Decentralized Data Allocation via Local Benchmarking for Parallelized Mobile Edge Learning

Duncan J. Mays, Sara A. Elsayed, Hossam S. Hassanein
School of Computing, Queen's University, Kingston, ON, Canada
duncan.mays@queensu.ca, selsayed@cs.queensu.ca, hossam@cs.queensu.ca

2022 International Wireless Communications and Mobile Computing (IWCMC) | 978-1-6654-6749-0/22/$31.00 ©2022 IEEE | DOI: 10.1109/IWCMC55113.2022.9824967

***Abstract*—Multi-Access Edge Computing (MEC) has emerged as a computing paradigm that can facilitate the use of Mobile Edge Learning (MEL), where Machine Learning (ML) models are processed at the edge. In MEL, it is important to address system heterogeneity in a way that minimizes staleness to improve learning accuracy. To do so, a centralized data allocation approach is typically used. However, this approach tends to overlook the privacy of learners, since learners' capabilities are assumed to be known beforehand by the orchestrator. In this context, we propose the Data Allocation via Benchmarking (DAB) scheme. DAB is a decentralized data allocation scheme that eliminates staleness and achieves a certain QoS while preserving the privacy of learners. DAB does not allow any information about the learners to be known to the orchestrator. Instead, each learner estimates the upper bound on the amount of data that it can train such that a certain training deadline is not exceeded. In addition, DAB proposes a novel method to enable each learner to accurately estimate its own hardware characteristics via benchmarking. Extensive performance evaluations on a real testing environment have shown that DAB can outperform the centralized data allocation scheme by up to 12% and 26% in terms of loss and prediction accuracy, respectively. Performance evaluations also show that the proposed benchmarking scheme yields an 83% reduction in benchmarking error compared to a prominent baseline scheme.**

***Index Terms*—Distributed Learning, Federated learning, Parallel Learning, Resource Allocation, Mobile Edge Learning, Edge Computing**

I. Introduction

With the proliferation of Internet of Things (IoT), it is expected that by 2027, 41 billion IoT devices will come online, generating an additional 800 zettabytes of data [1] [2]. The time-sensitive nature of such data is expected to force 90% of analytics to be performed at the edge to avoid latency of transmission to remote data centers in cloud computing [3]. MEC has emerged as a computing paradigm that can enable data processing at the edge (i.e., edge processing) [4]. ML is one example of such edge processing [5].

Performing ML in a distributed manner, which is referred to as Distributed Learning (DL), has gained significant momentum lately [5]. In particular, MEL, which enables ML models to be collaboratively trained on a collection of resource-constrained wireless edge devices, has been capturing the attention of the research community [6]. This can be attributed to the ongoing and substantial increase in the number of edge devices (i.e., learners). Despite being resource-constrained individually, the collective power of such devices can be significantly profuse. The integration of these abundant yet underutilized computational resources with MEL provides a promising edge learning paradigm for a broad range of IoT applications [5] [6].

MEL can be categorized into two categories; Parallelized Learning (PL) and Federated Learning (FL) [5] [6]. In PL, a global orchestrator transmits randomly picked subsets of data to each learner, whereas in FL, the learners train on locally stored datasets [5]. PL is used when the orchestrator lacks computational resources to train an effective model, and thus chooses to distribute the workload to a set of distributed learners [6]. In FL, the learners collect their own data and can take advantage of models trained on a larger dataset while keeping their own data private [6]. Both PL and FL involve a set of distributed learners, where each learner trains a model independently and in parallel, and then the orchestrator performs an aggregation process, by which these models are turned into a single, more generalizable model [7] [8].

One of the major challenging issues in PL and FL is system heterogeneity [9], as learners have varying computation and communication capabilities. In order to utilize such heterogeneous devices, any workload distribution system must have a way to determine the capabilities of these devices, whether through prior knowledge of hardware specifications, benchmarking or usage monitoring [10] [11]. Another aspect is that failing to address system heterogeneity can limit DL to the performance of its weakest learner [6] [13]. To resolve this issue, research efforts have contemplated changing the size of the learners' local models, allocating different amounts of data to each learner, and allowing learners to iterate over their local data a variable number of iterations [6] [7]. However existing schemes known to the authors assume that the learners' computation and communication capabilities are already known to the orchestrator or that the learners are willing to share this information with the latter [6] [9] [13]. In contrast, scenarios

where learners are reluctant to share such information due to privacy concerns are mostly overlooked.

In this paper, we strive to resolve the aforementioned issues by proposing the DAB scheme. DAB addresses system heterogeneity in PL by varying the amount of data allocated to each learner based on their computation and communication capabilities. It strives to eliminate staleness while preserving the privacy of learners by considering the learners' hardware characteristics as private data that cannot be shared with the orchestrator. To do so, DAB introduces a decentralized data allocation approach, where each learner determines the upper bound on the amount of data that it can process such that a certain training time threshold (i.e., deadline) is not exceeded.

In order to determine the aforementioned upper bound, DAB proposes a new benchmarking scheme, referred to as Subset Benchmarking (SB), which enables each learner to locally estimate its own computational capability. Note that most existing benchmarking schemes in ML strive to characterize learners based on the number of floating point operations that they can execute every second [14]. In contrast, the proposed SB scheme strives to accurately estimate the rate at which each learner can train a given model.

To the best of our knowledge, DAB is the first decentralized data allocation scheme in PL that eliminates staleness while preserving the learners' privacy. In addition, SB is a new benchmarking scheme that strives to accurately estimate the rate at which a learner is able to train a given model in PL. Finally, in contrast to simulation-based performance evaluations, all tests and evaluations in DAB are conducted on a real system composed of heterogeneous and distributed devices.

The remainder of the paper is organized as follows. Section II highlights some of the related work. Section III provides a detailed description of the proposed scheme (DAB). Section IV presents the performance evaluation and results. Section V summarizes our conclusions and future work.

II. Related Work

In this section, we provide an overview of some of the related work in distributed learning, as well as benchmarking.

A. Distributed Learning

The issue of system heterogeneity in DL (i.e., FL and PL) has been investigated in various works [6] [13] [15]. To resolve the issue of system heterogeneity, existing research efforts tend to change the size of the workload assigned to learners based on their capabilities, so that all learners can complete their task at the same time [13] [15]. To do that, one of the following three approaches is adopted: 1) changing the number of learning iterations that each learner performs [16], 2) changing the size of the model given to each learner [13], and 3) changing the amount of data allocated to each learner [17] [18].

In [16], learners perform a different number of local updates in each global communication cycle. However this approach tends to result in stale updates, which can affect the training accuracy [15]. Note that staleness is a metric that can be applied to two parameter updates from different learners, and that is meant to give some indication of the redundant information provided by both updates [6]. Some works have shown the benefit of allowing some staleness as a trade-off for better utilizing powerful learners with fast communication links [15] [16].

HeteroFL [13] deals with system heterogeneity by sending models of different sizes to learners with different capabilities. The orchestrator prunes the central model into different sizes using lottery ticket methods from [19]. These sub-models are sent to the learners. Upon global aggregation, the orchestrator expands all sub-models into their original size, and then aggregates them as normal. This method is resource-intensive, since the orchestrator must distill the model into parameter sets of different sizes.

In [17] and [18], significant improvements are yielded by changing the amount of data that is allocated to each learner. These methods are not computationally expensive for the orchestrator. In addition, they have the potential of eliminating staleness. However, most existing schemes that vary the amount of data allocated to each learner tend to assume that the orchestrator has a priori knowledge of the compute and communication characteristics of all the learners it recruits [6] [17] [18]. Despite facilitating centralized data allocation, which leads to more informed decisions, such schemes also tend to overlook the need for decentralized data allocation to preserve learners' privacy pertaining to their device characteristics.

In this work, we vary the amount of data allocated to each learner while fixing the number of iterations in order to eliminate staleness. In contrast to existing schemes, we propose a decentralized data allocation policy where learners' privacy is paramount. Learners' privacy is preserved by keeping any information pertaining to its characteristics local to the learner, rather than transmitting it to the orchestrator. In particular, learners only communicate with the orchestrator to download data and return their trained parameters. In addition, compute characteristics are obtained via benchmarking. In contrast to most existing schemes, performance evaluation is conducted on real edge computing environment rather than being simulation-based.

B. Benchmarking

The number of Floating Point Operations Per Second (FLOPS) that a machine is capable of has been used to characterize the training ability of a given hardware [14]. The FLOPS-based method, which is used to predict the runtimes of deep learning workloads, works by running the desired workload on a reference GPU, and then scaling the runtime by the ratios of the peak FLOPS of the reference and target

GPU. However, this method of performance modeling has been shown to be inaccurate [11].

The rate at which GPUs can perform backpropagation on Artificial Neural Networks (ANNs) is known to have a complicated relationship with esoteric hardware characteristics [20]. The present state-of-the-art in predictive performance modeling for deep learning is Habitat [11], which works exclusively with pytorch modules on CUDA-enabled GPUs. Habitat relies on information about hardware utilization and the amount of time each operation takes. It obtains this information by monkey-patching system calls, and using CUDA events [21]. While implementing Habitat on most systems is theoretically possible, doing so can take many work hours and require skilled developers. Our work uses a benchmarking paradigm that is simple, and that strives to accurately predict the training rate of static models on almost any runtime.

III. Data Allocation via Benchmarking (DAB)

In this section, we present the system model, the proposed SB scheme, and the procedure used to calculate the upper bound at each learner.

A. System Model

Consider a set of n learners that are recruited by the orchestrator in exchange of some incentives, denoted $W = \{w_1, w_2, ..., w_n\}$. Upon recruitment, each learner $w_k \in W$ independently runs two benchmarks, namely c_k and b_k. The benchmark c_k represents the compute power of learner w_k, which is the rate at which w_k can train the model in data samples per second. The benchmark b_k represents the download bandwidth of learner w_k, which is the bandwidth of the network connection that the learner has with the orchestrator in bytes per second. Given c_k and b_k, the learners then calculate the maximum number of data shards a_k that they can download and train for μ iterations before a given deadline D is reached. Similarly with other works, we assume that the channel is perfectly reciprocal within one global cycle [6]. The global update cycle of DAB operates as follows:

1) The orchestrator sends the training deadline D to each learner.
2) Each learner w_k runs benchmarks to obtain c_k and b_k.
3) Each learner w_k determines the upper bound a_k on the amount of data that it can download based on the estimated values of c_k and b_k.
4) Each learner w_k downloads a_k data shards, as well as the model parameters from the orchestrator.
5) Each learner w_k executes μ gradient updates on its local parameters.
6) Each learner w_k uploads its parameters to the orchestrator.
7) The orchestrator averages all parameters, and steps 4-7 are repeated until convergence.

The time that a learner w_k takes to download the data and the model parameters, performs learning updates, and uploads the parameters is represented by η_k. The time for the orchestrator to perform an aggregation is negligible. For any learner w_k, η_k is bounded above by the global deadline D.

Algorithm 1 : DAB at Learners

1: **Input:**
2: Number of data shards used in the benchmark n
3: Dummy model used in the benchmark $dummy\_model$
4: Training deadline D
5: Global update index G
6:
7: *data_allocation(dummy_model, n, D)*
8: **Begin**
9: $t_d^s \leftarrow curent\_time$ // t_d^s is the download start time
10: download n & record t_d^e // t_d^e is the download end time
11: $T_d = t_d^e - t_d^s$ // T_d is the download time
12: $b_k = n/T_d$ // b_k of learner w_k, $\forall w_k \in W$
13: $t_t^s \leftarrow curent\_time$ // t_t^s is the training start time
14: train $dummy\_model$ on n data shards & record t_t^e
15: $T_t = t_t^e - t_t^s$ // T_t is the training_time
16: $c_k = n/T_t$ // c_k of learner w_k, $\forall w_k \in W$
17: $numb\_shards = a_k$ // Eq. 5
18: **for all** $g \in G$ **do** // each iteration in G
19: download $numb\_shards$
20: download the most recent set of parameters from the orchestrator ρ
21: **for all** $i \in \mu$ **do** // each iteration in μ
22: train the client model on the downloaded data using ρ
23: send parameters back to the orchestrator to be aggregated
24: return training parameters
25: **End**

B. Subset Benchmarking (SB)

As demonstrated in Algorithm 1, to estimate the communication capability of devices in SB, each learner w_k downloads a predetermined number, n, of randomly selected data shards. Given the amount of time it takes to download the data shards, the learner can then infer its download speed b_k (lines 9-12). To estimate the compute capability of devices, each learner w_k trains a dummy model on this training data while measuring the rate at which gradient updates are performed to estimate c_k (lines 13-16). It is important that this dummy model has a matching architecture to the network being trained, but that the parameters are kept separate. We call our benchmarking method subset benchmarking since this task is meant to represent a subset of the training task.

SB exploits the iterative nature of backpropagation, where training a neural network is essentially repeating the same process for each batch in an epoch, and for every epoch until convergence. Thus, we benchmark learners by running a subset of the learning task on the learner, and extrapolating the runtime to the whole workload.

C. Calculating the Upper Bound

Once a learner w_k estimates its two benchmarks, namely c_k and b_k, it calculates the upper bound on the amount of data it can train within the deadline D. To determine this bound, each learner w_k first estimates the amount of time it takes to download the model parameters and the training data, which is denoted α_k and is given by Eq. 1, where Ω_m represents the

TABLE I
HARDWARE CHARACTERISTICS OF THE TESTBED DEVICES

Device	Clock Speed	Cores	Memory	Network
2070 Super	1605 MHz	2560	8 GB	Ethernet
Jetson Nano	920 MHz	128	2 GB	Ethernet
Pi1	1.4 GHz	4	512 MB	WiFi
Pi2	1.8 GHz	4	4 GB	WiFi
Pi3	1.5 GHz	4	4 GB	WiFi

size of the model parameters in bytes, Ω_d represents the size of each data shard in bytes.

$$\alpha_k = \frac{\Omega_m + \Omega_d a_k}{b_k} \tag{1}$$

Each learner w_k then estimates the amount of time it takes to train the model on a_k data samples for μ iterations, which is denoted β_k, and is given by Eq. 2.

$$\beta_k = \frac{\mu a_k}{c_k} \tag{2}$$

The time that each learner w_k takes to upload its data, denoted γ_k, is then estimated as given by Eq. 3.

$$\gamma_k = \frac{\Omega_m}{b_k} \tag{3}$$

The total training time that a learner w_k takes is the sum of the time it takes to download the model parameters and the training data, train the model on a_k data samples for μ iterations, and upload its data. This total training time is denoted as η_k, and is the sum of α_k, β_k, and γ_k, as given by Eq. 4

$$\eta_k = \alpha_k + \beta_k + \gamma_k = \frac{\Omega_m + \Omega_d a_k}{b_k} + \frac{\mu a_k}{c_k} + \frac{\Omega_m}{b_k} \tag{4}$$

Each learner is required to finish μ training iterations before a certain training deadline D. Thus, the total training time η_k that each learner w_k takes must be less than D (i.e., $\eta_k < D$). As shown in Algorithm 1, given their benchmarking scores b_k and c_k, each learner calculates the maximal a_k that enables it to finish μ training iterations before D (line 17). Such a value for a_k is obtained as given by Eq. 5.

$$a_k < \frac{D - \frac{2\Omega_m}{b_k}}{\frac{\mu}{c_k} + \frac{\Omega_d}{b_k}} \tag{5}$$

Once a_k is determined, each learner w_k repeats the following steps for G number of global iterations (line 18): 1) w_k downloads a_k data shards, as well as the most recent model parameters from the orchestrator (lines 19 & 20), 2) w_k executes μ gradient updates on its local parameters (lines 21 & 22), and 3) w_k sends its parameters to the orchestrator (line 23), which aggregates the received parameters and sends the average values back to all the learners.

IV. PERFORMANCE EVALUATION

In this section, we evaluate the performance of DAB compared to a representative of staleness-aware centralized data allocation schemes, which is referred to as the Centralized Staleness-Aware Data Allocation (CSA) scheme [6]. We also evaluate the performance of SB compared to the baseline FLOPS benchmarking scheme [14]. We use the following performance metrics: 1) the average benchmarking runtime error, which is calculated as the average of the ratio of the difference between the predicted runtime and the actual runtime rendered by each learner to train the network, to its actual runtime, 2) the loss of the trained network, which is calculated as the categorical cross entropy, and 3) the prediction accuracy, which is calculated as the ratio of correct predictions to the total number of predictions.

A. Experimental Setup

We implement DAB, CSA, SB, and FLOPS on a real testbed composed of three Raspberry Pi devices, as well as one 2070 Super, and one Jetson Nano. The hardware characteristics of each one of these devices are presented in Table I. The implementation is done using our custom-built Python framework that we refer to as Axon [12]. Experiments are conducted by training a feed-forward neural network as a classifier on the MNIST dataset [22]. The training set is composed of 60,000 greyscale images of handwritten digits at 28x28 resolution, with corresponding labels numbered zero through nine and is split into data shards of 500 samples each. The network architecture, which we refer to as TwoNN, is a feed-forward network with layer widths [784, 50, 20, 10], each of which is activated using the ReLu activation function, except the last layer which uses the softmax activation function. We train the network using the Stochastic Gradient Descent (SGD) optimizer with a learning rate of 0.1. The number of learning iterations, μ, is set to five, whereas the training deadline, D, is set to 15 seconds. The FLOPS of a learner is calculated by multiplying two 512x512 matrices 100 times, and keeping track of the time for the whole computation.

B. Results and Analysis

We evaluate SB compared to FLOPS in terms of the average runtime benchmarking error over varying numbers of data shards used in the benchmark, ranging from 2 to 8. The dummy model used in this experiment is a neural network that is trained on an amount of data of 120 shards at the 2070 Super and Jetson Nano devices, and 15 data shards at each of the Raspberry Pi devices. Note that the architecture of the neural network used is the same as the one stated above. As depicted in Figure 1, SB yields a significant reduction in benchmarking error, reaching up to 83%. This can be attributed to the fact that FLOPS restricts its estimations of the training time to the rate at which the hardware can perform floating point operations. However, the performance of various platforms in training neural networks is highly difficult to predict, since it

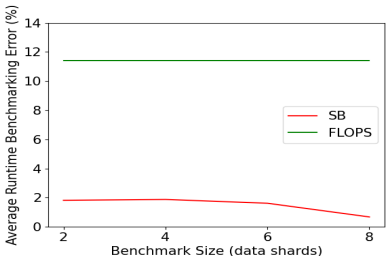


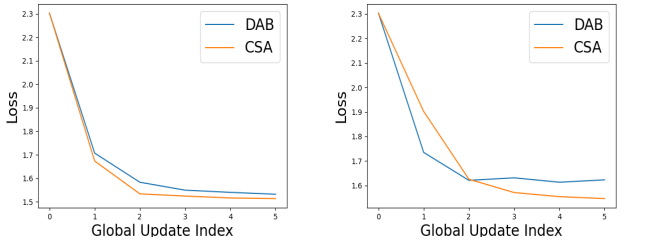
Fig. 1. The average runtime benchmarking error of SB and FLOPS over varying benchmark size (data shards).

involves many bottlenecks other than what FLOPS is restricted to.

In contrast, SB goes beyond such a restriction by using a dummy model that mimics the actual ML model that needs to be trained, thus timing the execution of the process itself, rather than developing a model for it. Note that since FLOPS does not use data shards, it remains constant. In contrast, as the size of data shards (i.e., benchmark size) increases, the benchmarking runtime error in SB decreases. This is because the more data shards that the dummy model is trained on, the more samples the average score is made from, which makes it closer to the actual training rate.Performance does not significantly improve when benchmark size is increased above 8 shards. It is worth mentioning that while SB has high accuracy, it is not generic. A learner's benchmarking score is specific to a single model architecture and cannot be used to predict that learner's training rate for other models.

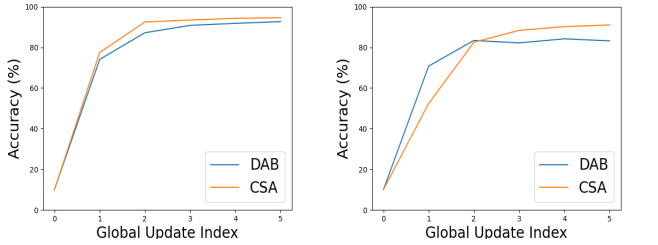
We evaluate the performance of DAB compared to CSA in terms of loss and accuracy over varying global update index on two clusters; a cluster of two devices, Pi1 and Jetson Nano, and a cluster of four devices, Pi1, Pi2, Pi3, and Jetson Nano. At each global update, the orchestrator averages the parameters of the models from each learner, and assesses the performance of the averaged parameters.

As depicted in Figure 2(a), the loss yielded by DAB while being trained on two devices closely approaches that yielded by CSA. This is despite being a decentralized scheme, thus lacking the global view of the hardware characteristics of the entire learners that CSA possesses. This can be attributed to the fact that DAB eliminates staleness by fixing the number of iterations, whereas CSA strives to minimize staleness under varying number of iterations. In addition, although the orchestrator does not have a global knowledge of the learners' hardware characteristics in DAB, it has a global knowledge of the upper bound of the data that each learner can download while abiding to the required training deadline. Note that DAB gets closer to CSA, with almost a 0% gap as the global update index decreases, whereas the gap starts to slightly increase by up to 6%, as the global update index increases. As the global update index increases, the leverage gained by CSA due to optimizing the number of learning iterations that learners



(a) Loss on a cluster of Pi1 and Jetson Nano (b) Loss on a cluster of Pi1, Pi2, Pi3 and Jetson Nano

Fig. 2. Loss of DAB and CSA over varying global update index on two clusters.



(a) Accuracy on a cluster of Pi1 and Jetson Nano (b) Accuracy on a cluster of Pi1, Pi2, Pi3 and Jetson Nano

Fig. 3. Accuracy of DAB and CSA over varying global update index on two clusters.

perform starts to manifest.

We conduct the same experiment to assess the loss of DAB compared to CSA on a cluster of four devices. As shown in Figure 2(b), as the global update index decreases, DAB outperforms CSA by up to 12%, whereas it starts to yield a slightly higher loss than CSA, reaching 5%, as the number of global update index increases. DAB outperforms CSA at lower global update index because CSA operates on a fixed number of data samples, which is split between learners, whereas DAB enables each learner to determine the maximum amount of data that it is able to process in the given time frame (with duplication if needed). This means that in the presence of extra training resources, DAB trains the neural network on more data per global update than CSA does, which is why it outperforms CSA when provided with four training devices but not when only two devices are available. As the global update index increases, CSA outperforms DAB, since the leverage gained by CSA due to determining the optimal number of iterations becomes more evident. Note that this requires a global knowledge that DAB does not provide for privacy reasons.

We perform the same aforementioned experiments to assess the prediction accuracy of DAB and CSA. A corresponding behavior to the loss is demonstrated in the accuracy plots in Figures 3(a) and 3(b). In the first cluster of two devices, depicted in Figure 3(a), DAB closely approaches CSA, with

almost a 0% gap as the global update index decreases, whereas the gap starts to slightly increase by up to 2% as the global update index increases. In particular, DAB reaches a prediction accuracy of 93% compared to 95% in CSA. This is because the loss of the network trained with DAB is higher than the network trained with CSA. As depicted in Figure 3(b,) in the cluster of four devices, DAB outperforms CSA by up to 26% as the global update index decreases, whereas it renders an 8% lower accuracy than CSA as the global update index increases. This can be attributed to the inverse behavior in terms of loss depicted in Figure 2(b).

V. Conclusion and Future Work

In this paper, we have proposed the decentralized DAB scheme that eliminates staleness in PL by fixing the number of iterations among all learners. DAB does so while preserving the privacy of learners by refraining from sharing any information about their hardware characteristics with the orchestrator. Instead, each learner estimates its own capabilities, and uses these estimations to determine the upper bound on the amount of data that it can train, such that a certain training deadline is not exceeded. In addition, we have proposed a new benchmarking scheme in PL, called SB, to allow each learner to estimate its own hardware characteristics.

Performance evaluations have been conducted on a real testbed, composed of multiple heterogeneous and distributed devices. Extensive performance evaluations have shown that SB significantly outperforms FLOPS by up to 83% in terms of runtime benchmarking error. It has also been shown that DAB outperforms CSA in terms of loss and prediction accuracy by up to 12% and 26%, respectively, as the global update index decreases, whereas the latter outperforms DAB by up to 5% and 8%, respectively, as the global update index increases. In the future, we plan to develop a dynamic data allocation scheme that responds to the changes that occur in the learners' capabilities, and that handles malicious learners submitting fraudulent upper bounds to the orchestrator.

VI. Acknowledgement

This research is supported by a grant from the Natural Sciences and Engineering Research Council of Canada (NSERC) under grant number: ALLRP 549919-20.

References

[1] W. Y. B. Lim et al., "Federated Learning in Mobile Edge Networks: A Comprehensive Survey," in IEEE Communications Surveys & Tutorials, vol. 22, no. 3, pp. 2031-2063, third quarter 2020, doi: 10.1109/COMST.2020.2986024.

[2] K. Gyarmathy, "Comprehensive Guide to IoT Statistics You Need to Know in 2020," 2020. [Online]. Available: https://www.vxchnge.com/blog/iot-statistics

[3] Rhea Kelly, "Internet of Things Data To Top 1.6 Zettabytes by 2020 – for 5G Mobile Networks and Beyond (IEEE ICC'20 Workshop - Campus Technology," 2015. [Online]. Available: https://campustechnology.com/articles/2015/04/15/internet-of-things-data-to-top-1-6-zettabytes-by-2020.aspx

[4] Y. Mao, C. You, J. Zhang, K. Huang and K. B. Letaief, "A Survey on Mobile Edge Computing: The Communication Perspective," in IEEE Communications Surveys & Tutorials, vol. 19, no. 4, pp. 2322-2358, Fourth quarter 2017, doi: 10.1109/COMST.2017.2745201.

[5] J. Konecny, H. B. McMahan, D. Ramage, "Federated Optimization: Distributed Optimization Beyond the Datacenter", CoRR, vol. abs/1511.03575, 2015.

[6] U. Mohammad, S. Sorour, "Asyncronous Task Allocation for Federated and Parallelized Mobile Edge Learning", 2020. [Online] Available: https://arxiv.org/abs/1905.01656

[7] Yu, H., S. Yang, S. Zhu. Parallel restarted sgd with faster convergence and less communication: Demystifying why model averaging works for deep learning. In Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, pages 5693–5700. 2019.

[8] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konecný, S. Mazzocchi, H. B. McMahan, T. V. Overveldt, D. Petrou, D. Ramage, J. Roselander, "Towards Federated Learning at Scale: System Design," ArXiv, vol. abs/1902.01046, 2019

[9] T. Li, A. K. Sahu, A. Talwalkar and V. Smith, "Federated Learning: Challenges, Methods, and Future Directions," in IEEE Signal Processing Magazine, vol. 37, no. 3, pp. 50-60, May 2020, doi: 10.1109/MSP.2020.2975749.

[10] M. A. Iverson, F. Ozguner and L. Potter, "Statistical prediction of task execution times through analytic benchmarking for scheduling in a heterogeneous environment," in IEEE Transactions on Computers, vol. 48, no. 12, pp. 1374-1379, Dec. 1999, doi: 10.1109/12.817403.

[11] G. X. Yu, Y. Gao, P. Golikov, G. Pekhimenko, "A Runtime-Based Computational Performance Predictor for Deep Neural Network Training", USENIX Annual Technical Conference 2021: 503-521

[12] D. Mays, "Axon-ECRG," 2021. [Online] Available: https://github.com/DuncanMays/axon-ECRG

[13] E. Diao, J. Ding, V. Tarokh, "HeteroFL: Computation and Communication Efficient Federated Learning for Heterogeneous Clients" 2021. [Online] Available: https://arxiv.org/abs/2010.01264

[14] R. Tang, A. Adhikari, J. Lin, "FLOPs as a Direct Optimization Objective for Learning Sparse Neural Networks", 2018, ArXiv

[15] Z. Wei, S. Gupta, X. Lian, and J. Liu, "Staleness-Aware Async-SGD for distributed deep learning," IJCAI International Joint Conference on Artificial Intelligence, vol. 2016-Janua, pp. 2350–2356, 2016.

[16] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, "When Edge Meets Learning : Adaptive Control for Resource-Constrained Distributed Machine Learning", in INFOCOM, 2018. [Online]. Available: https://researcher.watson.ibm.com/researcher/files/us-wangshiq/SW INFOCOM2018.

[17] U. Mohammad and S. Sorour, "Adaptive Task Allocation for Mobile Edge Learning," in 2019 IEEE Wireless Communications and Networking Conference Workshop (WCNCW). IEEE, apr 2019, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/document/8902527/

[18] U. Y. Mohammad, S. Sorour, and M. S. Hefeida, "Task allocation for mobile federated and offloaded learning with energy and delay constraints", in IEEE ICC 2020 Workshop on Edge Machine Learning for 5G Mobile Networks and Beyond (IEEE ICC'20 Workshop - EML5G), Dublin, Ireland, Jun. 2020.

[19] J. Frankle, M. Carbin, "The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks", 2019. [Online]. Available: https://arxiv.org/abs/1803.03635

[20] H. Zhu et al., "Benchmarking and Analyzing Deep Neural Network Training," 2018 IEEE International Symposium on Workload Characterization (IISWC), 2018, pp. 88-100, doi: 10.1109/IISWC.2018.8573476.

[21] NVIDIA Corporation. CUDA Runtime API - Event Management, 2019. https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__EVENT.html

[22] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," in Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, Nov. 1998, doi: 10.1109/5.726791.