Computation Scheduling for Distributed Machine Learning with Straggling Workers

Mohammad Mohammadi Amiri and Deniz Gündüz
Electrical and Electronic Engineering Department, Imperial College London, London SW7 2BT, U.K.

***Abstract*—We study scheduling of computation tasks across n workers in a large scale distributed learning problem. Computation speeds of the workers are assumed to be heterogeneous and unknown to the master, and redundant computations are assigned to the workers in order to tolerate straggling workers. We consider sequential computation and instantaneous communication from each worker to the master, and each computation round, which can model a single iteration of the stochastic gradient descent (SGD) algorithm, is *completed* once the master receives $k \leq n$ distinct computations, referred to as the *computation target*. Our goal is to characterize the *average completion time* as a function of the *computation load*, which denotes the portion of the dataset available at each worker, and the computation target. We propose two computation scheduling schemes that specify the computation tasks assigned to each worker, as well as their order of execution. We also establish a lower bound on the minimum average completion time. Numerical results show a significant reduction in the average computation time over the existing coded and uncoded computing schemes.**

***Index Terms*—Machine learning, distributed computation.**

I. Introduction

The growing computational complexity and huge memory requirements of emerging big-data applications involving massive datasets cannot be satisfied on a single machine. Hence, distributed computation (DC) across tens or even hundreds of computation servers, called the *workers*, has been a topic of great interest [1]–[3]. A major bottleneck in DC, and its application in large machine learning applications, is that the overall performance can significantly deteriorate due to slow servers, referred to as the *stragglers*. To mitigate this limitation, coded computation techniques, inspired by erasure codes against packet losses, have been proposed recently [4]–[13]. With coded computation, results from only a subset of non-straggling workers are sufficient to complete the computation task. Please see [14] for an overview and classification of different approaches.

Most existing coded computation techniques are designed to tolerate persistent stragglers, i.e., workers that are extremely slow compared to the rest for a long period of time; and therefore, they discard computations performed by straggling workers. However, persistent stragglers are rarely seen in practice, and we often encounter *non-persistent stragglers*, which, despite being slower, complete a significant portion of the assigned tasks by the time faster workers complete their assignments [15]. Recently, there have been efforts to exploit the computations that have been carried out by non-persistent stragglers at the expense of increasing the communication load [14]–[18]. Techniques studied in [14]–[17] are based on coding with associated encoding and decoding complexities, which may require central processing of all the data points at the *master*. Furthermore, the coded design proposed in [15] depends on the statistical behavior of the stragglers, which may not be possible to predict accurately in practice. The coding technique proposed in [17] requires a large number of data samples assigned to each worker, while the approach in [18] requires a large number of workers compared to the number of computation tasks that must be completed.

As opposed to the prevailing coded computation approaches in the literature, we do not apply any coding across the dataset or the computations; and instead, consider a centralized scheduling strategy, which specifies the computations tasks assigned to each worker and the other they are carried out. Each worker can compute a limited number of tasks, referred to as the *computation load*. Computations are carried out sequentially, and each computation is sent to the master immediately after it is completed. Communication delay from the workers to the master is ignored, although independent delays across workers can easily be incorporated into our framework (included in the longer version of the paper [19]). This sequential computation and communication framework allows the master to exploit even partial computations by slow workers.

Assuming that the computation time at each worker is random, the goal is to characterize the minimum *average completion time*. We first provide a closed-form expression for the average completion time as a function of the *computation schedule*. We propose two computation assignment schemes, and evaluate their average completion times. We also establish a lower bound on the minimum average completion time, which is shown to be tight numerically for low and high computation loads.

II. Problem Formulation

We consider DC of an arbitrary function h over a dataset $\boldsymbol{\mathcal{X}} = \{X_1, ..., X_n\}$ across n *workers*. Each X_i, which we will call as a data point, may correspond to a matrix/vector representing a minibatch of labeled data samples with the same size concatenated along the same dimension. Function $h : \mathbb{V} \to \mathbb{U}$ is an arbitrary function, where $\mathbb{V}$ and $\mathbb{U}$ are two vector spaces over the same field $\mathbb{F}$, and each X_i is an element of $\mathbb{V}$. The dataset $\boldsymbol{\mathcal{X}}$ is distributed across the workers by the

This work has been funded by the European Research Council (ERC) through project BEACON (No. 725731).

master, and a maximum number of $r \leq n$ data points are assigned to each worker, referred to as the *computation load*.

The computations of the tasks assigned to each worker are carried out sequentially. We define the task ordering (TO) matrix C as an $r \times n$ matrix of integers, $C \in [n]^{r\times n}$, $[n] \triangleq \{1, 2, \ldots, i\}$, specifying the computation schedule of the tasks assigned to each worker. Let c_{ij} denote the entry with the i-th row and j-th column of the TO matrix C, $i \in [r]$, $j \in [n]$. Each column of matrix C corresponds to a different worker, and its elements from top to bottom represent the order of computations. That is, the entry c_{ij} denotes the index of the element of the dataset that is computed by worker j as its i-th evaluation, $i \in [r]$, i.e., worker j, $j \in [n]$, first computes $h(X_{c_{1j}})$, then computes $h(X_{c_{2j}})$, and so on so forth until either it computes $h(X_{c_{rj}})$, or it receives an acknowledgement message from the master, and stops computations. While any C matrix is a valid TO matrix, it is easy to see that the optimal TO matrix will have r distinct entries in each of its columns.

We denote by T_i the time worker i spends to compute each task assigned to it. We assume that T_i is a random variable with cumulative distribution function (CDF) F_i, $i \in [n]$, and, for $j \neq i$, T_i is independent of T_j. While the computation speed of each server is random, we assume that, once its value is fixed, each computation at that server takes the same amount of time. This is motivated by the assumption that all the computation tasks are of equal size and complexity. Moreover, since tasks are computed sequentially one after another by a worker, they are likely to experience similar computing resources and delays. Each worker sends the result of its computation to the master immediately after its computation. We assume that the communication time is negligible compared to the computation time; that is, the result of each computation becomes available at the master immediately after its completion.

We denote the time that the master receives $h(X_j)$ by T_{X_j}, $j \in [n]$, which is a random variable. Let T_{i,X_j} denote the time worker i, $i \in [n]$, computes $h(X_j)$, then we have $T_{X_j} = \min_{i\in[n]}\{T_{i,X_j}\}$, $j \in [n]$. The distributions of $T_{X_1}, \ldots, T_{X_n}$ depend on the assignment of the computation tasks to the workers, as well as the order these tasks are carried out by each worker. If evaluation $h(X_j)$ has not been assigned to worker i, we assume that $T_{i,X_j} = \infty$, for $i, j \in [n]$. We note that T_{X_i}, in general, is not independent of T_{X_j}, for $i, j \in [n]$.

Computation is considered completed once the master recovers the results of k distinct tasks $h(X_i)$. We denote the random completion time by $T_C(r, k)$. We define the average completion time as $\overline{T}_C(r, k) \triangleq \mathbb{E}[T_C(r, k)]$, where the expectation is taken over the distributions of $T_1, ..., T_n$. We also define the minimum average completion time $\overline{T}^*(r, k) \triangleq \min_C \{\overline{T}_C(r, k)\}$, where the minimization is taken over all possible TO matrices C. It is trivial to see that the optimal TO matrix will have k distinct entries overall. The goal is to characterize $\overline{T}^*(r, k)$.

We highlight that, most existing coded DC schemes require the master to recover the computations (or, their average) for the whole dataset. However, it is known that convergence of stochastic gradient descent (SGD) is guaranteed even if the gradient is computed at only a portion of the dataset at each iteration [20]–[27]. This is indeed the case for the random straggling model considered here, where the straggling workers, and hence the gradient values that are not computed, vary randomly at each iteration.

III. Average Completion Time Analysis

In the following theorem, whose proof can be found in the longer version [19], we analyze $\overline{T}_C(r, k)$ for a TO matrix C.

Theorem 1. *For a given TO matrix C, we have*

$$\Pr\{T_C(r,k) > t\} = \sum\nolimits_{i=n-k+1}^{n} (-1)^{n-k+i+1} \binom{i-1}{n-k} \sum\nolimits_{\mathcal{S}\subset[n]:|\mathcal{S}|=i} \Pr\left\{T_{X_j} > t, \forall j \in \mathcal{S}\right\}, \quad (1)$$

which yields

$$\overline{T}_C(r,k) = \sum\nolimits_{i=n-k+1}^{n} (-1)^{n-k+i+1} \binom{i-1}{n-k} \sum\nolimits_{\mathcal{S}\subset[n]:|\mathcal{S}|=i} \int_0^\infty \Pr\left\{T_{X_j} > t, \forall j \in \mathcal{S}\right\} dt. \quad (2)$$

Note that the dependence of the completion time statistics on the TO matrix in (1) and (2) is through the statistics of T_{X_j}. We also note that the expectations in (1) and (2) are fairly general, and can apply to models where the computation time of a task may depend on its order of computation, rather than being the same for all the computations carried out by the same worker (see [19] for the details).

The minimum average completion time $\overline{T}^*(r, k)$ can be obtained as a solution of the optimization problem $\overline{T}^*(r, k) = \min_C \overline{T}_C(r, k)$. Providing a general characterization for $\overline{T}^*(r, k)$ is elusive. In the next section, we will propose two specific computation task assignment and scheduling schemes.

IV. Upper Bounds on $\overline{T}^*(r, k)$

Here we introduce and study *cyclic scheduling* (CS) and *staircase scheduling* (SS) schemes. The average completion time for these schemes will provide upper bounds on $\overline{T}^*(r, k)$.

A. Cyclic Scheduling (CS) Scheme

CS scheme is motivated by the symmetry across the workers when we have no prior information on their computation speeds. CS makes sure that each computation task has a different order at different workers. This is achieved by a cyclic shift operator. We denote the TO matrix of the CS scheme by C_{CS}, and its element in the i-th row and j-th column by $C_{\mathrm{CS}}(i, j)$, for $i \in [r]$ and $j \in [n]$. We have

$$C_{\mathrm{CS}}(i, j) = g(j + i - 1), \quad \text{for } i \in [r],\ j \in [n], \quad (3)$$

where function $g : \mathbb{Z} \to \mathbb{Z}$ is defined as follows:

$$g(l) \triangleq \begin{cases} l, & \text{if } 1 \leq l \leq n, \\ l - n, & \text{if } l \geq n + 1, \\ l + n, & \text{if } l \leq 0. \end{cases} \quad (4)$$

Due to the linear scaling of the computation time with the number of computations executed we have, for $j \in [n]$,

$$T_{g(j-i+1),X_j} = \begin{cases} iT_{g(j-i+1)}, & \text{for } i = 1,\dots,r, \\ \infty, & \text{for } i = r+1,\dots,n, \end{cases} \quad (5)$$

which results in $T_{X_j} = \min_{i=1,\dots,r} \left\{ iT_{g(j-i+1)} \right\}$.

B. Staircase Scheduling (SS) Scheme

The SS scheme introduces inverse computation orders at the workers. The entries of the TO matrix $C_{\rm SS}$ for the SS scheme are

$$C_{\rm SS}(i,j) = g(j + (-1)^{j-1}(i-1)), \text{ for } i \in [r],\ j \in [n]. \quad (6)$$

We remark here that the main difference between the CS and SS schemes is that in CS (according to (3)) all the workers have the same step size and direction in their computations, while in SS (according to (6)) workers with even and odd indices have different directions (ascending and descending, respectively) in the order they carry out the computations, but the same step size in their evaluations.

For the SS scheme, it can be verified that, for $j \in [n]$,

$$T_{g(j+(-1)^{j+i-1}(i-1)),X_j} = \begin{cases} iT_{g(j+(-1)^{j+i-1}(i-1))}, & \text{for } i = 1,\dots,r, \\ \infty, & \text{for } i = r+1,\dots,n, \end{cases} \quad (7)$$

which results in $T_{X_j} = \min_{i=1,\dots,r} \left\{ iT_{g(j+(-1)^{j+i-1}(i-1))} \right\}$.

In the longer version of the paper [19], we derive a closed-form expression for the average completion time of both the CS and SS schemes for a general statistical model on the delays.

V. Lower Bound on $\overline{T}^*(r,k)$

Next, we present a lower bound on the minimal average completion time $\overline{T}^*(r,k)$ by considering an adaptive model. Note that the TO matrix, in general, may depend on the statistics of the computation times, i.e., F_i, $i \in [n]$, but it cannot depend on the particular realization of $T_1,\dots,T_n$. In this section, we allow the master to employ a distinct TO matrix $C_{\mathbf{T}}$ for each realization of $\mathbf{T} = (T_1,\dots,T_n)$. For given $\mathbf{T}$, let $T_{C_{\mathbf{T}}}(\mathbf{T},r,k)$ denote the completion time; that is, the master can receive k distinct computations by time $T_{C_{\mathbf{T}}}(\mathbf{T},r,k)$. We define $T_{\rm LB}(\mathbf{T},r,k) \triangleq \min_{C_{\mathbf{T}}} \{T_{C_{\mathbf{T}}}(\mathbf{T},r,k)\}$, where the minimization is taken over all possible TO matrices $C_{\mathbf{T}}$. We also define $\overline{T}_{\rm LB}(r,k) \triangleq \mathbb{E}\left[T_{\rm LB}(\mathbf{T},r,k)\right]$, where the expectation is taken over $\mathbf{T}$. We have

$$\begin{aligned} \overline{T}^*(r,k) &= \min_C \left\{ \mathbb{E}\left[T_C(r,k)\right] \right\} \\ &\geq \mathbb{E}\left[\min_{C_{\mathbf{T}}} \left\{ T_{C_{\mathbf{T}}}(\mathbf{T},r,k) \right\}\right] = \overline{T}_{\rm LB}(r,k). \end{aligned} \quad (8)$$

It can be shown that the lower bound $T_{\rm LB}(\mathbf{T},r,k)$ is obtained as the solution of the greedy algorithm outlined in Algorithm 1, which finds the optimum solution $k_1^*,\dots,k_n^*$. After finding $k_1^*,\dots,k_n^*$, we can obtain $T_{\rm LB}(\mathbf{T},r,k) = \max\{k_1^*T_1,\dots,k_n^*T_n\}$. Since finding the statistics of $T_{\rm LB}(\mathbf{T},r,k)$ is analytically elusive, we obtain the lower bound on $\overline{t}^*(r,k)$ through Monte Carlo simulations.

Algorithm 1 $T_{\rm LB}(\mathbf{T},r,k)$

procedure FINDING $k_1^*,\dots,k_n^*$
  $k_i = 0$, $i \in [n]$
  $T_i' = T_i$, $i \in [n]$
  for $l = 1,2,\dots,k$ **do**
    Find m_l: $(k_{m_l}+1)\,T'_{m_l} = \min\{(k_1+1)\,T_1',\dots,(k_n+1)\,T_n'\}$
    $k_{m_l} \leftarrow k_{m_l} + 1$
    if $k_{m_l} = r$ **then**
      $T'_{m_l} \leftarrow \infty$
    end if
  end for
  $k_i^* = k_i$, $i \in [n]$
end procedure

VI. Numerical Results

Here we numerically compute the average completion time of the proposed CS and SS schemes, and compare them with the existing results in the literature, which includes *random assignment* (RA) [18], *polynomially coded* (PC) [12], and polynomially coded multi-message (PCMM) [14] schemes, as well as the above lower bound. We refer to the longer version of the paper [19] for the description of the RA, PC, and PCMM schemes, and point out that we have $r = n$ for the RA scheme, and $k = n$ and $r \geq 2$ for both PC and PCMM schemes. We denote the average completion time of the CS, SS, RA, PC, and PCMM by $\overline{T}_{\rm CS}(r,k)$, $\overline{T}_{\rm SS}(r,k)$, $\overline{T}_{\rm RA}(k,k)$, $\overline{T}_{\rm PC}(r,n)$, and $\overline{T}_{\rm PCMM}(r,n)$, respectively.

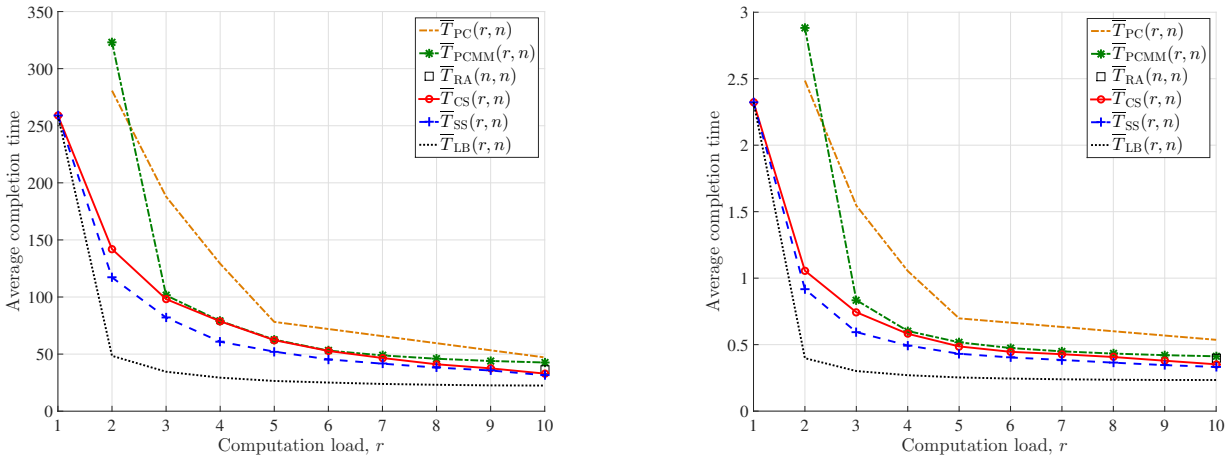
For the fairness of the comparison, we let $k = n$ for the proposed schemes (since the coded schemes PC and PCMM are designed to recover the target function at all the dataset). We assume a shifted exponential computation time at worker i, i.e., for $i \in [n]$,

$$F_i(t) = \begin{cases} 1 - e^{-\mu_i(t-\tau_i)}, & \text{if } t \geq \tau_i, \\ 0, & \text{if } t < \tau_i, \end{cases} \quad (9)$$

and investigate the following scenarios:

- Scenario 1: We consider $n = 10$, where μ_i and τ_i are picked independently and uniformly at random from intervals $(0,0.1)$ and $(0,1)$, respectively, for $i \in [10]$.
- Scenario 2: We consider $n = 10$, and where μ_i and τ_i are picked independently and uniformly at random from intervals $(0,8)$ and $(0,0.1)$, respectively, for $i \in [10]$.

We compare the upper and lower bounds on the minimum average completion time for Scenarios 1 and 2 in Fig. 1a and Fig. 1b, respectively. We note that for both scenarios, the values of $\mu_1,\dots,\mu_n$ and $\tau_1,\dots,\tau_n$ are distinct. In Scenario 1, the workers are on average slower compared to Scenario 2, and the worker speeds are more likely to be skewed. Both the SS and CS schemes significantly improve upon the PC scheme for both scenarios, while the relative improvement is more notable in Scenario 2. CS scheme outperforms the PCMM scheme with a significant gain for small values of



(a) Scenario 1

(b) Scenario 2

Fig. 1: Upper and lower bounds on the minimum average completion time.

r, but their performances become similar as r increases. SS scheme, on the other hand, improves upon the PCMM scheme for all values of r. The proposed CS and SS schemes for a computation load $r = n = 10$, provide a reduction of around $\%16$ and $\%20$ in average completion time upon the RA scheme in Scenario 1 and Scenario 2, respectively. For Scenario 1, SS can achieve the same average delay as the RA scheme for $r = 9$, while for Scenario 2, with a computation load $r = 6$, the performance of SS is close to that of the RA scheme. Observe that the proposed CS and SS schemes perform closer to the lower bound for Scenario 2 compared to Scenario 1 in terms of the average completion time.

VII. Conclusions

We have studied DC across inhomogeneous computation servers, i.e., workers, where each has access to a limited portion of the dataset, defined as the computation load. The computation here may correspond to each iteration of a gradient descent algorithm applied on a large dataset, and it is considered to be completed when the master receives any k distinct computations. In contrast to the growing literature on coded computation to mitigate straggling servers, here we study uncoded computations and sequential communication in order to benefit from all the computations carried out by the workers, including the slower ones. Since the instantaneous computation speeds of the workers are not known in advance, allocation of the tasks to the workers and their scheduling become crucial in minimizing the average completion time. In particular, we consider the assignment of the available data points to workers with a predesigned computation order. Workers send the result of each computation to the master as soon as it is executed. We have proposed two particular computation allocation schemes, called CS and SS. The CS csheme ensures that each computation task experiences different computation orders at different workers, in order to guarantee symmetry, achieved by a cyclic shift operator. However, since the goal is to recover distinct computations at the server, having the same (cyclic) computation order at all the workers may delay some of the computations. For example, the second computation task is computed after task 1 in all but one of the servers, which may result in a lot of redundant computations of task 1, while task 2 may be delayed. To resolve this, in SS, we introduce inverse computation orders at the workers. We have compared the performance of these proposed schemes with the existing schemes in the literature, referred to as PC [12], PCMM [14] and RA [18]. Numerical results show that the CS and SS schemes can provide significant reduction in the average completion time over these schemes, particularly when the speeds of the workers are comparable, in which case utilizing the computations of the relatively slower workers become more beneficial. We have also observed that SS in general outperforms CS thanks to the different computation schedules assigned to the workers. In [19], we show that these observations also hold for a practical scenario implemented on Amazon EC2 cluster.

We also highlight that the reduction in the average completion time is obtained at the expense of an increase in the amount of communications from the workers to the master. If the underlying communication infrastructure can accommodate multiple data packets from each worker, uncoded computation with one of the proposed scheduling schemes will be very efficient in exploiting multiple inhomogeneous computation servers. We also remark that, unlike the proposed schemes, the coded schemes PC and PCMM introduce additional encoding and decoding complexities, which have not been considered in the evaluations here, and may introduce further computation delay. Moreover, in the case of distributed SGD, having computed the partial gradient on separate data points may allow the workers to exploit more advanced methods to reduce their communication load, such as compression [22], [28] or quantization [20], [29], and can be beneficial in the case of communications over noisy channels [30], which may not be applicable in the case of coded computations (see [31] for an exception).

REFERENCES

[1] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2012.

[2] J. Choi, D. W. Walker, and J. J. Dongarra, "Pumma: Parallel universal matrix multiplication algorithms on distributed memory concurrent computers," *Concurrency: Practice and Experience*, vol. 6, no. 7, pp. 543–570, 1994.

[3] M. M. Amiri and D. Gündüz, "Machine learning at the wireless edge: Distributed stochastic gradient descent over-the-air," *CoRR*, vol. abs/1901.00844, 2019. [Online]. Available: http://arxiv.org/abs/1901.00844

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

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

[6] W. Halbawi, N. A. Ruhi, F. Salehi, and B. Hassibi, "Improving distributed gradient descent using Reed-Solomon codes," *arXiv:1706.05436v1 [cs.IT]*, Jun. 2017.

[7] 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:1803.01113v3 [cs.IT]*, May 2018.

[8] R. K. Maity, A. S. Rawat, and A. Mazumdar, "Robust gradient descent via moment encoding with LDPC codes," *arXiv:1805.08327v1 [cs.IT]*, May 2018.

[9] M. Ye and E. Abbe, "Communication-computation efficient gradient coding," *arXiv:1802.03475v1 [cs.IT]*, Feb. 2018.

[10] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Straggler mitigation in distributed matrix multiplication: Fundamental limits and optimal coding," *arXiv:1801.07487v2 [cs.IT]*, May 2018.

[11] S. Dutta, M. Fahim, F. Haddadpour, H. Jeong, V. Cadambe, and P. Grover, "On the optimal recovery threshold of coded matrix multiplication," *arXiv:1801.10292v2 [cs.IT]*, May 2018.

[12] S. Li, S. M. Mousavi Kalan, Q. Yu, M. Soltanolkotabi, and A. S. Avestimehr, "Polynomially coded regression: Optimal straggler mitigation via data encoding," *arXiv:1805.09934v1 [cs.IT]*, May 2018.

[13] Q. Yu, N. Raviv, J. So, and A. S. Avestimehr, "Lagrange coded computing: Optimal design for resiliency, security and privacy," *arXiv:1806.00939v2 [cs.IT]*, Jun. 2018.

[14] E. Ozfatura, D. Gündüz, and S. Ulukus, "Speeding up distributed gradient descent by utilizing non-persistent stragglers," *arXiv:1808.02240v2 [cs.IT]*, Aug. 2018.

[15] N. Ferdinand and S. C. Draper, "Hierarchical coded computation," in *Proc. IEEE Int'l Symp. on Inform. Theory (ISIT)*, Vail, CO, USA, Jun. 2018, pp. 1620–1624.

[16] S. Kiani, N. Ferdinand, and S. C. Draper, "Exploitation of stragglers in coded computation," *arXiv:1806.10253v1 [cs.IT]*, Jun. 2018.

[17] A. Mallick, M. Chaudhari, and G. Joshi, "Rateless codes for near-perfect load balancing in distributed matrix-vector multiplication," *arXiv:1804.10331v2 [cs.DC]*, Mar. 2018.

[18] S. Li, S. M. Mousavi Kalan, A. S. Avestimehr, and M. Soltanolkotabi, "Near-optimal straggler mitigation for distributed gradient methods," *arXiv:1710.09990v1 [cs.IT]*, Oct. 2017.

[19] M. Mohammadi Amiri and D. Gündüz, "Computation scheduling for distributed machine learning with straggling workers," *arXiv:1810.09992 [cs.DC]*, Oct. 2018.

[20] N. Strom, "Scalable distributed DNN training using commodity GPU cloud computing," in *INTERSPEECH*, 2015, pp. 1488–1492.

[21] A. F. Aji and K. Heafield, "Sparse communication for distributed gradient descent," *arXiv:1704.05021v2 [cs.CL]*, Jul. 2017.

[22] J. Konecny, H. B. McMahan, F. X. Yu, P. Richtarik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *arXiv:1610.05492v2 [cs.LG]*, Oct. 2017.

[23] F. Sattler, S. Wiedemann, K.-R. Müller, and W. Samek, "Sparse binary compression: Towards distributed deep learning with minimal communication," *arXiv:1805.08768v1 [cs.LG]*, May 2018.

[24] C.-Y. Chen, J. Choi, D. Brand, A. Agrawal, W. Zhang, and K. Gopalakrishnan, "AdaComp : Adaptive residual gradient compression for data-parallel distributed training," *arXiv:1712.02679v1 [cs.LG]*, Dec. 2017.

[25] Y. Lin, S. Han, H. Mao, Y. Wang, and W. J. Dally, "Deep gradient compression: Reducing the communication bandwidth for distributed training," *arXiv:1712.01887v2 [cs.CV]*, Feb. 2018.

[26] Z. Tao and Q. Li, "eSGD: Communication efficient distributed deep learning on the edge," in *Workshop on Hot Topics in Edge Computing (HotEdge)*, Boston, MA, USA, Jul. 2018.

[27] H. Wang, S. Sievert, S. Liu, Z. Charles, D. Papailiopoulos, and S. Wright, "ATOMO: Communication-efficient learning via atomic sparsification," *arXiv:1806.04090v2 [stat.ML]*, Jun. 2018.

[28] W. Wen, C. Xu, F. Yan, C. Wu, Y. Wang, Y. Chen, and H. Li, "TernGrad: Ternary gradients to reduce communication in distributed deep learning," *arXiv:1705.07878v6 [cs.LG]*, Dec. 2017.

[29] F. Seide1, H. Fu, J. Droppo, G. Li, and D. Yu, "1-bit stochastic gradient descent and its application to data-parallel distributed training of speech DNNs," in *INTERSPEECH*, Singapore, Sep. 2014, pp. 1058–1062.

[30] M. Mohammadi Amiri and D. Gündüz, "Machine learning at the wireless edge: Distributed stochastic gradient descent over-the-air," *arXiv:1901.00844 [cs.DC]*, Feb. 2019.

[31] E. Ozfatura, S. Ulukus, and D. Gündüz, "Distributed gradient descent with coded partial gradient computations," *CoRR*, vol. abs/1811.09271, 2018. [Online]. Available: http://arxiv.org/abs/1811.09271