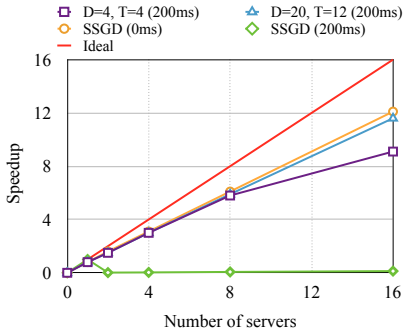
4.2 Results of Delayed Update

Fig. 4a shows the performance and scalability for delayed update with different latency. We start from delay steps 4 and gradually increase it. The learning curve shows that with our proposed error compensation delayed update demonstrates robust convergence and comparable performance compared vanilla SSGD, even when delay interval is as large as 20.

[†] https://wiki.linuxfoundation.org/networking/netem
[‡] https://linux.die.net/man/1/qperf

	Top-1%
Original SGD	76.63
D=4, T=4, C=1%	76.15
D=8, T=8, C=1%	76.32
D=12, T=8, C=1%	76.18
D=20, T=12, C=1%	75.81

(a) Delayed updates preserves the accuracy while tolerate a large latency. D, T, C indicate delayed update, temporal sparsity and gradient compression ratio respectively. The results are evaluated on ImageNet dataset.



(b) Better speedup and scalability with DTS. Measured on servers across the world.

Table 2: DTS demonstrates good scalability and accuracy on world-wide high latency training.

In original SSGD and modern deep learning frameworks, though the transmission is partially pipelined with backpropogration: Synchronizing gradients of n^{th} layer can be performed simultaneously with back-propagating $(n-1)^{\text{th}}$ layer, it is not enough to fully cover communication since latency on long distance connection is usually larger ($\geq$ 100ms) than propagating a layer (11ms). By delaying the synchronization, delayed update provides much more spare time for transmission and thus scales better. As shown in Fig. 4a, the larger the delayed interval is, the better tolerance towards latency. The scalability keeps stable until maximum tolerance is exceeded.

4.3 Results of Temporally Sparse Update

The connection of real-world network has many bothering issues such as unstable latency and congestion on routers. Temporally sparse is introduced to amortize these via reducing the synchronization frequency. The results are shown in Fig. 4b(*Left*): we start with temporal sparsity 4 and increase to 20. With proposed compensation, the temporally sparse update has a negligible loss on accuracy. Fig. 4(*Right*) exhibits how temporally sparse training scales under different network latency. The scalability degrades slower than because temporally sparse update amortizes the latency.

4.4 Distributed Training across the World

With two powerful techniques, we are close to our original target of distributed training across world. To evaluate DTS, we deploy located at four different locations across the world: Tokyo(Japan), London (U.K), Oregon (U.S.A) and Ohio(U.S.A). The latency information is shown in Fig.2. The real latency across the ring allreduce 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. In terms of latency variance and network congestion, we integrate temporally sparse update to alleviate which drastically reduces the bandwidth requirement by t times. However, there is still a bottleneck on bandwidth for cross-continent connections (*e.g.* Tokyo to London). Though theoretically bandwidth is achievable by switching to better internet service providers, AWS does not provide such an option. As an economical alternative, we adapt deep gradient compression (Lin et al., 2017) to work through. As shown in Tab. 2a, DTS has little effect on training accuracy. Meanwhile, DTS demonstrates strong scalability (0.72) under high latency (200ms between workers) and this performance is close to what conventional algorithms achieved inside a data center (speedup ratio 0.78 and 1us latency).

5 Conclusion

In this paper, we propose two novel delayed update and temporally sparse update methods to scale synchronous distributed training on servers located at different continents across the world. We demonstrate that delayed synchronization and temporally sparse update are both accuracy preserving while tolerate poor network conditions. We believe that our work will open future avenues for a wide range of decentralized learning applications.

REFERENCES

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL http://tensorflow.org/. Software available from tensorflow.org.

Alham Fikri Aji and Kenneth Heafield. Sparse communication for distributed gradient descent. *arXiv preprint arXiv:1704.05021*, 2017.

Takuya Akiba, Keisuke Fukuda, and Shuji Suzuki. ChainerMN: Scalable Distributed Deep Learning Framework. In *Proceedings of Workshop on ML Systems in The Thirty-first Annual Conference on Neural Information Processing Systems (NIPS)*, 2017. URL http://learningsys.org/nips17/assets/papers/paper_25.pdf.

Dan Alistarh, Jerry Li, Ryota Tomioka, and Milan Vojnovic. Qsgd: Randomized quantization for communication-optimal stochastic gradient descent. *arXiv preprint arXiv:1610.02132*, 2016.

Dan Alistarh, Torsten Hoefler, Mikael Johansson, Nikola Konstantinov, Sarit Khirirat, and Cédric Renggli. The convergence of sparsified gradient methods. In *Advances in Neural Information Processing Systems*, pp. 5973–5983, 2018.

Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konecny, Stefano Mazzocchi, H Brendan McMahan, et al. Towards federated learning at scale: System design. *arXiv preprint arXiv:1902.01046*, 2019.

Chia-Yu Chen, Jungwook Choi, Daniel Brand, Ankur Agrawal, Wei Zhang, and Kailash Gopalakrishnan. Adacomp: Adaptive residual gradient compression for data-parallel distributed training. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

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

Trishul M Chilimbi, Yutaka Suzue, Johnson Apacible, and Karthik Kalyanaraman. Project adam: Building an efficient and scalable deep learning training system. In *OSDI*, volume 14, pp. 571–582, 2014.

Christopher M De Sa, Ce Zhang, Kunle Olukotun, and Christopher Ré. Taming the wild: A unified analysis of hogwild-style algorithms. In *Advances in neural information processing systems*, pp. 2674–2682, 2015.

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

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

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

Google. Towards federated learning at scale: System design. 2017. URL https://ai.googleblog.com/2017/04/federated-learning-collaborative.html.

Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

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

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

Peng Jiang and Gagan Agrawal. A linear speedup analysis of distributed deep learning with sparse and quantized communication. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 31*, pp. 2525–2536. Curran Associates, Inc., 2018. URL http://papers.nips.cc/paper/7519-a-linear-speedup-analysis-of-distributed-deep-learning-with-sparse-and-quanti pdf.

Arthur Jochems, Timo M Deist, Johan Van Soest, Michael Eble, Paul Bulens, Philippe Coucke, Wim Dries, Philippe Lambin, and Andre Dekker. 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.

Arthur Jochems, Timo M Deist, Issam El Naqa, Marc Kessler, Chuck Mayo, Jackson Reeves, Shruti Jolly, Martha Matuszak, Randall Ten Haken, Johan van Soest, 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):344–352, 2017.

Jakub Konen, H. Brendan McMahan, Felix X. Yu, Peter Richtarik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. In *NIPS Workshop on Private Multi-Party Machine Learning*, 2016. URL https://arxiv.org/abs/1610.05492.

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

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

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

Xiangru Lian, Wei Zhang, Ce Zhang, and Ji Liu. Asynchronous decentralized parallel stochastic gradient descent. *arXiv preprint arXiv:1710.06952*, 2017.

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

Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

H Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, et al. Communication-efficient learning of deep networks from decentralized data. *arXiv preprint arXiv:1602.05629*, 2016.

Philipp Moritz, Robert Nishihara, Ion Stoica, and Michael I Jordan. Sparknet: Training deep networks in spark. *arXiv preprint arXiv:1511.06051*, 2015.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

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

Felix Sattler, Simon Wiedemann, Klaus-Robert Müller, and Wojciech Samek. Sparse binary compression: Towards distributed deep learning with minimal communication. *CoRR*, abs/1805.08768, 2018. URL http://arxiv.org/abs/1805.08768.

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

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

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Nikko Strom. Scalable distributed dnn training using commodity gpu cloud computing. In *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

Peng Sun, Wansen Feng, Ruobing Han, Shengen Yan, and Yonggang Wen. Optimizing network performance for distributed dnn training on gpu clusters: Imagenet/alexnet training in 1.5 minutes. *arXiv preprint arXiv:1902.06855*, 2019.

Hanlin Tang, Shaoduo Gan, Ce Zhang, Tong Zhang, and Ji Liu. Communication compression for decentralized training. In *Advances in Neural Information Processing Systems*, pp. 7652–7662, 2018a.

Hanlin Tang, Xiangru Lian, Ming Yan, Ce Zhang, and Ji Liu. D^2: Decentralized training over decentralized data. *CoRR*, abs/1803.07068, 2018b. URL http://arxiv.org/abs/1803.07068.

Hanlin Tang, Xiangru Lian, Tong Zhang, and Ji Liu. Doublesqueeze: Parallel stochastic gradient descent with double-pass error-compensated compression. In *International Conference on Machine Learning*, 2019.

Rajeev Thakur, Rolf Rabenseifner, and William Gropp. Optimization of collective communication operations in mpich. *The International Journal of High Performance Computing Applications*, 19 (1):49–66, 2005.

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

Wei Wen, Cong Xu, Feng Yan, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Terngrad: Ternary gradients to reduce communication in distributed deep learning. In *Advances in neural information processing systems*, pp. 1509–1519, 2017.

Eric P Xing, Qirong Ho, Wei Dai, Jin Kyu Kim, Jinliang Wei, Seunghak Lee, Xun Zheng, Pengtao Xie, Abhimanu Kumar, and Yaoliang Yu. Petuum: A new platform for distributed machine learning on big data. *IEEE Transactions on Big Data*, 1(2):49–67, 2015.