A Highly Efficient Distributed Deep Learning System For Automatic Speech Recognition

Wei Zhang, Xiaodong Cui, Ulrich Finkler, George Saon, Abdullah Kayi, Alper Buyuktosunoglu,
Brian Kingsbury, David Kung, Michael Picheny

IBM Research

{weiz,cuix,ufinkler,gsaon,kayi,alperb,bedk,kung,picheny}@us.ibm.com

Abstract

Modern Automatic Speech Recognition (ASR) systems rely on distributed deep learning to for quick training completion. To enable efficient distributed training, it is imperative that the training algorithms can converge with a large mini-batch size. In this work, we discovered that Asynchronous Decentralized Parallel Stochastic Gradient Descent (ADPSGD) can work with much larger batch size than commonly used Synchronous SGD (SSGD) algorithm. On commonly used public SWB-300 and SWB-2000 ASR datasets, ADPSGD can converge with a batch size 3X as large as the one used in SSGD, thus enable training at a much larger scale. Further, we proposed a Hierarchical-ADPSGD (H-ADPSGD) system in which learners on the same computing node construct a super learner via a fast allreduce implementation, and super learners deploy ADPSGD algorithm among themselves. On a 64 Nvidia V100 GPU cluster connected via a 100Gb/s Ethernet network, our system is able to train SWB-2000 to reach a 7.6% WER on the Hub5-2000 Switchboard (SWB) test-set and a 13.2% WER on the Callhome (CH) test-set in 5.2 hours. To the best of our knowledge, this is the fastest ASR training system that attains this level of model accuracy for SWB-2000 task to be ever reported in the literature.

Index Terms: speech recognition, distributed systems, deep learning

1. Introduction

Deep Learning (DL) drives the current Automatic Speech Recognition (ASR) systems and has yielded models of unprecedented accuracy [1, 2]. Stochastic Gradients Descent (SGD) and its variants are the de facto learning algorithms deployed in DL training systems. Distributed Deep Learning (DDL), which deploys different variants of parallel SGD algorithms in its core, is known today as the most effective method to enable fast DL model training. Several DDL algorithms exist – most notably, Synchronous SGD (SSGD) [3], parameter-server based Asynchronous SGD (ASGD) [4], and Asynchronous Decentralized Parallel SGD (ADPSGD) [5]. Among them, ASGD has lost its favor among practitioners due to its poor convergence behavior [6, 3, 7, 8]. In our previous work [9], we systematically studied the application of state-of-the-art SSGD (SSGD) and ADPSGD to challenging SWB-2000 tasks. To date, it is commonly believed that ADPSGD and SSGD converge with similar batch sizes [5]. In this paper, we find ADPSGD can potentially smoothen objective function landscape and work with much larger batch sizes than SSGD. As a result of this finding, we made the following contributions in this paper: (1) We systematically studied the convergence behavior of SSGD and ADPSGD on two public ASR datasets – SWB-300 and SWB-2000– with a state-of-the-art LSTM model and confirmed that

ADPSGD enables distributed training on a much larger scale. To the best of our knowledge, this is the first demonstration conducted on large-scale public datasets that an asynchronous system can scale with a larger batch size than SSGD. (2) To reduce system staleness, improve system scalability w.r.t number of learners and enable better communication and computation efficiency, we implemented a Hierarchical-ADPSGD (H-ADPSGD) system in which learners on the same computing node construct a "super-learner" via a fast allreduce[1] implementation and the super-learners communicate with each other in the ADPSGD fashion. Our system shortens the SWB-2000 training from over 8 days on one V100 gpu to 5.2 hours on 64 gpus (i.e., 40X speedup), and the resulting model reaches 7.6% WER on SWB and 13.2% on CH. To the best of our knowledge, this is the fastest system for state-of-the-art SWB-2000 model training ever reported in the literature.

2. Background and Problem Formulation

Consider the following stochastic optimization problem

$$\min_{\theta} F(\theta) = \mathbb{E}_{\xi}[f(\theta; \xi)] \tag{1}$$

where F is the objective function, θ is the parameters to be optimized (it is the weights of networks for DL) and $\xi \sim p(x)$ is a random variable on the training data x obeying distribution $p(x)$. Supposing that there are n training samples and ξ assumes a uniform distribution, $\xi \sim \text{Uniform}\{1, \cdots, n\}$, we have

$$\min_{\theta} F(\theta) = \mathbb{E}_{\xi}[f(\theta; \xi)] = \frac{1}{n}\sum_{i=1}^{n} f_i(\theta) \tag{2}$$

where $f_i(\theta)$ is evaluated at training sample x_i. In mini-batch based SGD, at each iteration k, we have

$$F(\theta_{k+1}) = F(\theta_k) - \alpha_k g(\theta_k, \xi_k) \tag{3}$$

where α_k is the learning rate and

$$g(\theta_k, \xi_k) \triangleq \frac{1}{m}\sum_{s=1}^{m} \nabla f(\theta_k, \xi_{k,s}) \tag{4}$$

with m being the size of the mini-batch.

DL problem is an optimization problem described above. DDL is the distributed computing paradigm that solves DL problems. At the dawn of DDL research, the Parameter Server (PS) architecture [4] was wildy popular. Figure 1(a) illustrates a PS design: each learner calculates gradients and sends them to the PS, before the PS sends the updated weights back to

---

[1]Allreduce is a broadcast operation followed by a reduction operation (e.g., summation).

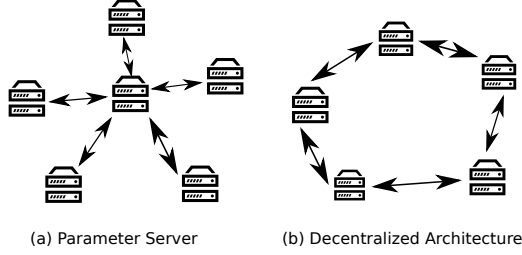(a) Parameter Server   (b) Decentralized Architecture

Fig. 1: *A centralized distributed learning architecture (aka Parameter-Server architecture) and a decentralized distributed learning architecture*

the learners. The core of PS architecture is the ASGD algorithm that allows each learner to calculate gradients and asynchronously push/pull the gradients/weights to/from the PS. The weight update rule in ASGD is given in Equation (5):

$$F(\theta_{k+1}) = F(\theta_k) - \alpha_k g(\theta_k^{\zeta_i}, \xi_k^{\zeta_i}) \qquad (5)$$

where $\zeta_i \in \{1, \cdots, \lambda\}$ is an indicator of the learner which made the model update at iteration k. Soon researchers realized that such design led to sub-par models that can be trained fast but accuracy-wise significantly lag behind single-learner baseline because of the large system staleness issue [10, 6, 8]. SSGD regained its popularity mainly because the synchronization problems exacerbated by the cheap commodity systems could be addressed by decades of High Performance Computing (HPC) research. In SSGD, the weight update rule is given in Equation (6) : each learner calculates gradients and receives updated weights in lockstep with the others.[2] SSGD has exactly the same convergence rate as the single learner baseline while enjoying reasonable speedup.

$$F(\theta_{k+1}) = F(\theta_k) - \alpha_k \left[\frac{1}{\lambda} \sum_{i=1}^{\lambda} g(\theta_k^i, \xi_k^i) \right] \qquad (6)$$

The summation and the following broadcast in Equation (6) is known as the "All-Reduce" operation in the HPC, which is a well-studied operation. When the message to be "AllReduced" is large, as in the DL case, an optimal algorithm exists that maximizes the communication bandwidth utilization [11]. Many incarnations of this algorithm exist, such as [12, 13, 14]. Like any synchronous algorithm, SSGD is subject to the *straggler problem*, which means a slow learner slows down the entire training system.

Wildfire [15] is the first decentralized training system that has been demonstrated to work on modern deep learning tasks. The follow-up work in [16, 5] rigorously proved the decentralized parallel SGD algorithm in both synchronous and asynchronous forms can maintain the SSGD convergence rate while resolving the straggler problem. Figure 1(b) illustrates a decentralized parallel SGD system , where each learner i calculates the gradients, updates its weights, and averages its weights with its neighbor j in a ring topology. DPSGD/ADPSGD weights update rule is defined in Equation (7).

$$\Theta_{k+1} = \Theta_k T_k - \alpha_k g(\Theta_k, \Xi_k) \qquad (7)$$

---

[2]Throughout this paper, we use λ to denote the number of learners.

where the columns of $\Theta_k = [\theta_k^1, \cdots, \theta_k^\lambda]$ are the weights of each learner; the columns of $\Xi_k = [\xi_k^1, \cdots, \xi_k^\lambda]$ are random variables associated with batch samples; the columns of $g(\Theta_k, \Xi_k) = [g(\theta_k^1, \xi_k^1), \cdots, g(\theta_k^\lambda, \xi_k^\lambda)]$ are the gradients of each learner, and T_k is a symmetric stochastic matrix (therefore doubly stochastic) with two 0.5s at neighboring positions for each row and column and 0s everywhere else. This amounts to performing model averaging with one learner's left or right neighbor in the ring for each mini-batch weights update. The key difference between DPSGD and ADPSGD is that ADPSGD allows overlapping of gradients calculation and weights averaging and removes the global barrier in DPSGD that forces every learner to synchronize at the end of each minibatch training. Naturally, ADPSGD runs much faster than DPSGD.

3. Scalability of ADPSGD

It is well known that when batch-size is increased, it is difficult for a DDL system to maintain model accuracy [3, 8]. In our previous work [9], we designed a principled method to increase batch size while maintaining model accuracy with respect to training epochs for both SSGD and ADPSGD on ASR tasks, up to batch size 2560. The key technique is as follows: (1) learning rate linear scaling by the ratio between effective batch size and baseline batch size. (2) linearly warmup the learning rate for the first 10 epochs before annealing at a rate $\frac{1}{\sqrt{(2)}}$ per epoch for the remaining epochs. For example, if the baseline batch size is 256 and learning rate is 0.1 and assume now the total batch size is 2560, then for the first 10 epochs, the learning rate increases by 0.1 every epoch before reaching 1.0 at the 10th epoch. We demonstrated that SSGD and ADPSGD can both scale up to the batch size of 2560 on ASR tasks in [9]. In this paper, we adopt the same hyper-parameter setup and study the convergence behavior for SSGD and ADPSGD under larger batch sizes. Figure 2a and Figure 2b plot the training loss and held-out loss w.r.t epochs for SSGD (in red) and ADPSGD (in green) under different batch sizes on SWB-300. The single-learner baseline with batch size 256 is plotted in black. ADPSGD can scale with a batch size 3x larger than SSGD (i.e., 8192 vs 2560). Figure 2c and Figure 2d show a similar trend for SWB-2000– SSGD only scales up to batch size 4096 whereas ADPSGD scales up to batch size 12288. To make SSGD converge with the same batch size as ADPSGD, one may use a much less aggressive learning rate (e.g. 4x smalle learning rate), which will take many more epochs to converge to the same level of accuracy.

We speculate ADPSGD can scale with a larger batch size than SSGD for the following reason: ADPSGD is carried out with local gradient computation and update on each learner (second term on the RHS of Equation (7)) and global model averaging (first term of the RHS of Equation (7)) across the ring. The locally updated information is transferred to other learners in the ring by performing model averaging through neighboring learners. If we halt the local updates and only conduct the model averaging through the doubly stochastic matrix T_k, the system will converge to the equilibrium where all learners share the identical weights, which is $\frac{1}{n} \sum_{i=1}^{n} \theta_k^i$. After the convergence, each learner pulls the weights, computes the gradients, updates the model and conducts another round of model averaging to convergence. This amounts to synchronous SGD. Therefore, ADPSGD has synchronous SGD as its special case. However, in ADPSGD, local updates and model averaging run simultaneously. It is speculated that the local updates are carried out on $\frac{1}{n} \sum_{i=1}^{n} \theta_k^i$ perturbed by a noise term. This may

(a) *Training loss comparison between SSGD and ADPSGD, on SWB300 data-set*

(b) *Heldout loss comparison between SSGD and ADPSGD, on SWB300 dataset*

(c) *Training loss comparison between SSGD and ADPSGD on SWB2000 data-set*

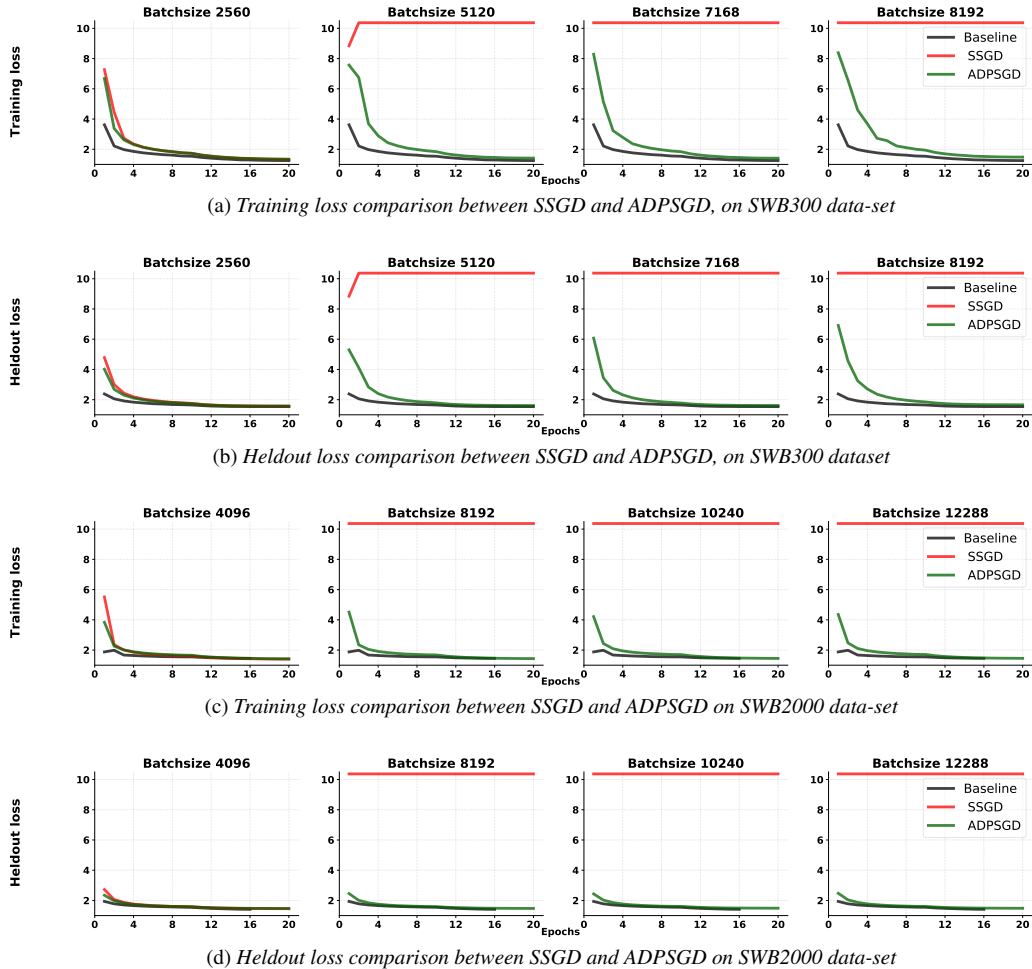(d) *Heldout loss comparison between SSGD and ADPSGD on SWB2000 data-set*

Fig. 2: *Convergence comparison between SSGD and ADPSGD on SWB-300 and SWB-2000 data-set. Baseline is plotted in black, SSGD in red, ADPSGD in green. For SWB-300, ADPSGD converges similarly to the baseline up to batch size 8192, whereas SSGD can only scale up to batch size 2560. Likewise, for SWB-2000, ADPSGD converges similarly to the baseline up to batch size 12288, whereas SSGD can only scale up to batch size 4096. For all SSGD and ADPSGD experiments, we use 16 learners.*

give opportunities to use a larger batch size that is not possible for synchronous SGD.

4. Design of Hierarchical ADPSGD

In practice, ADPSGD sees signficant accuracy drop when scaling over more than 16 learners on ASR tasks [5, 9] due to system staleness issue. We built H-ADPSGD, which is a hierarchical system as depicted in Figure 3, to address the staleness issue. N learners constructs a super-learner, which applies the weight update rule as in Equation (6). The staleness across learners in this super-learner is effectively 0. Each super-learner then participates in the ADPSGD ring and update its weights using Equation (7). Further, assume there are a total of λ learners; then *the convergence behavior of this λ-learner H-ADPSGD system is equivalent to that of a $\frac{\lambda}{N}$-learner ADPSGD with the same total batch size*. This design ensures H-ADPSGD scales with N more learners than its ADPSGD counterpart, while maintaining the same level of model accuracy. Moreover, H-ADPSGD improves communication efficiency and reduces main memory traffic and CPU pressure across super-learners.
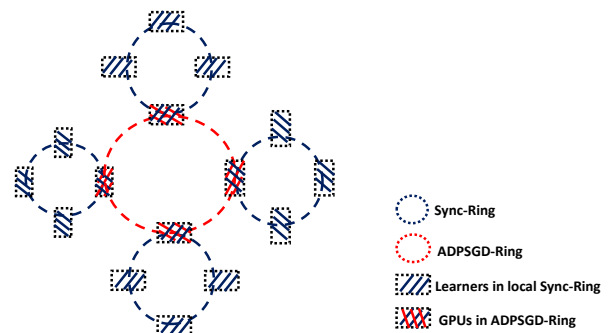

Fig. 3: *System architecture for H-ADPSGD. Learners, on the same computing node, constitute a super-learner via a synchronous ring (blue). The super-learners constructs the ADPSGD ring (red).*

Table 1: *WER comparison between baseline, SSGD, and ADPSGD for different batch sizes. –: not converged, BS: batch size. (H-)ADPSGD scales with 3X larger batch size than SSGD. ADPSGD scales up to 16 GPUs, H-ADPSGD scales up to 64 gpus.*

	BS256	BS4096		BS8192		BS10240		BS12288	
	Baseline	SSGD	(H-)ADPSGD	SSGD	(H-)ADPSGD	SSGD	(H-)ADPSGD	SSGD	(H-)ADPSGD
SWB	7.5	7.6	7.5	–	7.6	–	7.5	–	7.8
CH	13.0	13.1	13.2	–	13.2	–	13.5	–	13.6

5. Methodology

5.1. Software and Hardware

PyTorch 0.4.1 is our DL framework. Our communication library is built with CUDA 9.2 compiler, the CUDA-aware Open-MPI 3.1.1, and g++ 4.8.5 compiler. We run our experiments on a 64-GPU 8-server cluster. Each server has 2 sockets and 9 cores per socket. Each core is an Intel Xeon E5-2697 2.3GHz processor. Each server is equipped with 1TB main memory and 8 V100 GPUs. Between servers are 100Gbit/s Ethernet connections. GPUs and CPUs are connected via PCIe Gen3 bus, which has a 16GB/s peak bandwidth in each direction per socket.

5.2. DL Models and Dataset

The acoustic model is an LSTM with 6 bi-directional layers. Each layer contains 1,024 cells (512 cells in each direction). On top of the LSTM layers, there is a linear projection layer with 256 hidden units, followed by a softmax output layer with 32,000 units corresponding to context-dependent HMM states. The LSTM is unrolled with 21 frames and trained with non-overlapping feature subsequences of that length. The feature input is a fusion of FMLLR (40-dim), i-Vector (100-dim), and logmel with its delta and double delta (40-dim $\times$3). This model contains over 43 million parameters and is about 165MB large.

The language model (LM) is built using publicly available training data, including Switchboard, Fisher, Gigaword, and Broadcast News, and Conversations. Its vocabulary has 85K words, and it has 36M 4-grams. The two training datasets used are SWB-300, which contains over 300 hours of training data and has a capacity of 30GB, and SWB-2000, which contains over 2000 hours of training data and has a capacity of 216GB. The two training data-sets are stored in HDF5 data format. The test set is the Hub5 2000 evaluation set, composed of two parts: 2.1 hours of switchboard (SWB) data and 1.6 hours of call-home (CH) data.

6. Experimental Results

6.1. Convergence Results

Table 1 records the WER of SWB-2000 models trained by SSGD and ADPSGD under different batch sizes. Single-gpu training baseline is also given as a reference. *ADPSGD can converge with a batch size 3x larger than that of SSGD, while maintaining model accuracy.*

6.2. Speedup

Figure 4 shows the H-ADPSGD speedup. Using a batch size 128 per gpu, our system achieves 40X speedup over 64 gpus. It takes 203 hours to finish SWB-2000 training in 16 epochs on one V100 GPU. *It takes H-ADPSGD 5.2 hours to train for 16 epochs to reach WER 7.6% on SWB and WER 13.2% on CH, on 64 gpus.*
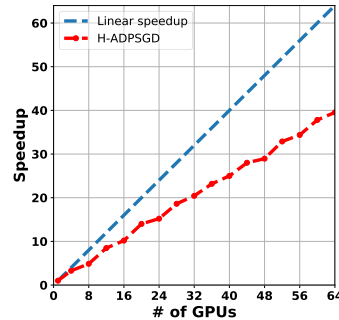

Fig. 4: *Speedup up to 64 gpus. Batch-size per gpu is 128.*

7. Conclusion and Future Work

In this work, we made the following contributions: (1) We discovered that ADPSGD can scale with much larger batch sizes than the commonly used SSGD algorithm for ASR tasks. To the best of our knowledge, this is the first asynchronous system that scales with larger batch sizes than a synchronous system for public large-scale DL tasks. (2) To make ADPSGD more scalable w.r.t number of learners, we designed a new algorithm H-ADPSGD, which trains a SWB-2000 model to reach WER 7.6% on SWB and WER 13.2% on CH in 5.2 hours, on 64 Nvidia V100 GPUs. To the best of our knowledge, this is the fastest system that trains SWB-2000 to this level of accuracy. We plan to investigating the theoretical justification for why ADPSGD scales with a larger batch size than SSGD in our future work.

8. Related Work

DDL systems enable many AI applications with unprecedented accuracy, such as speech recognition [7, 17], computer vision [3], language modeling [18], and machine translation [19]. During the early days of DDL system research, researchers could only rely on loosely-coupled inexpensive computing systems and adopted PS-based ASGD algorithm [4]. ASGD has inferior model performance when learners are many, and SSGD has now become a mainstream DL training method [6, 3, 7, 8, 20]. Consequently, these systems encounter the classical straggler problem in distributed system design. Recently, researchers proposed ADPSGD, which is proved to have the same convergence rate as SSGD while completely eliminating the straggler problem [5]. Previous work [9] applied ADPSGD to the challenging SWB2000-LSTM task to achieve state-of-the-art model accuracy in a record time (11.5 hours). In this work, we discovered that ADPSGD can scale with much larger batch sizes than SSGD and designed a hierarchical ADPSGD training system that further improves training efficiency, with the resulting system improving training speed over [9] by over 2X.

9. References

[1] G. Saon, G. Kurata, T. Sercu, K. Audhkhasi, S. Thomas, D. Dimitriadis, X. Cui, B. Ramabhadran, M. Picheny, L.-L. Lim, B. Roomi, and P. Hall, "English conversational telephone speech recognition by humans and machines," in *Interspeech*, 2017.

[2] W. Xiong, J. Droppo, X. Huang, F. Seide, M. L. Seltzer, A. Stolcke, D. Yu, and G. Zweig, "Toward human parity in conversational speech recognition," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 25, no. 12, pp. 2410–2423, Dec 2017.

[3] P. Goyal, P. Dollár, R. B. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, large minibatch SGD: training imagenet in 1 hour," *CoRR*, vol. abs/1706.02677, 2017. [Online]. Available: http://arxiv.org/abs/1706.02677

[4] J. Dean, G. S. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, and A. Y. Ng, "Large scale distributed deep networks," in *NIPS*, 2012.

[5] X. Lian, W. Zhang, C. Zhang, and J. Liu, "Asynchronous decentralized parallel stochastic gradient descent," in *ICML*, 2018.

[6] J. Chen, R. Monga, S. Bengio, and R. Jozefowicz, "Revisiting distributed synchronous sgd," in *International Conference on Learning Representations Workshop Track*, 2016. [Online]. Available: https://arxiv.org/abs/1604.00981

[7] D. Amodei(et.al.), "Deep speech 2 : End-to-end speech recognition in english and mandarin," in *ICML'16*. PMLR, 2016, pp. 173–182. [Online]. Available: http://proceedings.mlr.press/v48/amodei16.html

[8] W. Zhang, S. Gupta, and F. Wang, "Model accuracy and runtime tradeoff in distributed deep learning: A systematic study," in *IEEE International Conference on Data Mining*, 2016.

[9] W. Zhang, X. Cui, U. Finkler, B. Kingsbury, G. Saon, D. Kung, and M. Picheny, "Distributed deep learning strategies for automatic speech recognition," in *ICASSP'2019*, May 2019.

[10] W. Zhang, S. Gupta, X. Lian, and J. Liu, "Staleness-aware async-sgd for distributed deep learning," in *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, 2016, pp. 2350–2356.

[11] P. Patarasuk and X. Yuan, "Bandwidth optimal all-reduce algorithms for clusters of workstations," *J. Parallel Distrib. Comput.*, vol. 69, pp. 117–124, 2009.

[12] Baidu, *Effectively Scaling Deep Learning Frameworks*, available at https://github.com/baidu-research/baidu-allreduce.

[13] Nvidia, *NCCL: Optimized primitives for collective multi-GPU communication*, available at https://github.com/NVIDIA/nccl. [Online]. Available: https://github.com/NVIDIA/nccl

[14] M. Cho, U. Finkler, S. Kumar, D. S. Kung, V. Saxena, and D. Sreedhar, "Powerai DDL," *CoRR*, vol. abs/1708.02188, 2017. [Online]. Available: http://arxiv.org/abs/1708.02188

[15] R. Nair and S. Gupta, "Wildfire: Approximate synchronization of parameters in distributed deep learning," *IBM Journal of Research and Development*, vol. 61, no. 4/5, pp. 7:1–7:9, July 2017.

[16] X. Lian, C. Zhang, H. Zhang, C.-J. Hsieh, W. Zhang, and J. Liu, "Can decentralized algorithms outperform centralized algorithms? A case study for decentralized parallel stochastic gradient descent," in *NIPS*, 2017.

[17] K. Chen and Q. Huo, "Scalable training of deep learning machines by incremental block training with intra-block parallel optimization and blockwise model-update filtering," in *ICASSP'2016*, March 2016.

[18] R. Puri, R. Kirby, N. Yakovenko, and B. Catanzaro, "Large scale language modeling: Converging on 40gb of text in four hours," *CoRR*, vol. abs/1808.01371, 2018.

[19] M. Ott, S. Edunov, D. Grangier, and M. Auli, "Scaling neural machine translation," *EMNLP 2018 THIRD CONFERENCE ON MACHINE TRANSLATION*, vol. abs/1806.00187, 2018.

[20] W. Zhang, M. Feng, Y. Zheng, Y. Ren, Y. Wang, J. Liu, P. Liu, B. Xiang, L. Zhang, B. Zhou, and F. Wang, "Gadei: On scale-up training as a service for deep learning." The IEEE International Conference on Data Mining series(ICDM'2017), 2017.