Regulating Workers in Federated Learning by Yardstick Competition

Yunus Sarikaya
ysarikaya@sabanciuniv.edu
Sabanci University
Istanbul, Turkey

Ozgur Ercetin
oercetin@sabanciuniv.edu
Sabanci University
Istanbul, Turkey

ABSTRACT

Due to the large size of the training data, distributed learning approaches such as federated learning have gained attention recently. However, the convergence rate of distributed learning suffers from heterogeneous worker performance. In this paper, we consider an incentive mechanism for workers to mitigate the delays in completion of each batch. To motivate the workers to perform at their best by assigning higher computational resources to the learning task, we use a yardstick of average desired delay to complete each mini-batch calculation. The rewards are determined by on how much each worker deviates from this yardstick. We analytically obtain the optimum equilibrium strategy of the workers as well as the optimal reward function of the model owner that achieves the average desired delay while minimizing the cost of operation. Our numerical results indicate that by adjusting budget parameters, the model owner should judiciously decide on the number of workers due to trade off between the diversity provided by the number of workers and the latency of completing the training.

ACM Reference Format:
Yunus Sarikaya and Ozgur Ercetin. 2019. Regulating Workers in Federated Learning by Yardstick Competition. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The recent success of deep learning approaches for domains such as speech recognition and computer vision stems from many algorithmic improvements but also from the fact that the size of available training data has grown significantly over the years, together with the computing power. The current trend is to use a larger data set and to train deeper networks (higher number of layers) to improve the accuracy. However, the complexity and the memory requirements quickly become unmanageable within the resources of a single machine. An efficient way to deal with this colossal computing task within a reasonable training time is to adopt distributed computation, and to exploit computation and memory resources of multiple machines in parallel. In [1], the Federated Learning (FL) protocol was introduced to allow the mobile devices perform computation of model training locally on their training data according to the model released by the model owner. Such a design enables mobile users to collaboratively learn a shared prediction model while keeping all the training data private on the device.

Most of the popular distributed training algorithms include mini-batch versions of stochastic gradient descent (SGD). Unfortunately, fully-synchronous implementations of stochastic optimization are often slow in practice due to the need to wait for the slowest machine in each synchronous batch, i.e., they suffer from the so called *straggler effect*. For example, experiments on Amazon EC2 instances show that some workers can be five times slower than the typical performance [2]. There have been several attempts in the literature to mitigate the straggler effect by adding redundancy to the distributed computing system via coding [3, 4] or via scheduling computation tasks [5, 6].

Existing studies have an optimistic assumption that all the mobile devices contribute their resource unconditionally [7, 8], which is not practical in the real world due to resource cost incurred by model training [9]. Without well designed economic compensation, the self-interested mobile devices are reluctant to participate in model training [8, 10, 11]. Therefore, it is necessary to design an effective incentive mechanism for stimulating mobile devices to become workers for federated learning tasks [12, 13]. In [14], a game theoretical approach is established to consider a communication incentive in federated learning, where the aim was to construct a relay network and cooperative communications for supporting model update transfer. In [15], authors adopt the contract theory to design an incentive mechanism for encouraging the mobile devices with high-quality data to participate in federated learning. In particular, well-designed contracts can reduce information asymmetry through self-revealing mechanisms in which participants select only the contracts specifically designed for their types. Then, the authors combine a reputation-based worker selection scheme for reliable federated learning with contract theory by using a multi-weight subjective logic model [16]. In our previous work, we introduced an incentive mechanism in FL protocol to improve its efficiency by optimizing the power allocation across workers instead of treating all workers equally [17]. In this setting, at each gradient update step, the model owner offers an incentive to each worker participating in the federated learning process. Based on this incentive, the workers determine the CPU power they will use to calculate their gradient from the local data. The model owner has a finite budget and distributes its budget among its workers to achieve a fast convergence to a target error rate. We modeled the interaction between the mobile devices and the model owner as a Stackelberg game. In Stackelberg game, the model owner is the buyer as it buys the learning service provided by the mobile devices. Then, the mobile devices that are the service providers act as the sellers. The model owner inherently

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

acts as the single leader in the upper level of the Stackelberg game while the mobile devices are the corresponding followers.

In this paper, we investigate a new mechanism that improves efficiency of budget allocation of the parameter server. In particular, we observed that under aforementioned Stackelberg formulation, higher incentives are allocated to straggler workers and those workers that produce model updates quickly receiving smaller incentives. This incentive structure does not motivate the workers to improve their service level. In fact, it is beneficial for a worker to lag behind others to gain higher incentives. This may result in an inefficient equilibrium, wherein all workers slow down to increase their reward.

Therefore, we alleviate this problem by developing a new incentive mechanism wherein the workers are rewarded according to the duration of their mini-batch computation. The key problem with incentivizing the computing power is that it does not motivate the workers to improve their service quality (in this case, delay) if they are rational entities. In particular, workers would benefit from increased delays since this means a larger reward is handed out by the PS. In this paper, we address this issue by incentivizing the performance directly by giving reward to workers according to how much they improve upon a target delay value (yardstick). In line with the literature on yardstick competition, yardstick is determined according to the average performance of other competitors; hence, motivating all to participate in a truthful fashion.

Specifically, the parameter server announces a *yardstick* which is the desired delay for each worker to complete its model update. If the worker can upload its SGD parameters exactly equal to this deadline it receives a fixed reward. The reward increases proportionally with the duration the worker can improve upon the deadline, but it decreases if the duration of mini-batch update takes longer than the deadline.

Yardstick competition was first proposed to regulate the competition among firms, where the cost of service paid by the regulators to the firms do not incentivize the competitors to improve their services or reduce their costs.

2 SYSTEM MODEL

We consider a cooperative federated learning system as shown in Figure 1. Specifically, a model owner (MO) employs a set of mobile devices, i.e., workers to train a high-quality centralized model. The workers fetch the current parameters $\boldsymbol{w}_i$ from the MO as and when instructed in the algorithm. Then, they compute gradients using one mini-batch and push their gradients back to the model owner. At each iteration, the MO aggregates the gradients computed by the workers and updates the parameter $\boldsymbol{w}$.

Let $T_{i,t}$ be the time elapsed for the worker i to update the gradient in iteration t. Here, we consider plain *fully*-synchronous SGD such that the model owner waits for all the workers to push their gradients. Thus, iteration t is completed in $\max_i T_{i,t}$ time, i.e., when all workers send their gradient updates.

Each worker i allocates a CPU power, i.e., P_i^t to complete mini-batch in iteration t. Let the total number of CPU cycles required to complete the computation of gradient of mini-batch in iteration t be c_i^t. We assume that c_i^t is random and independently distributed across mini-batches and workers [18]. Hence, $T_{i,t}$ has a mean $\frac{P_i}{\bar{c}_i}$, where $\bar{c}_i$ is the mean of c_i^t which may follow any arbitrary distribution.

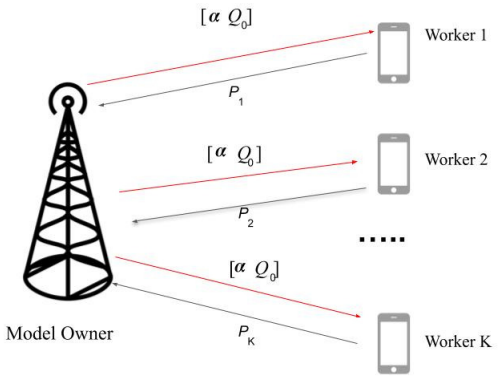


Figure 1: Game Model of Synchronous Federated Learning.

At the beginning of the learning task the MO announces a desired target delay, T_i^d for the completion of each mini-batch computation for every worker i. [1] In return, each worker $i = 1, \ldots, K$ receives a reward q_i^t from the MO. The MO defines the reward for worker according to the difference between the target delay desired by the MO, $T_i^{desired}$ and the realized delay in the previous iteration, i.e., $T_{i,t-1}$ [2]. Let $d_i^t = T_{i,t-1} - T_i^{desired}$ be the deviation of the actual processing delay and desired delay for worker i after ith iteration. The reward is defined per unit power allocated by the worker, and it is defined as:

$$q_i^t = Q_0 - \alpha d_i^t, \tag{1}$$

where Q_0 and α are design parameters. Note that Q_0 is the reward given to the worker, if it has met the desired target delay bound. Also note that when the worker improves upon the target delay then it receives α reward per unit improvement per unit CPU power allocated to the computation task. In the rest of the paper, we will drop the superscript t, since the parameters' dependence to the iteration number is clear. Then, the reward per unit computation power given in (1) can be rewritten as:

$$q_i = Q_0 - \alpha \cdot d_i. \tag{2}$$

3 WORKERS' OPTIMIZATION PROBLEM

Note that while worker i obtains a revenue of $q_i P_i$ from the model owner, it also has an energy cost incurred due to the computation of learning task. The energy cost depends on the value of CPU power used, P_i, as $\kappa c_i (P_i)^2$, where κ is a coefficient depending on the chip architecture [19]. Thus, each worker i aims to maximize the following utility function:

$$U_i(P_i) = q_i P_i - \kappa c_i (P_i)^2, \tag{3}$$

[1] The case when the MO updates the yardstick at the beginning of every iteration will also be analyzed numerically in Section ??.

[2] The reward may depend on a partial or full history of the realized delays albeit at increased complexity. The investigation of more advanced techniques is left as future work

subject to $P_i \leq P_{\max}$, where $P_{\max}$ is the maximum available CPU power of the worker. Note that the objective function (3) is the net profit of the worker participating in the learning task by CPU power i. The utility function is concave in P_i, so the optimal CPU power can be determined from first-order stationarity conditions as:

$$P_i^* = \max\left\{0, \min\left\{P_{\max}, \frac{q_i}{2\kappa c_i}\right\}\right\}. \quad (4)$$

Note that at the beginning of iteration t the worker is aware of the mini-batch size, so the worker is aware of the CPU cycles required to complete the computation of gradient, i.e., c_i, and hence, the time duration it will take to complete, i.e., $\frac{c_i}{P_i}$. By inserting the definition of q_i from (2), we can rewrite the optimal worker CPU power allocation as:

$$P_i^* = \max\left\{0, \min\left\{P_{\max}, \frac{Q_0}{2\kappa c_i}\right\}\right\}. \quad (5)$$

4 MODEL OWNER'S OPTIMIZATION PROBLEM

The objective of the MO is to complete the learning task as quickly as possible, where the duration of the learning task depends on the dataset, the algorithms used in learning, desired accuracy and the processing delay of the computation of subgradients by each worker. In this work, we mainly focus on developing a mechanism to induce quicker computations of the workers, and thus, we consider the maximum delay of completion of iteration based on the CPU power allocations of the workers as the only relevant criterion to achieve this objective. The total average cost of the MO is given as:

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{i} q_i^t P_i^{t*}, \quad (6)$$

where T is the total number of iterations undertaken in the learning application.

In developing a regulation framework, the MO would like to achieve the aforementioned objective at a minimum average cost. Hence, MO should select Q_0 and α accordingly. The MO does not have instantaneous value of the number of CPU cycles required to complete the computation of gradient of mini-batch, but knows its long-term mean $\bar{c}_i$.

Interestingly, if the reward function is an affine function, the optimal CPU power of worker i depends only on the reward given to the worker if it is operating at the desired delay point. However, the worker participates in the learning task only if it has a non-negative utility. Hence, we can obtain a condition on the value of α by inserting (5) in (3) for the *typical* case of the number of CPU cycles required to complete computation in an iteration is equal to the mean $\bar{c}_i$ as:

$$\begin{aligned} Q_0 - \alpha \cdot \frac{\bar{c}_i}{P_i^*} - \kappa \bar{c}_i (P_i^*)^2 &\geq 0, \\ \frac{Q_0^2}{4\kappa \bar{c}_i} - \alpha \bar{c}_i &\geq 0, \\ \alpha &\leq \frac{Q_0^2}{4\kappa \bar{c}_i^2}. \end{aligned} \quad (7)$$

Note that the MO aims to update its model parameter after every desired delay period, $T_i^{desired}$. If the MO is using plain synchronous SGD, it needs to wait for the completion of computations of subgradients from all workers before updating the model. Hence, for fully synchronous SGD, the MO should select $T_i^{desired} = T^{desired}$, for all i.

We first analyze the homogeneous case, where c_i^t is identically distributed with mean $\bar{c}$ for all i. In this case, the average delay per iteration is simply $\frac{\bar{c}}{P^*}$, where P^* is the optimal CPU power allocation of workers for the *typical* case of the number of CPU cycles needed for completion being equal to $\bar{c}$. Hence, typically we want the average computation time of mini-batch subgradients per iteration to be equal to the desired delay bound, i.e.,

$$\begin{aligned} \frac{\bar{c}}{P^*} &= T^{desired}, \\ \Rightarrow \frac{\bar{c}}{\frac{Q_0}{2\kappa\bar{c}}} &= T^{desired}. \end{aligned} \quad (8)$$

In order to satisfy this condition, the reward function should satisfy:

$$Q_0^* = \frac{\bar{c}^2}{T^{desired}} 2\kappa. \quad (9)$$

Hence, inserting (9) into (7), we obtain

$$\alpha \leq \frac{\bar{c}^2}{(T^{desired})^2}\kappa. \quad (10)$$

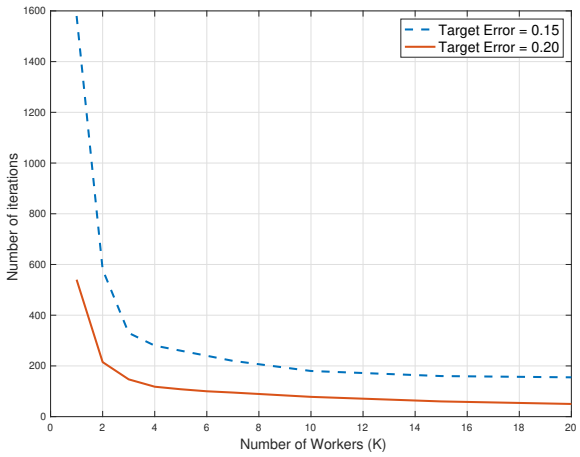
Therefore, from (2), we can see that in order to minimize the total cost while operating at the desired delay point, the value of α should be equal to the right side of (10). Finally, the average cost of the MO operating at the desired delay point can be calculated by inserting the optimal values of Q_0, α and the optimal CPU allocation of the workers in (6):

$$\frac{\bar{c}^3}{(T^{desired})^2}\kappa. \quad (11)$$

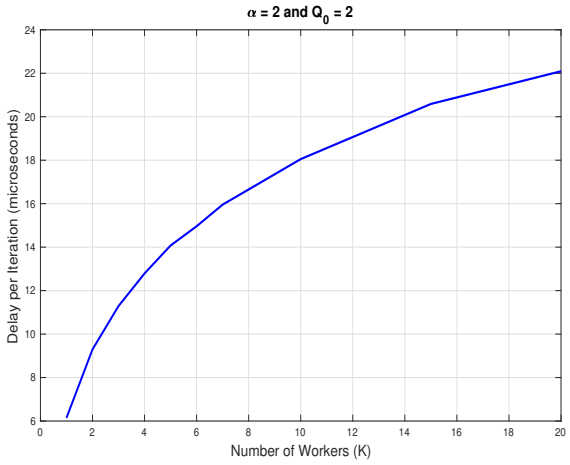
Hence, if the average computation task per mini-batch is high, then the cost of the MO operating at a desired delay point is also higher. On the other hand, if the MO is less sensitive to delay, then its cost of operation can be reduced.

5 NUMERICAL RESULTS

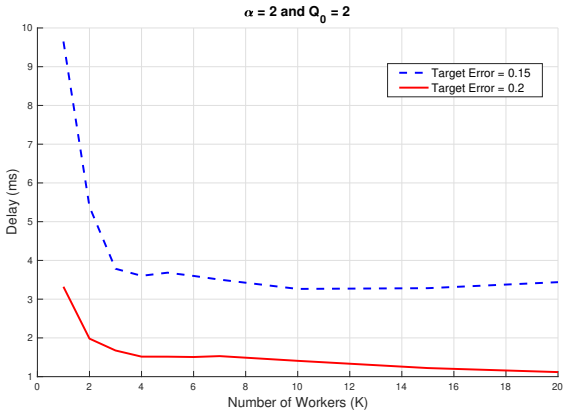
In the simulations, we use MNIST dataset for which we first convert the 28 x 28 images into single vectors of length 784. We use a single layer of neurons followed by soft-max cross entropy with logits loss function. Thus, effectively the parameters consist of a weight matrix $\boldsymbol{W}$ of size 784 x 10 and a bias vector $\boldsymbol{b}$ of size 1 x 10. We use a regularizer of value 0.01, and learning rate of value 0.05. Furthermore, maximum CPU power of a single worker, P_{max} is selected as $5 \cdot 10^8$ cycles per second, and $\kappa = 10^{-11}$. For implementation we used Tensorflow with Python3. For the run-time simulations, we generate random variables from the respective distributions in Python to represent the computation times. Specifically, the computation time for each worker is generated from an exponential distribution with mean $\frac{P_i}{c_i}$. Furthermore, to consider heterogeneous workers in the system, we select c_i uniformly randomly in the range of $[0.5 \cdot 10^3, 1.5 \cdot 10^3]$. Due to the randomness of the selection of c_i and randomness of stochastic gradient descent (SGD), we run each realization 50 times



(a)



(b)



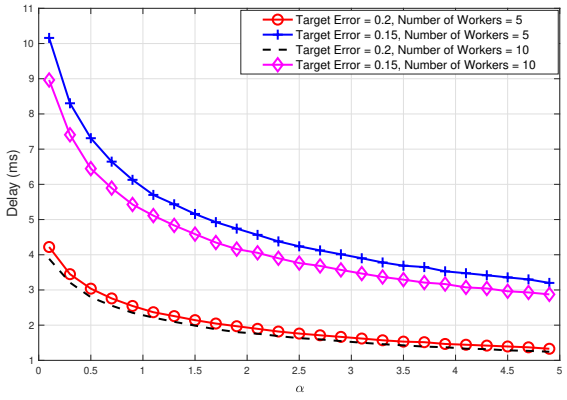
(c)

Figure 2: Number of iterations, average delay per iteration and average delay with respect to number of workers.

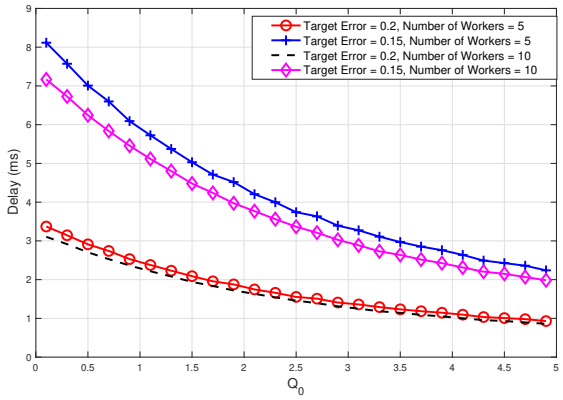
and take the average. In the simulations, we are interested in the error rate defined as the ratio of the difference between the processed image and the original image with the original image. The simulations end when a target error rate is realized, and the time elapsed to reach this target error rate is recorded. In all simulations, $T^{desired}$ at iteration t is taken as delay of preceding iterations which is averaged over the number of preceding iterations and the number of workers.

For a given number of workers, there is one-to-one mapping between the number of iterations and the corresponding wall-clock delay. In Fig. 2, we illustrate the relationship for the case when the budget, α and Q_0, are equal to 2. As seen in Fig. 2a, the number of iterations decreases with increasing number workers due to the effect of increasing diversity gain. On the other hand, as seen in Fig. 2b, the average time required to complete an iteration increases with increasing number of workers, since the model owner has to wait for all workers to finish their corresponding iterations. In Fig. 2c as obtained the results given Fig 2a and 2b, we show that the total delay decreases with increasing number of workers.

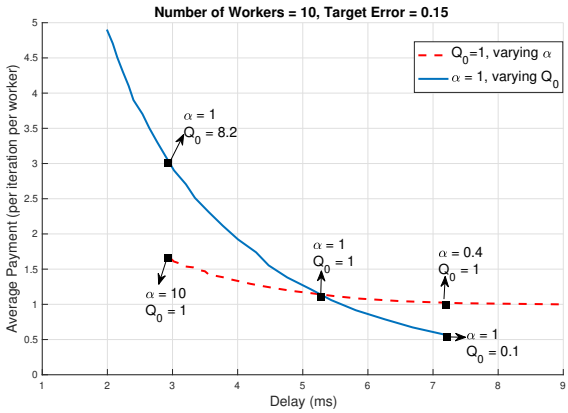
Next, we investigate the effect of varying the number of workers and the budget parameters α and Q_0 to reach a target error rate in Fig. 3. Note that for every value of the budget parameters α and Q_0, the latency decreases with the number of workers, since the error improves with increasing K. This is because, a higher number of workers in the training leads to an increase in diversity, and thus, the system requires fewer iterations to reach the target error rate. In Fig. 3a, we fix the value of Q_0 to 1 and vary the value of α and in Fig. 3b, we fix the value of α to 1 and vary the value of Q_0. As seen in both figures, the delay decreases with increasing value of α and Q_0, since the model owner gives more incentive to the workers to contribute their power to the learning. Notice that the slope in Fig. 3a is steeper compared to the one in Fig. 3b. This is because, with increasing α value, model owner encourages the workers to



(a) α vs. Delay



(b) Q_0 vs. Delay



(c)

Figure 3: Performance analysis of the Yardstick Competition.

obtain smaller delays than the desired delay and this results in faster decrease in total delay.

Finally, we investigate the budget performance with respect to the achieved delay with varying values of α and Q_0. As depicted in Fig. 3c, most of delay values can be achieved with different α and Q_0 values. However, for small delay values, it is better to have high α value instead of high Q_0 in terms of payment performance. On the other hand, for high delay values, the result is reversed such that relatively high Q_0 value results in better payment performance. This is because, Q_0 value encourages workers to participate in distributed learning, and α value encourages workers to user more CPU power, so that their delay gets smaller compared to the desired delay.

6 CONCLUSION

Note that apart from a few recent works, the literature on FL almost entirely treat workers identically, where it is assumed that the workers dedicate all of their resources to the computational task of model worker. Clearly, with workers being individual (selfish) devices such as smartphones, this is not a valid assumption. These devices perform the learning task along with several other ongoing tasks; and thus, their limited computational resource should be shared among these tasks. Our work is one of the first in FL, which introduces a mechanism for the model owner to distribute incentives to the participating workers in order to improve the convergence time of the learning task. Our results demonstrate the importance of an efficient resource allocation in a practical learning system with non-altruistic workers. Specifically, we demonstrate that a reward function

In case, the workers are heterogeneous, we cannot obtain a simple common expression for all workers as in (8). Hence, the objective of the MO should be to minimize the absolute drift from the desired delay point. We leave the analysis of this case along with several other extensions to our future work.

One important direction of extension of this work is to consider a dynamic game formulation that arises when the dynamic channel and worker CPU conditions are taken into account. Additionally, the case of multiple MOs opens up several important questions, where the MOs should not only regulate the workers but also compete among

each other for the services of the workers. Finally, the generalization of the reward function other than that of the affine function analyzed in this work would be an interesting future direction.

REFERENCES

[1] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Arcas, "Communication-efficient learning of deep networks from decentralized data," in *International Conference on Artificial Intelligence and Statistics (AISTATS)*, Fort Lauderdale, FL, USA, Apr. 2017.

[2] A. G. D. R. Tandon, Q. Lei and N. Karampatziakis, "Gradient coding: avoiding stragglers in distributed learning," in *Proc. Int. Conf. on Machine Learning*, Sydney, Australia, Feb. 2017, pp. 3368–3376.

[3] 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.

[4] C. Karakus, Y. Sun, S. Diggavi, and W. Yin, "Straggler mitigation in distributed optimization through data encoding," in *Advances in Neural Information Processing Systems 30 (NIPS)*, Long Beach, NY, USA, Dec. 2017, pp. 5440–5448.

[5] A. Harlap, H. Cui, W. Dai, J. Wei, G. R. Ganger, P. B. Gibbons, G. A. Gibson, and E. P. Xing, "Addressing the straggler problem for iterative convergent parallel ml," in *ACM Symposium on Cloud Computing (SoCC)*, Santa Clara, CA, USA, Oct. 2016, pp. 98–111.

[6] Y. Sun, J. Zhao, S. Zhou, and D. Gündüz, "Heterogeneous computation across heterogeneous workers," *CoRR*, vol. abs/1904.07490, 2019. [Online]. Available: http://arxiv.org/abs/1904.07490

[7] M. Shayan, C. Fung, and et al., "Biscotti: A ledger for private and secure peer-to-peer machine learning," *CoRR*, vol. abs/1811.09904, 2018. [Online]. Available: http://arxiv.org/abs/1811.09904

[8] H. Kim, J. Park, M. Bennis, and S.-L. Kim, "Blockchained on-device federated learning," *IEEE Communication Letters*, vol. 1, no. 1, pp. 1–1, 2019.

[9] Z. Zhou, P. Liu, and et al., "Computation resource allocation and task assignment optimization in vehicular fog computing: A contract-matching approach," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 1, pp. 3113–3125, Apr. 2019.

[10] N. H. Tran, W. Bao, and et al., "Federated learning over wireless networks: Optimization model design and analysis,," in *IEEE Conference on Computer Communications*, Apr. 2019, pp. 1387–1395.

[11] Y. Li, C. Courcoubetis, and L. Duan, "Recommending paths: Follow or not follow?" in *IEEE Conference on Computer Communications*, Apr. 2019, pp. 928–936.

[12] W. Y. B. Lim, N. C. Luong, D. T. Hoang, Y. Jiao, Y.-C. Liang, Q. Yang, D. Niyato, and C. Miao, "Federated learning in mobile edge networks: A comprehensive survey," *CoRR*, vol. 1909.11875, 2019. [Online]. Available: https://arxiv.org/abs/1909.11875

[13] T. Anh, N. Luong, and D. Niyatoetal, "Efficient training management for mobile crowd-machine learning: A deep reinforcement learning approach," *IEEE Communication Letters*, vol. 1, no. 1, pp. 1–1, 2019.

[14] S. Feng, D. Niyato, P. Wang, D. I. Kim, and Y. Liang, "Joint service pricing and cooperative relay communication for federated learning," *CoRR*, vol. abs/1811.12082, 2018. [Online]. Available: http://arxiv.org/abs/1811.12082

[15] J. Kang, Z. Xiong, D. Niyato, Y.-C. L. H. Yu, and j. . C. v. . a. y. . . u. . h. a. . a. e. . . t. . F. b. . h. b. . d. D. I. Kim, title = Incentive design for efficient federated learning in mobile networks: A contract theory approach.

[16] J. Kang, Z. Xiong, D. Niyato, S. Xie, and J. Zhang, "Incentive mechanism for reliable federated learning: A joint optimization approach to combining reputation and contract theory," *IEEE Internet of Things*, vol. 1, no. 1, pp. 1–1, 2019.

[17] Y. Sarikaya and O. Ercetin, "Motivating workers in federated learning: A stackelberg game perspective," *CoRR*, vol. abs/1908.03092, 2019. [Online]. Available: https://arxiv.org/abs/1908.03092

[18] 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," *CoRR*, vol. abs/1803.01113, 2018. [Online]. Available: https://arxiv.org/abs/1803.01113

[19] J. Zhang, X. Hu, Z. Ning, E. C. . Ngai, L. Zhou, J. Wei, J. Cheng, and B. Hu, "Energy-latency tradeoff for energy-aware offloading in mobile edge computing networks," *IEEE Internet of Things Journal*, vol. 5, no. 4, pp. 2633–2645, Aug 2018.