Adaptive Distributed Stochastic Gradient Descent for Minimizing Delay in the Presence of Stragglers

Serge Kas Hanna*, Rawad Bitar*, Parimal Parag†, Venkat Dasari‡, and Salim El Rouayheb*
* Department of Electrical and Computer Engineering, Rutgers University, Piscataway, NJ, USA
† Department of Electrical Communication Engineering, Indian Institute of Science, Bengaluru, KA, India
‡ US Army Research Laboratory, Aberdeen Proving Ground, MD, USA
Emails: {serge.k.hanna, rawad.bitar, salim.elrouayheb}@rutgers.edu, parimal@iisc.ac.in, venkateswara.r.dasari.civ@mail.mil

***Abstract*—We consider the setting where a master wants to run a distributed stochastic gradient descent (SGD) algorithm on n workers each having a subset of the data. Distributed SGD may suffer from the effect of stragglers, i.e., slow or unresponsive workers who cause delays. One solution studied in the literature is to wait at each iteration for the responses of the fastest $k < n$ workers before updating the model, where k is a fixed parameter. The choice of the value of k presents a trade-off between the runtime (i.e., convergence rate) of SGD and the error of the model. Towards optimizing the error-runtime trade-off, we investigate distributed SGD with adaptive k. We first design an adaptive policy for varying k that optimizes this trade-off based on an upper bound on the error as a function of the wall-clock time which we derive. Then, we propose an algorithm for adaptive distributed SGD that is based on a statistical heuristic. We implement our algorithm and provide numerical simulations which confirm our intuition and theoretical analysis.**

***Index Terms*—Distributed SGD, adaptive policy, stragglers.**

I. Introduction

We consider a distributed computation setting in which a master wants to learn a model on a large amount of data in his possession by dividing the computations on n workers. The data at the master consists of a matrix $X \in \mathbb{R}^{m\times d}$ representing m data vectors $\mathbf{x}_\ell$, $\ell = 1, \dots, m$, and a vector $\mathbf{y} \in \mathbb{R}^m$ representing the labels of the rows of X. Define $A \triangleq [X|\mathbf{y}]$ to be the concatenation of X and $\mathbf{y}$. The master would like to find a model $\mathbf{w}^\star \in \mathbb{R}^d$ that minimizes a loss function $F(A, \mathbf{w})$, i.e, $\mathbf{w}^\star = \arg\min_{\mathbf{w}\in\mathbb{R}^d} F(A, \mathbf{w})$. This optimization problem can be solved using Gradient Descent (GD), which is an iterative algorithm that consists of the following update at each iteration j,

$$\mathbf{w}_{j+1} = \mathbf{w}_j - \eta\nabla F(A, \mathbf{w}_j) \triangleq \mathbf{w}_j - \frac{\eta}{m}\sum_{\ell=1}^{m} \nabla F(\mathbf{a}_\ell, \mathbf{w}_j), \quad (1)$$

where η is the step size, and $\nabla F(A, \mathbf{w})$ is the gradient of $F(A, \mathbf{w})$. To distribute the computations, the master partitions the data equally to n workers. The data partitioning is horizontal, i.e., each worker receives a set of rows of A with all their corresponding columns. Let S_i be the sub-matrix of A sent to worker i. Each worker computes a partial gradient defined as $\nabla F(S_i, \mathbf{w}_j) \triangleq \frac{1}{s}\sum_{\mathbf{a}_\ell\in S_i} \nabla F(\mathbf{a}_\ell, \mathbf{w}_j)$, where $s = m/n$ is the number of rows in S_i (assuming n divides m). The master computes the average of the received partial gradients to obtain the actual gradient $\nabla F(A, \mathbf{w})$, and then updates $\mathbf{w}_j$.

In this setting, waiting for the partial gradients of all the workers slows down the process as the master has to wait for the stragglers [1], i.e., slow or unresponsive workers, in order to update $\mathbf{w}_j$. Many approaches have been proposed in the literature to alleviate the problem of stragglers. A natural approach is to simply ignore the stragglers and obtain an estimate of the gradient rather than the full gradient, see [2], [3]. This framework emerges from single-node (non-distributed) mini batch stochastic gradient descent (SGD) [4]. Batch SGD is a relaxation of GD in which $\mathbf{w}_j$ is updated based on a subset (batch) B ($|B| < m$) of data vectors that is chosen uniformly at random from the set of all m data vectors, i.e., $\mathbf{w}_{j+1} = \mathbf{w}_j - \frac{\eta}{|B|}\sum_{\mathbf{a}_\ell\in B} F(\mathbf{a}_\ell, \mathbf{w}_j)$. It is shown that SGD converges to $\mathbf{w}^\star$ under mild assumptions on the loss function $F(A, \mathbf{w})$, but may require a larger number of iterations as compared to GD [4]–[10].

Consider the approach where the master updates the model based on the responses of the fastest $k < n$ workers and ignores the remaining stragglers. Henceforth, we call this approach *fastest-k SGD*. The update rule for fastest-k SGD is given by

$$\mathbf{w}_{j+1} = \mathbf{w}_j - \frac{\eta}{k}\sum_{i\in R_j} \nabla F(S_i, \mathbf{w}_j) \triangleq \mathbf{w}_j - \eta\, \hat{\mathbf{g}}(\mathbf{w}_j), \quad (2)$$

where R_j is the set of the fastest k workers at iteration j; and $\hat{\mathbf{g}}(\mathbf{w}_j)$ is the average of the partial gradients received by the master at iteration j which is an estimate of the full gradient $\nabla F(A, \mathbf{w}_j)$. Note that if we assume that the response times of the workers are random *iid*, then one can easily show that fastest-k SGD is essentially equivalent to single-node batch SGD since the master updates the model at each iteration based on a uniformly random batch of data vectors belonging to the set of the fastest k workers. Therefore, fastest-k SGD converges to $\mathbf{w}^\star$ under the random *iid* assumption on the response times of the workers and the standard assumptions on the loss function $F(A, \mathbf{w})$.

The convergence rate of distributed SGD depends on two factors simultaneously: (i) the error in the model versus the

The work of the first and last authors was supported in part by ARL Grant W911NF-17-1-0032.

The work of the third author was supported in part by the Science and Engineering Research Board under Grant DSTO-1677, the Department of Telecommunications, Government of India, under Grant DOTC-0001, the Robert Bosch Center for Cyber-Physical Systems, and the Centre for Networked Intelligence (a Cisco CSR initiative) of the Indian Institute of Science, Bangalore.

number of iterations; (ii) the time spent per iteration. Therefore, in this work we focus on studying the convergence rate with respect to the wall-clock time rather than the number of iterations. In fastest-k SGD with fixed step size, the choice of the value of k presents a trade-off between the convergence rate and the error floor. Namely, choosing a small value of k will lead to fast convergence since the time spent per iteration would be short, however, this will also result in a low accuracy in the final model, i.e., higher error floor. Towards optimizing this trade-off, we study adaptive policies for fastest-k SGD where the master starts with waiting for a small number of workers k and then gradually increases k to minimize the error as a function of time. Such an optimal adaptive policy would guarantee that the error is minimized at any instant of the wall-clock time. This would be particularly useful in applications where SGD is run with a deadline, since the learning algorithm would achieve the best accuracy within any time restriction.

A. *Related work*

1) Distributed SGD: The works that are closely related to our work are that of [2], [3]. In [3] the authors study fastest-k SGD for a predetermined k. In [2], the authors consider the same setting as [3] and analyze the convergence rate of fastest-k SGD with respect to the number of iterations. In addition to the convergence analysis with respect to the number of iterations, the authors in [2] separately analyze the time spent per iteration as a function of k.

Several works proposed using redundancy when distributing the data to the workers. The master then uses coding theoretical tools to recover the gradient in the presence of a fixed number of stragglers, for example [11]–[24]. In [25]–[27] the authors propose a mixed strategy in which the master distributes the data redundantly and uses coding techniques to obtain the whole gradient for a given number of stragglers. In addition, if more workers than accounted for are stragglers, the master can use the same coding techniques to compute an estimate of the gradient.

Note that the setting of the previously mentioned works, and the setting of interest for our work, focuses on the so-called synchronous SGD in which the workers are all synchronized at each iteration (i.e., have the same model). The literature also studies the asynchronous setting. In asynchronous distributed SGD, whenever a worker finishes its assigned computation, it sends the result to the master who directly updates $\mathbf{w}$ and sends an updated $\mathbf{w}$ to that worker who starts a new computation of the partial gradient while the other workers continue their previous computation, see for example [2], [28]–[32].

2) Single-node SGD: Murata [33] showed that irrespective of its convergence speed, the single-node SGD algorithm with fixed step size goes through a transient phase and a stationary phase. In the transient phase, $\mathbf{w}_j$ approaches $\mathbf{w}^\star$ exponentially fast in the number of iterations. Whereas, in the stationary phase, $\mathbf{w}_j$ oscillates around $\mathbf{w}^\star$. Note that if a decreasing step size over the iterations is used, then $\mathbf{w}_j$ converges to $\mathbf{w}^\star$ rather than oscillating around it, however this leads to a long transient phase and hence a lower convergence rate. To detect the phase transition, [34] uses a statistical test based on Pflug's method [35] for stochastic approximation. Detecting the phase transition serves many purposes, such as indicating when to stop the SGD algorithm or when to start implementing further tricks to reduce the distance between $\mathbf{w}_j$ and $\mathbf{w}^\star$. In this paper, we build on this line of work to derive the times at which the master should start waiting for more workers in fastest-k SGD.

In another line of work on single-node SGD, the authors in [36] suggested increasing the batch size with the number of iterations as an alternative to decreasing the step size. The results in [36] show that increasing the batch size while keeping a constant step size, leads to near-identical model accuracy as decreasing the step size, but with fewer parameter updates, i.e., shorter training time.

B. *Our contributions*

We focus on straggler mitigation in synchronous distributed SGD with fixed step size. We consider a setting where the master distributes the data without redundancy. Under standard assumptions on the loss function, and assuming independent and identically distributed random response times for the workers, we give a theoretical bound on the error of fastest-k SGD as a function of time rather than the number of iterations. We derive an adaptive policy which shows that this bound on the error can be optimized as a function of time by increasing the value of k at specific times which we explicitly determine in terms of the system parameters. Furthermore, we develop an algorithm for adaptive fastest-k SGD that is based on a statistical heuristic which works while being oblivious to the system parameters. We implement this algorithm and provide numerical simulations which show that the adaptive fastest-k SGD can outperform both non-adaptive fastest-k SGD and asynchronous SGD.

II. Preliminaries

In this paper we focus on fastest-k SGD with fixed step size. We consider a random straggling model where the time spent by worker i to finish the computation of its partial gradient (i.e., response time) is a random variable X_i, for $i = 1, \dots, n$. We assume that $X_i, i = 1, \dots, n$, are *iid* and independent across iterations. Therefore, the time per iteration for fastest-k SGD is given by the k^{th} order statistic of the random variables $X_1, \dots, X_n$, denoted by $X_{(k)}$. In the previously described setting, the following bound on the error of fastest-k SGD as a function of the number of iterations was shown in [2], [5].

Proposition 1 (Error vs. iterations of fastest-k SGD [2], [5]). *Under certain assumptions (stated in [2], [5]), the error of fastest-k SGD after j iterations with fixed step size satisfies*

$$\mathbb{E}\left[F(\mathbf{w}_j) - F^\star\right] \leq \frac{\eta L \sigma^2}{2cks} + (1-\eta c)^j \left(F(\mathbf{w}_0) - F^\star - \frac{\eta L \sigma^2}{2cks}\right),$$

where L and c are the Lipschitz and the strong convexity parameters of the loss function respectively, $F^\star$ is the optimal value of the loss function, and σ^2 is the variance bound on the gradient estimate.

III. Theoretical Analysis

In this section, we present our theoretical results. The proofs of these results are available in [37]. In Lemma 1, by applying techniques from renewal theory, we give a bound on the error of fastest-k SGD as a function of the wall-clock time t rather than the number of iterations. The bound holds with high probability for large t and is based on Proposition 1.

Lemma 1 (Error vs. wall-clock time of fastest-k SGD)**.** *Under the same assumptions as Proposition 1, the error of fastest-k SGD after wall-clock time t with fixed step size satisfies*

$$\mathbb{E}\left[F(\mathbf{w}_t) - F^\star \middle| J(t)\right] \leq \frac{\eta L \sigma^2}{2cks} + (1-\eta c)^{\frac{t}{\mu_k}(1-\epsilon)} \left(F(\mathbf{w}_0) - F^\star - \frac{\eta L \sigma^2}{2cks}\right), \quad (3)$$

with high probability $\left(Pr \geq 1 - \frac{\sigma_k^2}{\epsilon^2}\left(\frac{2}{t\mu_k} + \frac{1}{t^2}\right)\right)$ for large t, where $0 < \epsilon \ll 1$ is a constant error term, $J(t)$ is the number of iterations completed in time t, and μ_k is the average of the k^{th} order statistic $X_{(k)}$.

Notice that the first term in (3) is constant (independent of t), whereas the second term decreases exponentially in t ($\eta c < 1$ from [5]). In fact, it is well-known that SGD with constant step size goes first through a transient phase where the error decreases exponentially fast, and then enters a stationary phase where the error oscillates around a constant term [33]. From (3), it is easy to see that the rate of the exponential decrease in the transient phase is governed by the value of $1/\mu_k$. μ_k is an increasing function of k, thus the exponential decrease is fastest for $k = 1$ and slowest for $k = n$. Whereas the stationary phase error which is upper bounded by $\eta L \sigma^2 / 2cks$, is highest for $k = 1$ and lowest for $k = n$. This creates a trade-off between the rate of decrease of the error in the transient phase, and the error floor achieved in the stationary phase. Ultimately, we would like to first have a fast decrease through the transient phase, and then have a low error in the stationary phase. To this end, we look for an adaptive policy for varying k that starts with $k = 1$ and then switches to higher values of k at specific times in order to optimize the error-runtime trade-off. Such an adaptive policy guarantees that the error is minimized at every instant of the wall-clock time t.

Since the bound in (3) holds with high probability for large t, we explicitly derive the switching times that optimize this bound. Note that for the sake of simplicity, we drop the constant error term ϵ in our analysis.

Theorem 1 (Bound-optimal Policy)**.** *The bound-optimal times $t_k, k = 1, \ldots, n-1$, at which the master should switch from waiting for the fastest k workers to waiting for the fastest $k+1$ workers are given by*

$$t_k = t_{k-1} + \frac{\mu_k}{-\ln(1-\eta c)} \times \Big[\ln\left(\mu_{k+1} - \mu_k\right) - \ln\left(\eta L \sigma^2 \mu_k\right) + \ln\left(2ck(k+1)s(F(\mathbf{w}_{t_{k-1}}) - F^\star) - \eta L (k+1)\sigma^2\right)\Big],$$

where $t_0 = 0$.

Example 1 (Theoretical analysis on adaptive fastest-k SGD with *iid* exponential response times)**.** *Suppose $X_i \sim \exp(\mu)$, $i = 1, \ldots, n$. The average time spent per iteration is $\mu_k = H_n - H_{n-k}$, where H_n is the harmonic number. Let $n = 5, \mu = 5, \eta = 0.001, \sigma^2 = 10, F(\mathbf{w}_0) - F^\star = 100, L = 2, c = 1, s = 10$. We evaluate the bound in Lemma 1 for multiple fixed values of k (non-adaptive) and compare it to adaptive fastest-k SGD if we apply the switching times in Theorem 1. The results are shown in Fig. 1.*

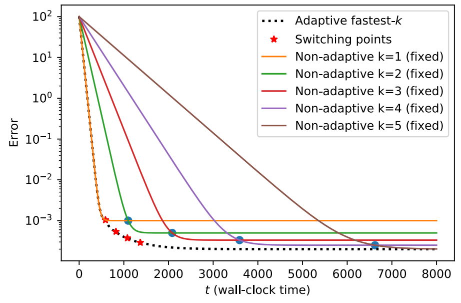

Fig. 1: The upper bound on the error given by (3) as a function of time, evaluated for $k = 1, 2, 3, 4, 5$.

Notice from Fig. 1 that for the time interval $[0, t_1)$, the adaptive policy assigns $k = 1$ since it gives the fastest error decrease in the beginning. Then, as the error approaches the stationary phase, the policy increases k to $k = 2$. This allows the error to decrease below the error floor for $k = 1$. The procedure continues until k attains its maximum value of $k = n = 5$. The results demonstrate the adaptive version enables achieving lower error values in less time.

This analysis shows the potential of adaptive strategies in optimizing the error-runtime trade-off. It also suggests that the value of k should be gradually increased throughout the runtime of fastest-k SGD in order to optimize this trade-off. Although this analysis provides useful insights about how to adapt k over time, it may not be effective in practice for the following two reasons: (i) the policy optimizes an upper bound on the error (Lemma 1) which is probabilistic and may be loose; (ii) the policy requires the knowledge of several system parameters including the optimal value of the loss function $F^\star$ which is typically unknown. Nevertheless, we use the insights provided by the theoretical analysis to design a practical algorithm for adaptive fastest-k SGD. This algorithm is based on a statistical heuristic and is oblivious to the system parameters as we explain in Section IV.

IV. Adaptive Fastest-k SGD Algorithm

In this section we present an algorithm for adaptive fastest-k SGD that is realizable in practice. As previously mentioned, SGD with fixed step size goes first through a transient phase where the error decreases exponentially fast, and then enters a stationary phase where the error oscillates around a constant term. Initially, the exponential decrease is fastest for $k = 1$. Then, as the stationary phase approaches, the error decrease becomes slower and slower until a point where the error starts oscillating around a constant term and does not decrease any further. At this point, increasing k allows the error to decrease further because the master would receive more partial gradients and hence would obtain a better estimate of the full gradient. The goal of the adaptive policy is to detect this phase transition in order to increase k and keep the error decreasing.

The adaptive policy we present in this section detects this phase transition by employing a statistical test based on a modified version of Pflug's procedure for stochastic approximation [35]. The main component of our policy is to monitor the signs of the products of consecutive gradients computed by the master based on (2). The underlying idea is that in the transient phase, due to the exponential decrease of the error, the gradients are likely to point in the same direction, hence their inner product is positive. Our policy consists of utilizing a counter that counts the difference between the number of times the product of consecutive gradients is negative (i.e., $\hat{\mathbf{g}}_j{}^T \hat{\mathbf{g}}_{j-1} < 0$) and the number of times this product is positive, throughout the iterations of the algorithm.

In the beginning of the algorithm, we expect the value of the counter to be negative and decrease because of the exponential decrease in the error. Then, as the error starts moving towards the stationary phase, negative gradient products will start accumulating until the value of the counter becomes larger than a certain positive threshold. At this point, we declare a phase transition and increase k. The complete algorithm is given in Algorithm 1.

Algorithm 1: Adaptive fastest-k SGD

```
input : starting point $\mathbf{w}_0$, data $\{X, \mathbf{y}\}$, number of workers $n$, step size $\eta$,
        maximum number of iterations $J$, adaptation parameters step, thresh,
        burnin
output : weight vector $\mathbf{w}_J$
$j \leftarrow 1$
$k \leftarrow 1$
countNegative $\leftarrow 0$
countIter $\leftarrow 1$
Distribute $X$ to the $n$ workers
while $j \leq J$ do
    Send $\mathbf{w}_{j-1}$ to all workers
    Collect the responses of the fastest $k$ workers
    $\mathbf{w}_j \leftarrow \mathbf{w}_{j-1} - \eta \hat{\mathbf{g}}_{j-1}$
    if $\hat{\mathbf{g}}_j{}^T \hat{\mathbf{g}}_{j-1} < 0$ then
        countNegative $\leftarrow$ countNegative $+ 1$
    else
        countNegative $\leftarrow$ countNegative $- 1$
    end
    if countNegative $>$ thresh and countIter $>$ burnin and $k \leq n -$ step
      then
        $k \leftarrow k +$ step
        countNegative $\leftarrow 0$
        countIter $\leftarrow 0$
    end
    countIter $\leftarrow$ countIter $+ 1$
    $j \leftarrow j + 1$
end
return $\mathbf{w}_J$
```

V. Simulations

A. Experimental setup

We simulated the performance of the adaptive fastest-k SGD (Algorithm 1) described earlier for n workers on synthetic data X. We generated X as follows: (i) we pick each row vector $\mathbf{x}_\ell$, $\ell = 1, \ldots, m$, independently and uniformly at random from $\{1, 2, \ldots, 10\}^d$; (ii) we pick a random vector $\bar{\mathbf{w}}$ with entries being integers chosen uniformly at random from $\{1, \ldots, 100\}$; and (iii) we generate $\mathbf{y}_\ell \sim \mathcal{N}(\langle \mathbf{x}_\ell, \bar{\mathbf{w}} \rangle, 1)$ for all $\ell = 1, \ldots, m$. We run linear regression using the ℓ_2 loss function. At each iteration, we generate n independent exponential random variables with rate $\mu = 1$.

B. Adaptive fastest-k SGD vs Non-adaptive fastest-k SGD

Figure 2 compares the performance of the adaptive fastest-k SGD to non-adaptive for $n = 50$ workers. In the adaptive version we start with $k = 10$ and then increase k by 10 until reaching $k = 40$, where the switching times are given by Algorithm 1. Whereas for the non-adaptive version, k is fixed throughout the runtime of the algorithm. The comparison shows that the adaptive version is able to achieve a better error-runtime trade-off than the non-adaptive one. Namely, notice that the adaptive k-sync reaches its lowest error at approximately $t = 2000$, whereas the non-adaptive version reaches the same error only for $k = 40$ at approximately $t = 6000$. These results confirm our intuition and previous theoretical results.

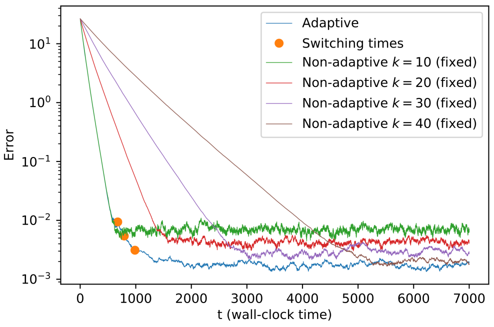

Fig. 2: Error as a function of the wall-clock time for non-adaptive fastest-k SGD with fixed $k = 10, 20, 30, 40$; and adaptive fastest-k SGD (Algorithm 1). The experimental setup is the following: $d = 100$, $m = 2000$, $n = 50$, $\eta = 0.0005$. The adaptation parameters chosen here are $step = 10$, $thresh = 10$, and $burnin = 0.1\times$(number of data points) $= 200$. We start the adaptive fastest-k SGD with $k = 10$ and increase k by 10 until reaching $k = 40$.

C. Comparison to Asynchronous SGD

Figure 3 compares the adaptive fastest-k SGD to the fully asynchronous version of distributed SGD [2]. Similar conclusions can be drawn as in the case of Figure 2.

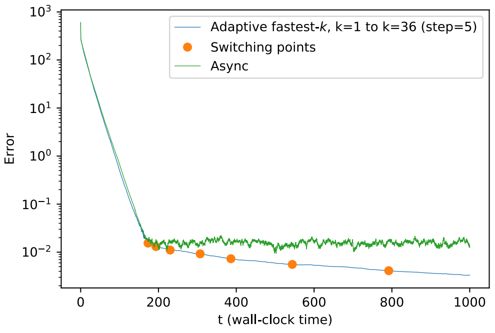

Fig. 3: Error as a function of time for adaptive fastest-k SGD (Algorithm 1) and asynchronous SGD. The experimental setup is the following: $d = 100$, $m = 2000$, $n = 50$, $\eta = 0.0002$. The adaptation parameters chosen here are $step = 5$, $thresh = 10$, and $burnin = 0.1\times$(number of data points) $= 200$. We start the adaptive fastest-k SGD with $k = 1$ and increase k by 5 until reaching $k = 36$.

References

[1] J. Dean and L. A. Barroso, "The tail at scale," *Communications of the ACM*, vol. 56, no. 2, pp. 74–80, 2013.

[2] 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," *arXiv preprint arXiv:1803.01113*, 2018.

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

[4] H. Robbins and S. Monro, "A stochastic approximation method," *The Annals of Mathematical Statistics*, vol. 22, no. 3, pp. 400–407, 1951.

[5] L. Bottou, F. E. Curtis, and J. Nocedal, "Optimization methods for large-scale machine learning," *SIAM Review*, vol. 60, no. 2, pp. 223–311, 2018.

[6] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and distributed computation: numerical methods*, vol. 23. Prentice hall Englewood Cliffs, NJ, 1989.

[7] A. Cotter, O. Shamir, N. Srebro, and K. Sridharan, "Better mini-batch algorithms via accelerated gradient methods," in *Advances in neural information processing systems*, pp. 1647–1655, 2011.

[8] A. Agarwal and J. C. Duchi, "Distributed delayed stochastic optimization," in *Advances in Neural Information Processing Systems*, pp. 873–881, 2011.

[9] O. Dekel, R. Gilad-Bachrach, O. Shamir, and L. Xiao, "Optimal distributed online prediction using mini-batches," *Journal of Machine Learning Research*, vol. 13, no. Jan, pp. 165–202, 2012.

[10] O. Shamir and N. Srebro, "Distributed stochastic optimization and learning," in *Communication, Control, and Computing (Allerton), 2014 52nd Annual Allerton Conference on*, pp. 850–857, IEEE, 2014.

[11] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient coding: Avoiding stragglers in distributed learning," in *International Conference on Machine Learning*, pp. 3368–3376, 2017.

[12] M. Ye and E. Abbe, "Communication-computation efficient gradient coding," *arXiv preprint arXiv:1802.03475*, 2018.

[13] N. Raviv, I. Tamo, R. Tandon, and A. G. Dimakis, "Gradient coding from cyclic mds codes and expander graphs," *arXiv preprint arXiv:1707.03858*, 2017.

[14] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," *IEEE Transactions on Information Theory*, vol. 64, no. 3, pp. 1514–1529, 2018.

[15] N. Ferdinand and S. Draper, "Anytime stochastic gradient descent: A time to hear from all the workers," *arXiv preprint arXiv:1810.02976*, 2018.

[16] Q. Yu, N. Raviv, J. So, and A. S. Avestimehr, "Lagrange coded computing: Optimal design for resiliency, security and privacy," *arXiv preprint arXiv:1806.00939*, 2018.

[17] S. Kiani, N. Ferdinand, and S. C. Draper, "Exploitation of stragglers in coded computation," in *2018 IEEE International Symposium on Information Theory (ISIT)*, pp. 1988–1992, IEEE, 2018.

[18] L. Chen, Z. Charles, D. Papailiopoulos, *et al.*, "Draco: Robust distributed training via redundant gradients," *arXiv preprint arXiv:1803.09877*, 2018.

[19] C. Karakus, Y. Sun, S. Diggavi, and W. Yin, "Straggler mitigation in distributed optimization through data encoding," in *Advances in Neural Information Processing Systems*, pp. 5434–5442, 2017.

[20] W. Halbawi, N. Azizan-Ruhi, F. Salehi, and B. Hassibi, "Improving distributed gradient descent using reed-solomon codes," *arXiv preprint arXiv:1706.05436*, 2017.

[21] S. Dutta, V. Cadambe, and P. Grover, "Short-dot: Computing large linear transforms distributedly using coded short dot products," in 29*th Annual Conference on Neural Information Processing Systems (NIPS)*, pp. 2092–2100, 2016.

[22] Y. Keshtkarjahromi and H. Seferoglu, "Coded cooperative computation for internet of things," *CoRR*, vol. abs/1801.04357, 2018.

[23] M. Fahim, H. Jeong, F. Haddadpour, S. Dutta, V. Cadambe, and P. Grover, "On the optimal recovery threshold of coded matrix multiplication," in *Communication, Control, and Computing (Allerton), 2017 55th Annual Allerton Conference on*, pp. 1264–1270, IEEE, 2017.

[24] R. Bitar, P. Parag, and S. El Rouayheb, "Minimizing latency for secure distributed computing," in *International Symposium on Information Theory (ISIT)*, pp. 2900–2904, June 2017.

[25] Z. Charles, D. Papailiopoulos, and J. Ellenberg, "Approximate gradient coding via sparse random graphs," *arXiv preprint arXiv:1711.06771*, 2017.

[26] Z. Charles, D. Papailiopoulos, and J. Ellenberg, "Approximate gradient coding via sparse random graphs," *arXiv preprint arXiv:1711.06771*, 2017.

[27] R. K. Maity, A. S. Rawat, and A. Mazumdar, "Robust gradient descent via moment encoding with ldpc codes," *arXiv preprint arXiv:1805.08327*, 2018.

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

[29] J. Liu, S. J. Wright, C. Ré, V. Bittorf, and S. Sridhar, "An asynchronous parallel stochastic coordinate descent algorithm," *The Journal of Machine Learning Research*, vol. 16, no. 1, pp. 285–322, 2015.

[30] S. Shalev-Shwartz and T. Zhang, "Accelerated mini-batch stochastic dual coordinate ascent," in *Advances in Neural Information Processing Systems*, pp. 378–385, 2013.

[31] S. J. Reddi, A. Hefny, S. Sra, B. Poczos, and A. J. Smola, "On variance reduction in stochastic gradient descent and its asynchronous variants," in *Advances in Neural Information Processing Systems*, pp. 2647–2655, 2015.

[32] X. Pan, M. Lam, S. Tu, D. Papailiopoulos, C. Zhang, M. I. Jordan, K. Ramchandran, and C. Ré, "Cyclades: Conflict-free asynchronous machine learning," in *Advances in Neural Information Processing Systems*, pp. 2568–2576, 2016.

[33] N. Murata, "A statistical study of on-line learning," *Online Learning and Neural Networks. Cambridge University Press, Cambridge, UK*, pp. 63–92, 1998.

[34] J. Chee and P. Toulis, "Convergence diagnostics for stochastic gradient descent with constant learning rate," in *International Conference on Artificial Intelligence and Statistics*, pp. 1476–1485, 2018.

[35] G. C. Pflug, "Non-asymptotic confidence bounds for stochastic approximation algorithms with constant step size," *Monatshefte für Mathematik*, vol. 110, no. 3-4, pp. 297–314, 1990.

[36] S. L. Smith, P.-J. Kindermans, C. Ying, and Q. V. Le, "Don't decay the learning rate, increase the batch size," *arXiv preprint arXiv:1711.00489*, 2017.

[37] S. Kas Hanna, R. Bitar, P. Parag, V. Dasari, and S. El Rouayheb, "Adaptive distributed stochastic gradient descent for minimizing delay in the presence of stragglers (extended version)," 2019. http://eceweb1.rutgers.edu/~csi/AdaptiveSGD.pdf.