Gradient Staleness in Asynchronous Optimization Under Random Communication Delays

Haider Al-Lawati and Stark C. Draper

***Abstract*—Distributed optimization is widely used to solve large-scale optimization problems by parallelizing gradient-based algorithms across multiple computing nodes. In asynchronous optimization, the optimization parameter is updated using stale gradients, which are gradients calculated with respect to out-of-date parameters. Although large degrees of staleness can slow convergence, little is known about the impact of staleness and its relation to other system parameters. In this work, we study and analyze centralized asynchronous optimization. We show that the process of gradient arrival to the master node is similar in nature to a renewal process. We derive bounds on expected staleness and show its connection to other system parameters such as the number of workers, expected compute time and communication delays. Our derivations can be used in existing convergence analyses to express convergence rates in terms of other known system parameters. Such an expression gives further details on what factors impact convergence.**

I. Introduction

Gradient-based algorithms such as stochastic gradient descent (SGD) play a central role in solving many engineering problems that involve stochastic optimization and online learning. These algorithms are iterative and historically have been implemented on a single compute node. The aim of these algorithms is to produce a sequence of parameters that converges to the optimal solution of the optimization problem. Typically, each iteration consists of two steps: computing the gradients of the objective function and updating the optimization parameters using these gradients. Many machine learning applications have achieved state-of-the-art performances thanks to the ability of SGD and its variants to scale to solve large problem sizes [1]–[3]. However, with the advent of large-scale problems, the trend has shifted towards distributed implementations [4]–[6]. For instance, training deep neural network consisting of billions of parameters (e.g., in natural language processing problems) using large datasets on a single compute node can be computationally prohibitive.

A typical approach in parallelizing gradient-based algorithms is to implement a hub-and-spoke system in which a number of compute nodes known as workers are connected to a central node. This central node, also known as the master, keeps track of the optimization parameters while the workers compute the gradients of the objective function in each iteration and push the computations to the master. The migration from the single-node to the distributed computational paradigm comes at the cost of added communication latency and synchronization overhead. In synchronous optimization, the master must wait for all workers to complete their computations before updating the parameters. A very well-known phenomenon in distributed systems is that the speed of the workers can vary unpredictably and that certain workers may be unpredictably slow [7], [8]. These slow workers are known as stragglers. The speed of convergence in synchronous optimization can be limited by these stragglers. Dealing with stragglers in order to speed up convergence has been an active area of research over the past few years [9]–[12].

To alleviate the burden of synchronization and to avoid slow-downs due to stragglers, optimization can proceed in an asynchronous fashion. In asynchronous approaches, the master does not need to wait for all workers. Rather, the master updates the parameters as soon as it receives gradients from a subset of worker. This approach leads to a faster parameters update rate on the expense of using stale (i.e., out-of-date) gradients. Unfortunately, gradient staleness can negatively impact convergence [13]–[15]. When analyzing the convergence of asynchronous methods, gradient staleness is often assumed to be bounded. As a result, the convergence rates of asynchronous methods are presented in terms of the maximum staleness. Despite its role in convergence of asynchronous methods, gradient staleness analysis has not been sufficiently analyzed and has been mostly attributed to the number of workers based on empirical evidences.

Recently, we studied gradient staleness assuming a hub-and-spoke distributed setting in which the communication delay between master and workers is deterministic [16]. We characterized asynchronous distributed optimization as a delayed renewal process and derived the expected value of gradient staleness for a general distribution on per-iteration gradient compute time. However, in practice, inter-node communication time is not deterministic. In this paper, we investigate the gradient staleness that arise in asynchronous distributed optimization when communication is random. We assume that communication times are independent and identically distributed (i.i.d.) random variables, i.i.d. both across workers and across iterations. While the resulting asynchronous system is not a renewal process, it does share some characteristics with renewal process. We derive an upper bound on the expected gradient staleness. The upper bound is achieved with equality when the updated parameters arrive in order. We show numerically that this upper bound becomes tighter as the number of workers become smaller. The bound also gets tighter as the variance of the latency of the communication link from master to workers becomes smaller irrespective of the number of workers. Our derivations reveal that it is possible to adjust gradient staleness by tuning certain parameters such as minibatch size or the number of workers. Our results can

978-1-6654-0540-9/22/$31.00 ©2022 IEEE

be used with existing convergence analyses to derive the convergence requirements in terms of system parameters such as expected compute time, expected communication time and the number of workers.

II. Related Work

Asynchronous distributed optimization has been studied at least since the seminal work of Tsitsiklis et al. [17] in which the authors study the implementation of gradient-based algorithms in a distributed fashion. In recent years, asynchronous optimization has gained a lot of interest due to its importance in solving large-scale machine learning and online learning problems. Most of the existing literature in distributed optimization focuses on either developing frameworks [18]–[21] or algorithms [22]–[25] to achieve faster convergence. Other lines of works study theoretical aspects of distributed optimization such as convergence analysis [4], [13], [26], [27].

As stated earlier, optimization parameters are updated more frequently in asynchronous methods than they are in synchronous methods since there is no need to wait for stragglers to complete their computation. While this approach mitigates the detrimental effects of stragglers, asynchronous methods suffer from optimization parameters being updated using stale (i.e., out-of-date) gradients. Gradient-staleness is an important quantity that shows up in the theoretical analysis of asynchronous algorithms. For instance, when using the asynchronous SGD algorithm with a fixed staleness τ, the convergence rate after T iteration is of order $\mathcal{O}\big(\sqrt{\tau/T}\big)$. Even when the effect of gradient staleness is asymptotically negligible as in [13], larger staleness values have in practice been shown to slow down convergence [28].

Despite its significant impact on convergence in practice, there is but a handful of research publications that investigate the impact of staleness [27], [29], [30]. Furthermore, most of these results are based on intensive experiments rather than rigorous mathematical analyses. We observe that the connection between gradient staleness and other system parameters is usually overlooked. Our recent work in [16] addresses this gap and provides some insight into the sources of gradient staleness and how staleness is related to other system parameters such as expected compute time, communication time, and the number of workers. In [16], we assumed a simple communication model in which the communication time is deterministic. In contrast, in this paper we study a general system model in which communication time can follow any distribution.

III. System Model

Our hub-and-spoke distributed system consists of n workers connected to a central master node. The aim of the distributed system is collaboratively to solve the following optimization problem in an online manner.

$$\min_{w\in\mathcal{W}} F(w) \text{ where } F(w) := \mathbb{E}_x\big[f(w,x)\big] \tag{1}$$

where $x \in \mathcal{X}$ is a random variable that follows distribution P.

The master generates a sequence of optimization parameters $\{w(k)\}_{k\geq 1}$ that converges to $w^* = \arg\min_{w\in\mathcal{W}} F(w)$ while the workers compute the gradients of the objective function and submit their results to the master. The master use these gradients to update the parameters $w(k)$ in the k-th step.

Gradient Computation: Gradient computation is an iterative process. The process begins with the initialization of the optimization parameters $w(0)$ identically across all workers. Since the optimization process is asynchronous, workers may be using different versions of the optimization parameter at any given time after the initial step. Let $w_i(j)$ denotes the optimization parameters used by worker i in the j-th compute round. Each worker computes a local minibatch of size b in each iteration. The local minibatch size b does not change across workers or across iteration. Let $X_i(j)$ denote the time worker i takes to process the j-th minibatch. We assume that the sequence $\{X_i(j), i \in [n], j = 1, 2, \cdots\}$ are i.i.d. with $\mathbb{E}[X_i(j)] = \mu$ for all $i \in [n]$. We assume that workers never idle. Thus, as soon as worker i completes its b computations, it sends the sum of the gradients to the master and immediately starts a new round of computation; communication proceeds in the background. Let $Y_i(j)$ denote the communication time (delay) incurred by worker i's j-th transmission to reach the master. We assume that the $Y_i(j)$ are identically distributed with $\mathbb{E}[Y_i(j)] = T_w$. Let $V_i(j)$ denote the wall clock time at which this j-th local minibatch of worker i arrives at the master. Then, $V_i(j) = \sum_{l=1}^{j} X_i(l) + Y_i(j)$.

Parameter Update: The master waits for K local minibatches to update its parameters and then broadcasts the updated parameters to all workers. Let $w(k)$ be the updated parameters in the k-th update round. Let $Z_i(k)$ denote the communication delay $w(k)$ incurs to reach worker i. $Z_i(k)$ is a random variable with $\mathbb{E}[Z_i(k)] = T_m$ for all $i \in [n]$ and for all $k \geq 1$. Note that we index the iterations at the master with k while we index the compute iterations at workers with j as these two processes are asynchronous.

Let $N(t)$ denote the total number of local minibatches (each of size b gradients) the master has received by time t, where $t \geq 0$. We can express $N(t) = \sum_{i=1}^{n} N_i(t)$, where $N_i(t)$ is the number of local minibatches the master received from worker i by time t. Let $M(t)$ be the index of the most up-to-date optimization parameters at time t. In another word, $M(t)$ is the number of times the master has updated the optimization parameters by time t. Thus, $M(t) = \lfloor N(t)/K \rfloor$. Define $T(k)$ to be the time between the $(k-1)$-st and k-th updates, i.e., $T(k)$ is the k-th inter-update time. Hence the time when the k-th update takes place is $T_{\text{up}}(k) := \arg\min_{t\geq 0}\big\{M(t) \geq k\big\} = \sum_{i=1}^{k} T(i)$.

Gradient Staleness: Suppose that worker i starts computing its j-th local minibatch using $w(k)$ at time $t_1 = \sum_{l=1}^{j-1} X_i(j)$, i.e., $w_i(j) = w(k)$. This implies that $w(k)$ has been broadcasted by the master at the latest by time $t_2 = t_1 - Z_i(j)$. Since all parameters updates $w(m)$ for $m < k$ have been broadcasted prior to t_2, k is the largest value satisfying $T_{\text{up}}(k) \leq t_1 - Z_i(k)$. Equivalently, k is the largest integer such that $\sum_{l=1}^{k} T(l) + Z_k(k) \leq t_1$. The j-th minibatch of worker

i is sent to the master at time $t_1 + X_i(j)$ and is received by the master at time $t = Y_w^{(i)}(j)$. The actual parameter at the master at time t is $w\big(M(t)\big)$. Hence the staleness experienced by these gradients is $\tau(t) = M(t) - k$.

IV. Main Results

We now present our main results. Due to space limitation, proofs are omitted.

First, observe that $N_i(t)$, defined in the previous section as the total number of local minibatches the master receives from worker i by time t, is a counting process with arrival times $V_i(j), j = 1, 2, \cdots$. The process $N_i(t)$ is not a renewal process in general since the inter-arrival times are not i.i.d. However, the following lemma shows that $N_i(t)$ has some properties similar to a renewal process.

Lemma IV.1. *Let $N_i(t)$ be the number of local minibatches the master has received from worker i by time t. Then*

$$\lim_{t\to\infty} \frac{\mathbb{E}[N_i(t)]}{t} = \frac{1}{\mu}, \tag{2}$$

where μ is the expected per-iteration gradient compute time.

Lemma IV.2. *Let $N_i(t)$ and μ be as defined above. Let $\delta > 0$, then*

$$\lim_{t\to\infty} \frac{\mathbb{E}[N_i(t+\delta) - N_i(t)]}{t} = \frac{\delta}{\mu}. \tag{3}$$

Equations (2) and (3) give similar results to the elementary renewal theorem and Blackwell's theorem [31], respectively, although $N_i(t)$ is *not* a renewal process. These two lemmas are important to prove Theorem IV.4.

Lemma IV.3. *Let $M(t)$ be as defined in Sec.III. Then, in the limit, the expected rate of updates the master makes is*

$$\lim_{t\to\infty} \frac{\mathbb{E}[M(t)]}{t} = \frac{n}{K\mu}. \tag{4}$$

Next, we present an upper bound on expected gradient staleness.

Theorem IV.4. *Let $\tau(t)$ be the staleness suffered by gradients received by the master at time t. Then, in the limit*

$$\lim_{t\to\infty} \mathbb{E}[\tau(t)] \leq \frac{n}{K}\frac{T_w + T_m + \mu}{\mu}, \tag{5}$$

where T_w, T_m, and μ are as defined in Sec. III.

We herein provide a sketch of the proof of the theorem. The details of the proof are omitted due to space limitation. The complete proof will appear in the full version of the paper. The proof of Theorem IV.4 uses the results from Lemmas IV.1, IV.2, and IV.3. Suppose that the master computes $w(k)$ and broadcasts it at time t. The expected time that takes $w(k)$ to reach worker i is T_w. Worker i computes gradients using $w(k)$ and finishes at time $t + T_w + \mu$ on average. These gradients on average take T_m seconds to reach the master. The expected time since $w(k)$ was broadcasted till its gradients arrive is $T_w + T_m + \mu$ and during this period the master has made at most $\frac{n}{K\mu}(T_w + T_m + \mu)$ parameter updates which is exactly the upper bound on gradient staleness.

The bound is achieved with equality if the optimization parameters arrive at each worker in sequence. That is, if $w(k)$ always arrives before $w(k+1)$ for all k. To appreciate the role of in-order arrivals, consider the situation when $w(k+a)$ for some nonnegative integer a happens to have a very short communication delay and arrives at worker i before $w(k), w(k+1), ..., w(k+a-1)$. In this situation, the worker ignores all the other updates and uses $w(k+a)$. The result is lower staleness. Note that when communication delays are deterministic, the optimization parameters arrive in order. The upper bound is achieved with equality in this case as well. Our result is consistent with that in our previous work [16] wherein we assumed deterministic communication delays.

The performance of a distributed algorithm is usually evaluated in terms of their regret bounds or convergence rate. These metrics show how the system performs with respect to the number of iterations or data points sampled. The regret and convergence rate of asynchronous algorithms are usually expressed in terms of the number of steps and staleness. For instance, in [32], the convergence rate after T steps is $\mathcal{O}(\sqrt{\mathbb{E}[\tau]/T})$, where τ is gradient staleness. Using our derivation of $\mathbb{E}[\tau]$ from (5), we can express this as $\mathcal{O}\Big(\sqrt{n\big(T_w + T_m + \mu\big)/(K\mu T)}\Big)$. If $\mu \gg T_w + T_m$, the convergence rate is of order $\mathcal{O}(\sqrt{n/(KbT)})$.

In the previous example, the convergence rate was expressed in terms of the expected staleness. However, many convergence analyses in literature assume that staleness is bounded above and derive the convergence rates accordingly. These analyses do not contain the expected staleness. For instance, in [26], the authors show that if the number of steps $T \geq \mathcal{O}(B^2)$ for $\tau \leq B$, then the convergence rate is $\mathcal{O}(1/\sqrt{KbT})$. We can re-derive the same results by considering $\mathbb{E}[\tau]$ instead of B. The details of these derivations are omitted here due to space restriction but will appear in the full version. In this case, to achieve the same regret bound, the number of steps must be $T \geq \mathcal{O}(\mathbb{E}[\tau^2]^2)$. We can tighten this bound if $\mathrm{VAR}(\tau) \leq 2\mathbb{E}[\tau] + 1$. In this case, $T \geq \mathcal{O}(\mathbb{E}[\tau]^2)$. Therefore, if $T \geq (n/K)^2(T_w + T_m + \mu/(\mu))^2$, then we achieve a convergence rate of $\mathcal{O}(1/\sqrt{KbT})$. This bound provides a direct way to evaluate the minimum number of steps required to achieve a certain convergence rate target as it is expressed in terms of other known system parameters. The above two examples illustrate how to benefit from the gradient staleness results we derived in this work.

V. Numerical Results

In this section, we present simulation results to validate our theoretical derivations and examine the performance of the upper bound we derived earlier. In our simulation, we assume the compute time (i.e., $X_i(j)$) is distributed according to the shifted exponential distribution. This distribution is widely used in the literature to model compute time in distributed systems. The shifted exponential distribution has the following probability density function (pdf)

$$f_X(t) = \beta \exp\big\{-\beta(t-\alpha)\big\}, \qquad t \geq \alpha > 0.$$

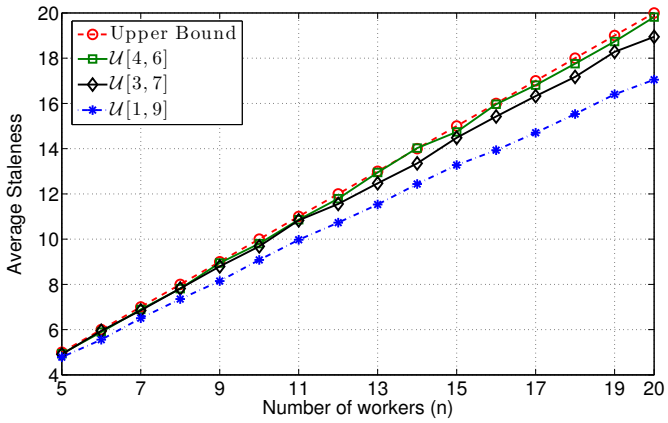

Fig. 1. Average staleness for various uniformly distributed communication delay versus the upper bound.

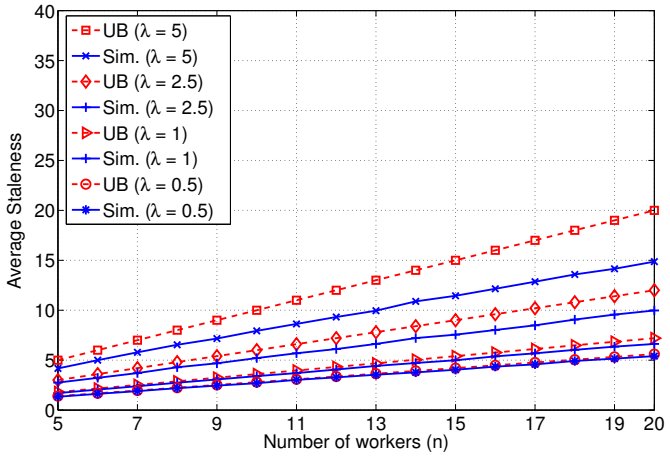

Fig. 2. Average staleness achieved with exponential distribution for communication time versus the theoretical upper bound.

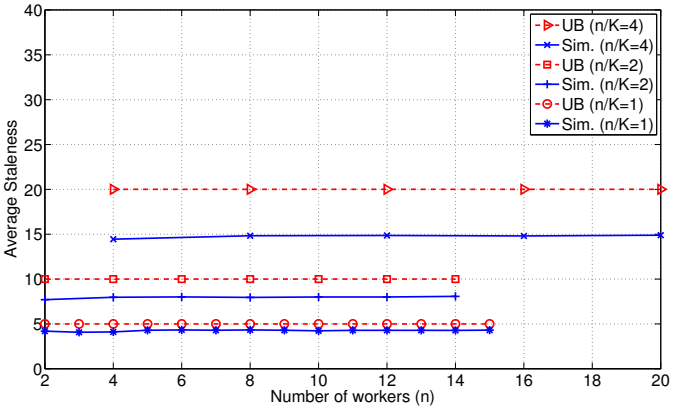

Fig. 3. Average staleness versus the upper bound with respect to n/K

The parameter α ensures that the minimum time required to complete gradient computation is strictly larger than zero. The expected value is $\mu = \alpha + \beta^{-1}$. In our experiment we use $\beta = 1.5$ and $\alpha = 1$. Hence, the expected time that a single worker takes to compute a local minibatch per iteration is $\mu = 2.5$.

For the communication time, we consider both the exponential distribution with parameter $1/\lambda$ (i.e., λ is the expected value) and the uniform distribution on the interval $[a, b]$ which we denote $\mathcal{U}[a, b]$. We run experiments for various values of λ, a, and b. We run each experiment 10 times and present the averaged results.

In our first set of experiments, we set $K = 5$ and vary the number of workers n from K to 20. We remind the reader that K is the number of local minibatches that the master uses per parameters update. We model the communication time for both the uplink transmission (i.e., from workers to master) and the downlink broadcast (i.e., from master to workers) according the uniform distribution $[a, b]$. We run simulations for the following choices of the uniform distributions $\mathcal{U}[4, 6]$, $\mathcal{U}[3, 7]$, and $\mathcal{U}[1, 9]$. Fig. 1 depicts the performance of all three cases. The figure compares the actual gradient staleness with the upper bound derived in (5). We observe that the bound is very tight for $\mathcal{U}[4, 6]$ and $\mathcal{U}[3, 7]$ while it is tight for $\mathcal{U}[1, 9]$ only for smaller values of n. The graph shows that the upper bound is tighter for distributions with smaller variances. The upper bound for all three cases is the same since all three distributions have the same expected value of 5.

In our next set of experiments, we use the exponential distribution to model the uplink and downlink communication delays. We run experiments for $\lambda = 0.5, 1, 2.5$, and 2.5. Fig. 2 compares the simulated average staleness achieved in each setup with its respective upper bound calculated using (5). We again observe that the upper bound is tighter for distributions having smaller variances. Furthermore, the upper bound is tighter for smaller values of n.

Finally, we compare the actual average gradient staleness with the upper bound for different values of n/K. In Fig. 3, we observe that even under communication delays with high variance, the upper bound is accurate for smaller values of n/K even when n is large. This is because when n/K is small, the update rate of the optimization parameters is small according to Lemma IV.3. This means that the chance for parameters to arrive at workers in an out-of-order manner is smaller since newer updates take longer to be produced at the master. We observe that the gap between the upper bound and the average staleness scales with the quantity n/K and the variance of the communication time distribution.

VI. Conclusion

In this paper, we analyze gradient staleness that arises in asynchronous optimization. Our analysis assumes a hub-and-spoke configuration with random inter-node communication delays. We derive an upper bound on the expected gradient staleness. Our derivations show how gradient staleness is related to expected compute time, expected communication delay, and the number of workers. Our results can be used in existing convergence results to derive further insight about which system parameter affect convergence.

VII. Acknowledgement

This work was supported by Huawei Technologies Canada, the Natural Science and Engineering Research Council (NSERC) of Canada through a Discovery Research Grant, and the Omani Government Postgraduate Scholarship. We thank Jason Lam and Zhenhua Hu of Huawei and Tharindu Adikari from University of Toronto for technical discussions.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Inf. Process. Sys.*, 2012, pp. 1097–1105.

[2] K. Simonyan and A. Zisserman., "Very deep convolutional networks for large-scale image recognition," in *Proc. of Int. Conf. on Learning and Representations*, 2015.

[3] L. Bottou and Y. L. Cun, "Large scale online learning," in *Advances in Neural Inf. Process. Sys.*, 2004, pp. 217–224.

[4] O. Dekel, R. Gilad-Bachrach, O. Shamir, and L. Xiao, "Optimal distributed online prediction using mini-batches," *Journal of Machine Learning Research*, pp. 165–202, Jan. 2012.

[5] M. Zinkevich, J. Langford, and A. J. Smola, "Slow learners are fast," in *Advances in Neural Inf. Process. Sys.*, 2009, pp. 2331–2339.

[6] A. Nedic and A. Ozdaglar, "Distributed subgradient methods for multi-agent optimization," *IEEE Trans. on Automatic Cont.*, pp. 48–61, Jan. 2009.

[7] P. Garraghan, X. Ouyang, R. Yang, D. McKee, and J. Xu, "Straggler root-cause and impact analysis for massive-scale virtualized cloud datacenters," *IEEE Trans. on Services Computing*, pp. 91–104, Jan. 2019.

[8] J. Dean and L. A. Barroso, "The tail at scale," *Communications of the ACM*, pp. 74–80, 2013.

[9] N. Ferdinand, H. Al-Lawati, S. C. Draper, and M. Nokleby, "Anytime minibatch: Exploiting stragglers in online distributed optimization," in *Proc. of Int. Conf. on Learning Representations*, 2019.

[10] E. Ozfatura, D. Gündüz, and S. Ulukus, "Speeding up distributed gradient descent by utilizing non-persistent stragglers," *CoRR*, 2018.

[11] C. Karakus, Y. Sun, S. Diggavi, and W. Yin, "Straggler mitigation in distributed optimization through data encoding," in *Advances in Neural Inf. Process. Sys.*, 2017, pp. 5434–5442.

[12] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient coding: Avoiding stragglers in distributed learning," in *Proc. of Int. Conf. on Machine Learning*, ser. Proc. of Machine Learning Research, Aug. 2017, pp. 3368–3376.

[13] A. Agarwal and J. C. Duchi, "Distributed delayed stochastic optimization," in *Advances in Neural Inf. Process. Sys.*, 2011, pp. 873–881.

[14] S. Dutta, G. Joshi, S. Ghosh, P. Dube, and P. Nagpurkar, "Slow and stale gradients can win the race: Error-runtime trade-offs in distributed SGD," in *Proc. of Int. Conf. on Artificial Intelligence and Statistics*, 2018, pp. 803–812.

[15] S. Zheng, Q. Meng, T. Wang, W. Chen, N. Yu, Z. Ma, and T. Liu, "Asynchronous stochastic gradient descent with delay compensation," in *Proc. of the Int. Conf. on Machine Learning*, 2017.

[16] H. Al-Lawati and S. C. Draper, "Gradient delay analysis in asynchronous distributed optimization," in *Proc. of Int. Conf. on Acoustics, Speech, and Sig. Proc.*, 2020, pp. 3992–3996.

[17] J. Tsitsiklis, D. Bertsekas, and M. Athans, "Distributed asynchronous deterministic and stochastic gradient optimization algorithms," *IEEE Trans. on Automatic Cont.*, pp. 803–812, Sep. 1986.

[18] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, Q. V. Le, and A. Y. Ng, "Large scale distributed deep networks," in *Advances in Neural Inf. Process. Sys.*, 2012, pp. 1223–1231.

[19] W. Dai, J. Wei, X. Zheng, J. K. Kim, S. Lee, J. Yin, Q. Ho, and E. P. Xing, "Petuum: A framework for iterative-convergent distributed ML," in *Proc. Int. Conf. Neural Inf. Process. Sys.*, 2013, pp. 1–15.

[20] Z. Peng, Y. Xu, M. Yan, and W. Yin, "Arock: An algorithmic framework for asynchronous parallel coordinate updates," *SIAM Journal on Scientific Computing*, vol. 38, pp. A2851–A2879, 2016.

[21] T. Chilimbi, Y. Suzue, J. Apacible, and K. Kalyanaraman, "Project adam: Building an efficient and scalable deep learning training system," in *Proc. of the USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, Oct. 2014, pp. 571–582.

[22] H. R. Feyzmahdavian, A. Aytekin, and M. Johansson, "An asynchronous mini-batch algorithm for regularized stochastic optimization," *IEEE Trans. on Automatic Cont.*, pp. 3740–3754, Dec. 2016.

[23] J. C. Duchi, A. Agarwal, and M. J. Wainwright, "Dual averaging for distributed optimization: Convergence analysis and network scaling," *IEEE Trans. on Automatic Cont.*, pp. 592–606, Mar. 2012.

[24] L. Xiao, "Dual averaging method for regularized stochastic learning and online optimization," in *Advances in Neural Inf. Process. Sys.*, 2009, pp. 2116–2124.

[25] N. Feng, B. Recht, C. Re, and S. J. Wright, "Hogwild!: A lock-free approach to parallelizing stochastic gradient descent," in *Advances in Neural Inf. Process. Sys.*, 2011, pp. 693–701.

[26] X. Lian, Y. Huang, Y. Li, and J. Liu, "Asynchronous parallel stochastic gradient for nonconvex optimization," in *Advances in Neural Inf. Process. Sys.*, 2015, pp. 2737–2745.

[27] I. Mitliagkas, C. Zhang, S. Hadjis, and C. Ré, "Asynchrony begets momentum, with an application to deep learning," in *Proc. 54th Annual Allerton Conf. on Comm., Control, and Computing*, 2016, pp. 997–1004.

[28] J. Chen, R. Monga, S. Bengio, and R. Jozefowicz, "Revisiting distributed synchronous SGD," in *Proc. of Int. Conf. on Learning and Representations*, 2016.

[29] W. Dai, Y. Zhou, N. Dong, H. Zhang, and E. Xing, "Toward understanding the impact of staleness in distributed machine learning," in *Int. Conf. on Learning Representations*, 2019.

[30] A. Srinivasan, A. Jain, and P. Barekatain, "An analysis of the delayed gradients problem in asynchronous SGD," in *Proc. of Int. Conf. on Learning Representations (Workshop Track)*, 2018.

[31] R. G. Gallager, *Discrete Stochastic Processes*. USA: Springer, 1995.

[32] A. Nedich, D. Bertsekas, and V. Borkar, "Distributed asynchronous incremental subgradient methods," *Studies in Comp. Math.*, pp. 381–407, 2001.