Toward Scalable and Robust AIoT via Decentralized Federated Learning

Pinyarash Pinyoanuntapong, Wesley Houston Huff, Minwoo Lee, Chen Chen, and Pu Wang

Abstract

As Artificial Intelligence of Things (AIoT) has become increasingly important for modern AI applications,federated learning (FL) is envisioned to be the enabling technology for AIoT, especially for large-scale, data privacy-preserving scenarios. However, most existing FL is managed in a centralized manner (CFL), which confronts the limitations of scalability given the AIoT device explosion. The key challenge faced by CFL is the communication bottleneck at the central model aggregation server, which leads to a high server-to-worker communication delay and thus severely slows down the model convergence. To address this challenge, this article introduces a generic decentralized FL (DFL) framework that can operate in either synchronous (Sync-DFL) mode or asynchronous (Async-DFL) mode to alleviate the high communication congestion around the central server. Moreover, Async-DFL is the first DFL in the literature to provide a generic FL framework that is fully asynchronous and able to completely avoid worker waiting, which leads to robust distributed model training in the inherently heterogeneous IoT environments, where stragglers (i.e., slow devices) are very common due to the largely varying computing/networking speeds of IoT devices. Our DFL framework is implemented, deployed, and experimented with in both simulation and physical testbeds. The results show that Async-DFL can accelerate the convergence speed of model training twice as fast as CFL, while maintaining convergence accuracy and effectively combating the impact of the stragglers.

Introduction

The Internet of Things (IoT) is a network of a variety of things or objects that are able to interact with each other and cooperate with their neighbors to reach common goals [1]. Advances in wireless communication (e.g., 5G) and artificial intelligence (AI) have created a synergistic move toward Artificial Intelligence of Things (AIoT) [2] , which consists of AI-empowered IoT devices that can analyze data used within devices and make proactive, intelligent, and accurate decisions without the involvement of humans. AIoT applications such as smart surveillance camera networks, intelligent transportation, smart and connected healthcare, smart home, and smart grids have paved the way to build smart cities. Toward this, a myriad of smart edge devices need to be installed, and a large number of AIoT devices produce enormous data. The rapid growth of the size of the data and the number of devices need AI-driven automated processing, often done in a centralized manner. The centralized machine learning model needs the training data to be collocated at a common server, and therefore needs to transfer a large amount of IoT device data from the network edge to the central server. This imposes a huge burden on the communication networks while inducing severe vulnerability of data privacy.

Federated learning (FL) [3] is an emerging privacy-preserving deep learning paradigm that enables distributed neural model training on edge devices while keeping their data local to prevent privacy leakage. In FL, the workers (e.g., IoT devices) only need to send their local model updates to the server that aggregates these updates to continuously improve the shared global model. This approach significantly reduces the data privacy risk by only sending and receiving the computed local models to the server or vice versa rather than sending the data itself. Moreover, FL can greatly reduce the required number of communication rounds for model convergence by increasing computation parallelization, where more IoT devices are involved as workers, and by increasing local computation, where a worker performs multiple iterations of model updates before sending the updated model to the server. Through FL, IoT devices can still learn much more accurate models with small local datasets.

Classical FL relies on frequent centralized model aggregation, which may be applied for a single-hop IoT network, where the IoT devices are connected to each other and the central server over single-hop cellular or WiFi connections. Different from single-hop IoT systems that rely on cellular/WiFi systems with high infrastructure deployment and operational costs, wireless multihop IoT networks, consisting of a mesh of interconnected wireless IoT devices, have been widely exploited to build cost-efficient large-scale IoT systems [1]. However, adopting centralized FL (CFL) over a multihop IoT network faces a significant challenge: a communication bottleneck at the central server. As the number of IoT devices increases, the full network load is forced to pass through the single server node, which leads to network congestion and thus greatly slows down model convergence. In addition, the server carries the full responsibility of aggregation and has an unavoidable single point of failure.

Decentralized FL (DFL) [4, 5] is an emerging FL paradigm that can effectively address the aforementioned limitations of CFL. Under DFL, in each epoch, all workers update their local models via multiple stochastic gradient descent (SGD) iterations. Then each worker averages its local model only with its neighbors. By removing the single point of aggregation, DFL can lead to a robust learning framework with increased scalability. Moreover, distributing the traffic load from a central node can maximize the utilization of a network's bandwidth and accelerate the model convergence speed.

Despite its great advantages, existing DFL solutions do have compromises and areas for possible improvement. A common DFL paradigm is synchronous optimization, where the workers from a local group (e.g., one-hop neighbors) must receive the models from each other and finish their respective model averaging before stepping into the next epoch. For example, HADFL [5] defines a synchronization topology containing work-

Pinyarash Pinyoanuntapong, Wesley Houston Huff (corresponding author), Minwoo Lee and Pu Wang are with the University of North Carolina at Charlotte, USA; Chen Chen is with the University of Central Florida, USA.

Digital Object Identifier: 10.1109/IOTM.006.2100216

 2576-3180/22/$25.00 © 2022 IEEE

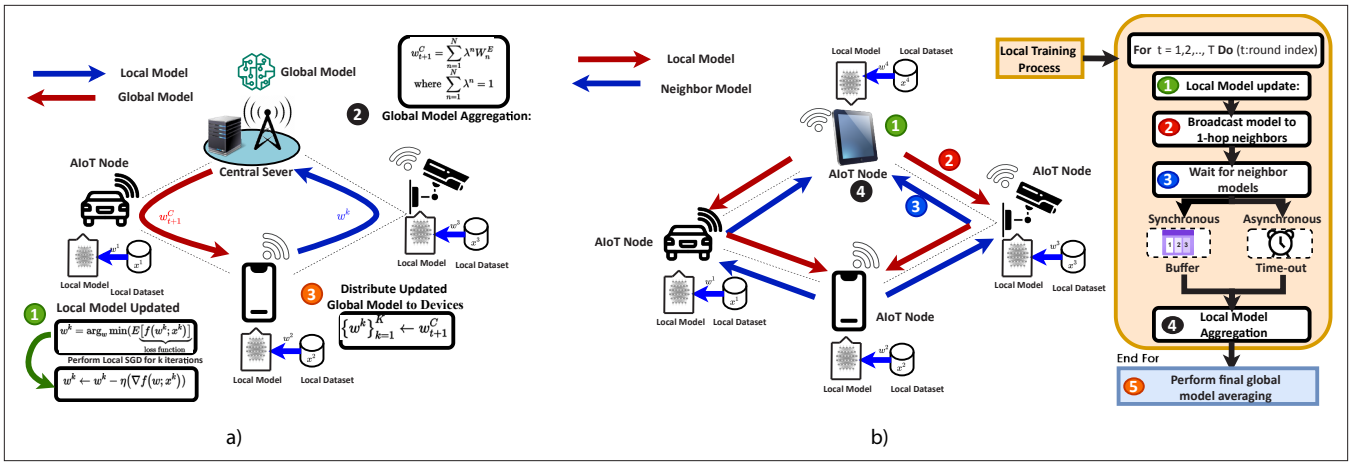

FIGURE 1. Federated learning in an IoT network: a) classical centralized FL;b) decentralized FL.

ers that need to synchronize their model updates and aggregation. The probability-based worker selection is further adopted to mitigate the impact of stragglers. Moreover, HADFL needs to enforce model consensus by letting the synchronized workers broadcast their models to other non-synchronized workers. The collaborative FL algorithm [6] lets the workers, which cannot reach the central server directly, perform synchronized distributed FL with their one-hop neighbors, while the other workers still follow CFL. This synchronous nature, however, makes FL very vulnerable to stragglers (i.e., slow workers), which can greatly slow down the overall model convergence speed.

In this article, we propose the first composable generic DFL framework, which can operate in either synchronous mode or asynchronous mode. In particular, the asynchronous DFL mode can avoid the communication bottleneck of CFL via decentralized local model updates while mitigating the impact of stragglers via asynchronous model averaging. In the preliminary experiments, we demonstrate the superior convergence performance of DFL compared to CFL and showcase the robustness of asynchronous DFL in the presence of stragglers.

References [7, 8] provide a comprehensive survey of the challenges and solutions for applying FL algorithms over wireless communication networks, such as compression, wireless resource management, FL algorithm design, over-the-air computation, and privacy/security issues. However, these works do not provide detailed discussion and experimental evaluations on the distributed FL algorithms that are essential for AIoT applications. Our article aims to generalize the algorithmic foundation of DFL by proposing a simple framework that can easily transform into synchronous or asynchronous DFL algorithms with minimal code change. Moreover, to the best of our knowledge, our asynchronous DFL (Async-DFL) is also the first FL framework in the literature that is fully asynchronous, without requiring any wait time. The most relevant work to our Async-DFL is the asynchronous decentralized parallel stochastic gradient descent (AD-PSGD) algorithm [9]. However, there are two major differences between Async-DFL and AD-PSGD. First, AD-PSGD is actually a semi-asynchronous solution, and Async-DFL is a fully asynchronous scheme. In particular, AD-PSGD requires each worker to wait for the local model from one of its neighbors and then performs model aggregation. With Async-DFL, each worker does not need to wait for the model from any of its neighbors, and it opportunistically aggregates the local models it has already received. Second, Async-DFL is federated in the sense that it relies on the edge devices, local data, and production networks to perform distributed model training. Therefore, similar to FedAvg, Async-DFL exploits local SGD to improve communication efficiency, where each worker updates its local model by performing multiple rounds of SGD before sending it to the aggregation server. Meanwhile, Async-DFL needs to achieve model convergence under non-independent and identically distributed (IID) data distribution. On the contrary, AD-PSGD is designed for distributed deep learning in data centers, where the communication cost is not an issue and the data across all the workers are IID. Therefore, AD-PSGD only needs each worker to update its local model with SGD for one round and then sends the updated model or gradient to the parameter server over a high-speed communication link.

Centralized Federated Learning

One of the key challenges of machine learning is finding the best way to make use of as much data and processing power to train the learning model in the shortest possible time. Over time, it has been found that it is more cost- and time-effective to divide the work among many cheaper and weaker edge devices than try to place it all on a single expensive, powerful machine. In the past, traditionally this form of machine learning relied on a central server to read the data of IoT edge devices. This has been able to provide a wealth of data on which models can train; howeer, it not only places a great computational load on the central server but also completely opens the edge devices to critical privacy concerns whose relevance in real-world problems is ongoing.

The FL paradigm changes this. Its operation can be broken down into two key elements: the workers and the aggregator(s). In FL, the IoT device's data itself is never read or taken by a source from outside the device. Rather, each device takes on the shared responsibility of training, with each worker device having its own local model being trained. Aggregator devices are central servers responsible for receiving the models of workers from across the network, then summing, averaging, and/or refactoring them to a composite new model, and then broadcasting across the network for the next round of training.

As shown in Fig. 1a, the de facto CFL algorithm, FedAvg [10], and many of its variants are designed to handle distributed training of a common neural network, which can be modeled as a distributed parallel non-convex optimization problem with the training loss function as the optimization objective. To solve such a problem, each worker begins with the same initial untrained model, and then alternates between local SGD iteration and global model averaging for multiple (server-worker communication) rounds/epochs. During local SGD iteration, the worker tries to reduce its training loss $F(w)$ by performing H_k mini-batch SGD iterations with each iteration updating the local model weights. After finishing local model updates, the workers send their local models w_k $k \leq K$ to the central server, which averages them to produce the updated global model accordingly. The new global model is broadcast to the workers, and the above procedure is repeated.

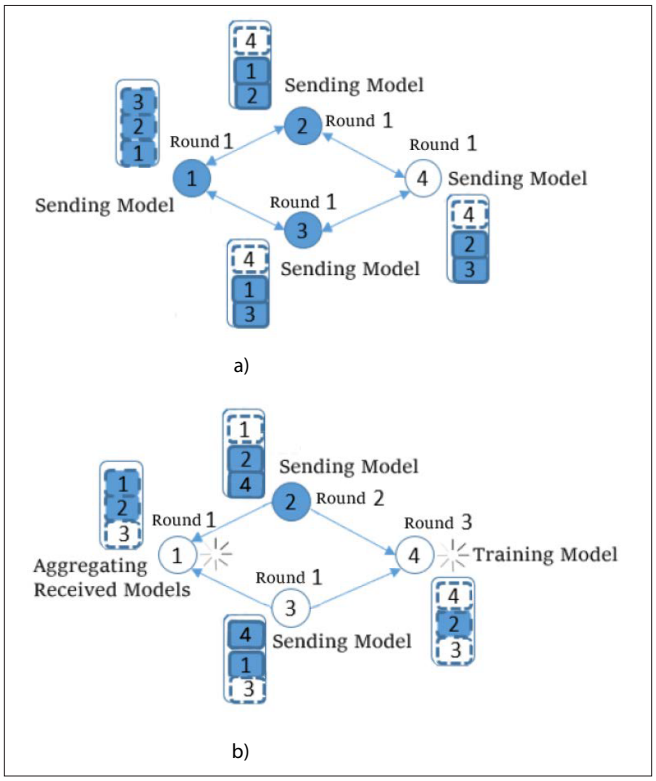

FIGURE 2. In synchronous DFL, workers keep time by waiting for the slowest workers to fill their buffers before aggregating. In asynchronous DFL, the faster workers aggregate immediately no matter how full the buffer is, which allows them to avoid waiting but necessitates program robustness to handle communication at any time: a) synchronous DFL; b) asynchronous DFL.

Challenges of CFL

CFL serves as the next step, shifting from the traditional centralized paradigm to distributed machine learning; however, its centralized model aggregation nature still presents problems. As with any centralized approach, CFL presents an unavoidable communication bottleneck in the system. Although the computational load of training is distributed, the network traffic is still concentrated at one point, which places a strain on the central server's throughput and does not fully utilize the bandwidth across the network. In addition, CFL often has to contend with multihop communication, which further reduces its performance. With that being the case, CFL favors topologies like star networks that allow them to have all workers send their models in one hop; however, that type of network is inflexible and scales poorly compared to IoT mesh networks, which are more practical in real-world applications, leaving CFL to fare poorly in development.

Decentralized Federated Learning

Generic Decentralized FL Framework:

With our DFL, each worker communicates with one-hop neighbors and shares the responsibility of aggregation as shown in Fig. 1b. Rather than sending its local model to a central server node, the worker sends copies of its model to its network neighbors. This means that each worker is the aggregator for their own local model and those of their neighbors, without bearing the full responsibility of every worker's model at once. Critically, DFL exploits the benefits of fast single-hop wireless connections, as opposed to the multihop variants of CFL. The ability of the workers to operate without needing knowledge of a broader network or the supervision of a central server allows the framework a large degree of flexibility, robustness, and scalability.

As shown in Fig. 1b, we propose a generic DFL framework that can be easily turned into either the synchronous DFL (Sync-DFL) mode pr asynchronous DFL (Async-DFL) mode. Sync-DFL and Async-DFL share almost the same local training procedure that is iterating between local model update and local model aggregation, including:

- Model updates via local SGD
- Model broadcasting among one-hop neighbors
- Model receiving from neighbors
- Local model aggregation/averaging of its own model and received neighbor models

The above local training procedure is repeated until the predefined number of training epochs is reached. After the local training is done, each worker sends its local model to the global aggregation node that performs the last-round global model aggregation on the received local models and produces the final inference model used by IoT devices. The global aggregation node can be the gateway device that connects IoT networks to the Internet or a randomly selected regular IoT device.

The difference between Sync-DFL and Async-DFL modes comes from two perspectives. First, they adopt different local model aggregation triggers. In Sync-DFL mode, local model aggregation begins whenever the model buffer is full, that is, the models from all one-hop neighbors arrive. Therefore, combined with the chaining effect, the training clocks (i.e., training rounds) of all workers in the network are synchronized. In Async-DFL mode; local model aggregation is triggered whenever the predefined waiting time has passed. Therefore, each worker has to maintain its own training round counter. In this case, a slower worker may receive multiple updated local models from a faster worker that has already finished multiple rounds of training. In this case, the slower worker will only keep the most up-to-date one in the buffer. Similarly, the faster worker may receive a very "old" local model from a slower worker that is still in its early training round. Second, they adopt different last-round global aggregation strategies. In Sync-DFL mode, all workers synchronize their local training so that the global aggregator can receive the models from all workers almost simultaneously. Therefore, all the local models are used for the final global model aggregation. In Async-DFL mode, the workers finish local training at different time instances. Therefore, the global aggregator will only wait for the arrivals of a certain percentage of local models and then perform last-round global model aggregation.

One of the key features of the proposed framework relies on the model buffer design. The model buffer has two purposes. First, it enables implicit inter-device synchronization for the Sync-DFL mode, which is explained in the following section. Second, it can be used to achieve the flexible speed-quality trade-off between training convergence speed and model training accuracy/quality. Each model represents a different subset of training data used to improve the overall training, so the more models aggregated the better. For the synchronous case, some workers must spend time waiting for each round of training, but the quality of each training round is maximized in doing so. For the asynchronous case, the speed-quality trade-off can be adjusted based on how many local models arrive at the buffer before the local model aggregation is performed.

Synchronous DFL

With Sync-DFL, the workers follow two converse rules: first, they only aggregate once they have a full buffer, and second, they will always wait for their stragglers before beginning their own next round of training. Essentially, the worker's buffer functions as a clock to keep time with the rest of the network, as illustrated in Fig. 2a. Each worker works in the same global round and generally begins their model broadcasting at a similar

time. Once each worker finishes their broadcasting, they begin checking to see whether their buffer is full. If the buffer is not full, the worker knows that it is outpacing the network and will wait until the buffer is full to proceed. Therefore, the workers aggregate within a similar time, emptying their respective buffers and then moving on to the next round's training.

Asynchronous DFL

Async-DFL subverts the approach of Sync-DFL. Rather than having workers wait to ensure an ideal or close-to-ideal aggregation, it instead has the workers proceed regardless of the state of the buffer. It will always have at least its own model to aggregate even if the buffer is otherwise empty. Optionally, it may have an adjustable wait timer for other workers to catch up. Over time, the faster workers begin to leave the slower workers behind, leading to the workers sharing their models with their neighbors regardless of what epoch step they are in, as shown in Fig. 2b. Eventually, they will end up with the workers training on different epochs until the stragglers are left to finish training on their own without further broadcasts from their finished neighbors.

The Async-DFL method allows the faster workers to proceed without waiting for the stragglers. Moreover, with this asynchronous design, the computing and networking can operate in parallel, where the model aggregation is carried on while the devices are sending and receiving the models. The reduced model sharing makes Async-DFL more subject to overfitting, which may worsen the per-epoch improvements to accuracy to the global testset. However, as long as the worker is aggregating at least a partially full buffer, it will counteract overfitting as it converges, and in practice that is usually the case. As the worker communication is generally the limiting factor for overall epoch speed as opposed to the actual model training, avoiding unnecessary waiting or redundant communication is important to optimize convergence speed.

The Async-DFL method allows the faster workers to proceed without waiting for the stragglers.

Simulation and Physical Testbed Evaluation of FL Management Modes

In our set of experiments, we investigate and study how Sync-DFL and Async-DFL efficiently improve the convergence speed and performance of FL in both a simulated environment and a live network testbed. Our results are based on a set of experiments in which we first ran a model training on the network in the centralized as well as decentralized, synchronous and asynchronous cases.

Experiment Setup

Heterogeneous Data Settings and Models: We use the federated learning benchmark datasets FEMINST with 62 classes from LEAF (https://leaf.cmu.edu/) and CIFAR-10 (https://www.cs.toronto.edu/~kriz/cifar.html) with 10 classes. We followed the non-IID data settings with realistic partition method from LEAF. For CIFAR-10, we use the Dirichlet distribution Dir(b) to create heterogeneous data partitions for all the workers in unbalanced settings. The degree of heterogeneity is controlled by the value of beta, which is 0.5 in our case. We model the computation/communication heterogeneity by adding 40 s training delay for the stragglers. Our CNN is composed of two convolutional layers followed by a fully connected layer that utilize local SGD. The convolutional layers contain 32 and 64 layers, respectively and are attached via a 2×2 max pooling layer. The fully connected layer contains 128 units with ReLU activation and outputs into the final layer as fully connected with softmax activation, whose size is around 5.8 MB, which represents low communication traffic. Then we evaluate a CIFAR-10 dataset with a deep neural network model, MobileNet [11], whose size is about 15 MB, representing high traffic volume.

CFL Baseline and DFL Implementations: We adopt our custom-designed FedEdge experiential framework [12] to implement the CFL, Sync-DFL, and Async-DFL solutions. We utilize the widely adopted FedAvg [10] as the CFL baseline. FedEdge [13, 14, 12] is a software-defined experiential framework with an integrated federated computing module empowered by TensorFlow and a software-defined wireless multihop networking module based on Mininet-wifi [15].

FedEdge is the first experimental framework in the literature for FL over multihop wireless edge computing networks (e.g., IoT networks), which allowed us to quickly prototype, deploy, and evaluate novel FL algorithms along with machine-learning-based system optimization methods in both simulated and real wireless devices. The experimental framework, FedML [10], has been used in exploring the possibilities of Sync-DFL. However, given that decentralized FedML's primary purpose is benchmarking rather than development, it is not designed for a live running environment for its results or set up with a network topology common to real-world applications in their experiments. Moreover, FedML does not support the generic DFL framework that enables both Sync-DFL and Async-DFL.

Physical Testbed Setup: The multihop wireless edge computing network is used as the physical IoT network testbed [12]. This testbed consists of 10 wireless edge computing nodes (i.e., IoT workers), where each node includes a wireless embedded router for communication and an Nvidia Xavier node for computing. The wireless routers are connected with three wireless interface cards to enable the multi-radio wireless node. Each mesh router is in mesh point (MP) mode, with fixed 2.4 and 5 GHz channel, 20 MHz channel width in 802.11ac operating mode, and 15 dBm transmit power. The state-of-the-art Batman-adv is used for distributed multihop routing, which establishes the server-to-worker in the Sync-DFL case. We connect a server to R1, and worker 9 served as a straggler with a delay of 40 s. Each experiment runs for 30 global epochs and 5 local rounds with a batch size of 10 and a learning rate of 0.002 for both CNN (2Conv +2FC) and MobileNet models.

Network Simulation Setup: We first evaluated the performance of the FL solutions in a FedEdge simulator [14]. To validate and analyze the performance of the generic decentralized FL framework, we established a wireless multihop IoT network with 5×3 grid topology with 15 workers, as shown in Fig 3a. The link bandwidth is set to be 24 Mb/s. Every worker's neighbors were the workers one hop away cardinally. For centralized FL, worker 1 served as the network's server node while also still serving as a worker. In our straggler case experiments for the CIFAR10 dataset, we treat worker 14 (5 network hops from worker 1) as the straggler by adding 40 s delay to each training epoch to simulate the limited computing power of IoT devices or a CPU performance drop. Each run includes 5 local rounds per global epoch with a batch size of 10, a learning rate of 0.002, and 50 global rounds.

Performance and Communication Comparisons

We evaluate the model convergence by observing the learning curves and the wall clock time when the testing accuracy achieves certain thresholds (0.5, 0.6, 0.7, and 0.75), where all methods can achieve the same maximum accuracy of 0.75 (Fig. 4). Table 1 summarizes the convergence time for all methods to achieve the maximum testing accuracy. Note that the com-

putation time relative to the overall convergence time is actually negligible; the vast majority of convergence time is spent by nodes either sending models or waiting to receive a particular model.

Model Convergence without Stragglers: Figures 4a–4c present the performance comparison of FL in terms of accuracy and wall clock convergence time of CFL, Sync-DFL, and Async-DFL with no straggler case. We performed two sets of experiments in the testbed. First, we consider LEAF dataset and CNN (2 Conv + 2 FC) as a lightweight model. Both Sync-DFL and CFL are able to reach the same accuracy at a similar point of time, as shown in Fig. 4a. However, we observe that CFL with MobileNet model takes a longer time to converge because the Mobilenet model is 3× larger than the 4 layers of CNN. For the 15-worker case of CIFAR-10 and MobileNet in simulation shown in Fig. 4c, both Sync-DFL and Async-DFL are able to achieve nearly 0.7 test accuracy within about 11 minutes in the simulation, outpacing CFL taking roughly twice as long at 23 minutes to achieve the same accuracy performance (all cases reaching a maximum accuracy of about 0.75).

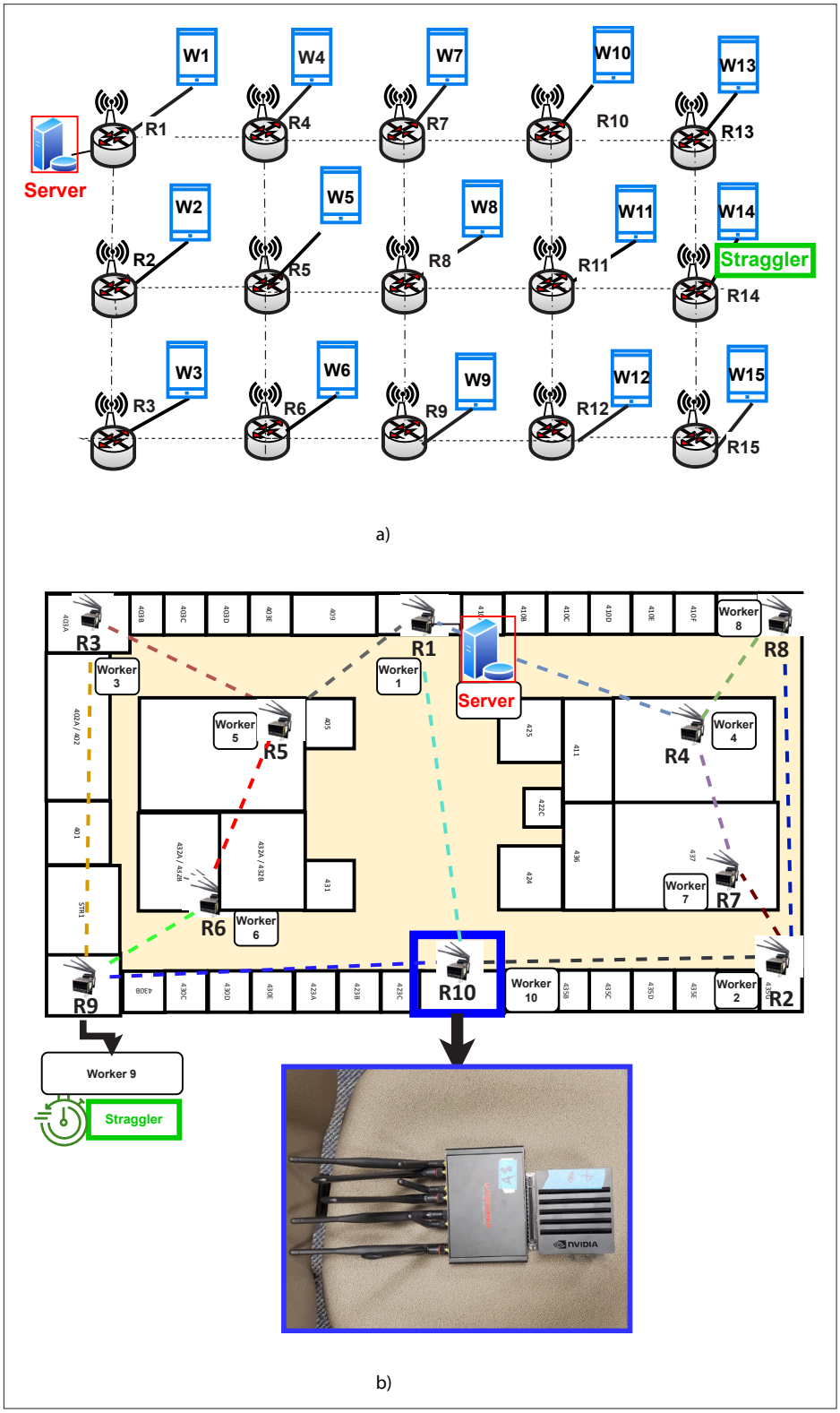

FIGURE 3. IoT multihop network topologies: a) a) simulated network topology; b) testbed topology.

The prolonged convergence time of CFL is due to the fact that each worker needs to send their updated local model to the central server through long and random multihop communication links in the IoT network, while the server needs to wait for all models from workers to proceed to the next training round. This causes network congestion to the bottleneck links around the server at router 1.

Sync-DFL and Async-DFL, meanwhile, only require the workers to send their updated models to their single-hop neighbors, which can fully utilize the network bandwidth and more evenly distribute traffic load. This can effectively alleviate the FL traffic around bottleneck links. As a result, Sync-DFL and Async-DFL reduce the model training time effectively by half in the experiments without any noticeable loss in convergence accuracy.

Model Convergence with Stragglers: When a straggler is introduced into the network, all FL solutions are affected by the increased model convergence time, where Async-DFL is affected least, as shown in Figs. 4d–4f. Sync-DFL's results resemble more closely that of CFL. This is because both Sync-DFL and CFL rely on synchronized local model training of all workers in the network, and thus the convergence time of Sync-DFL and CFL is directly affected by stragglers. However, compared to CFL, Sync-DFL can still achieve higher convergence speed because it confines the model exchange among single-hop neighbors, while CFL still suffers the communication bottleneck issue around the central server. This effect can be expected to continue as the network grows larger. Meanwhile, DFL's network traffic is only limited to the workers' number of single-hop neighbors as the network scales up.

In the simulated test (Fig. 4f),

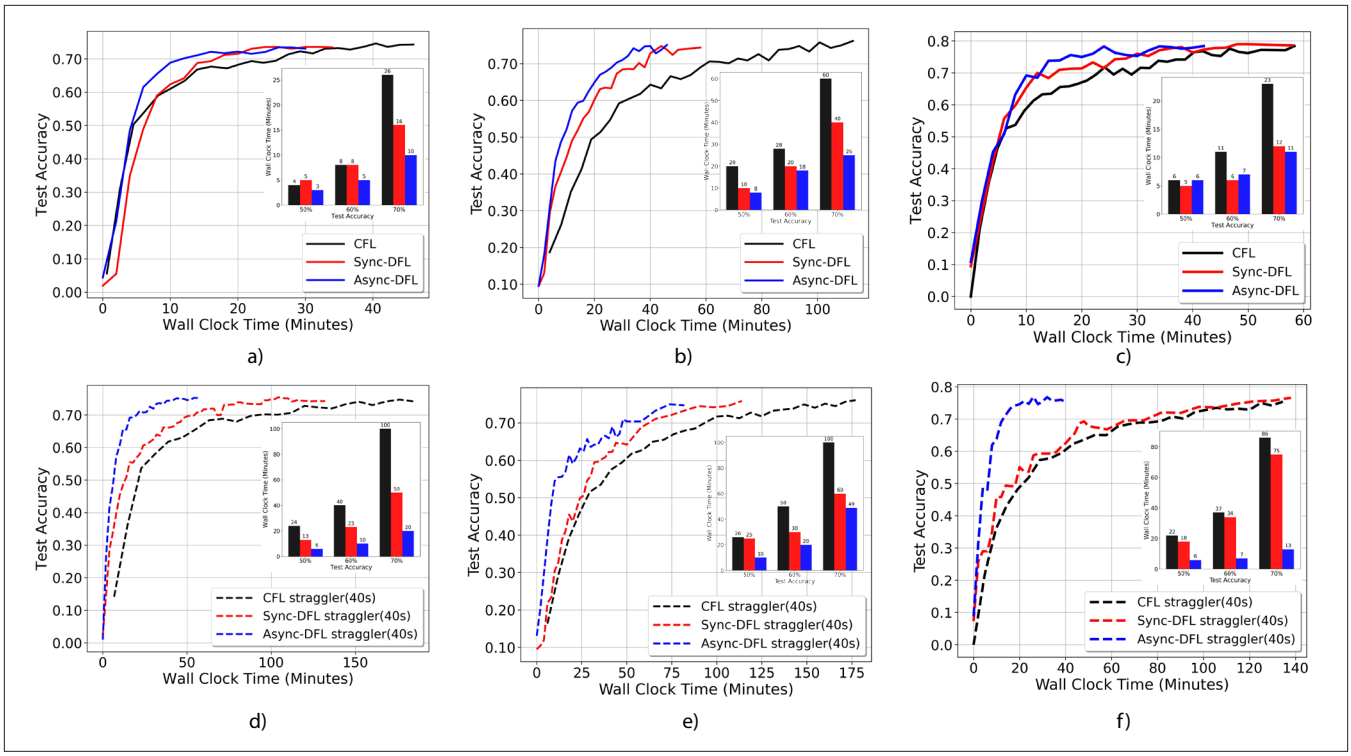

FIGURE 4. Comparison of centralized and decentralized federated learning performance in simulation and testbed environments (LEAF and CIFAR-10), each with cases having no straggler or having a straggler with a 40 s per local training round: a) testbed: LEAF (CNN 2 Conv + 2 FC) (no straggler); b) testbed: CIFAR-10 (MobileNet) (no straggler); c) simulation: CIFAR-10 (MobileNet) (no straggler); d) testbed: LEAF (CNN 2 Conv + 2 FC) (with straggler; e) testbed: CIFAR-10 (MobileNet) (with straggler); f) simulation: CIFAR-10 (MobileNet) (with straggler).

Methods	Testbed				Simulation	
	FMNIST (4 layers CNNs) Acc (75%)		CIFAR-10 (MobileNet) Acc (75%)		CIFAR-10 (MobileNet) Acc (75 %)	
	Non-straggler	Straggler	Non-straggler	Straggler	Non-straggler	Straggler
CFL	33 (1)	175 (1)	110 (2)	175 (2)	23 (1)	140 (1)
Sync-DFL	23 (1)	110 (1)	45 (2)	110 (2)	35 (1)	140 (1)
Async-DFL	15 (1)	50 (1)	38 (2)	75 (2)	25 (1)	20 (1)

TABLE 1. The total convergence time to achieve maximum (75%) of test accuracy and the corresponding computation time in parentheses (in minutes).

Async-DFL reaches 0.70 accuracy by about 13 minutes. Both CFL and Sync-DFL reach the same accuracy by around 86 and 75 minutes, respectively, roughly a 6× increase in time to converge compared to Async-DFL; the dramatic difference in tolerance can be seen visually in Fig. 4f. In the testbed, LEAF is used to demonstrate a relatively light job load in a live environment, whereas CIFAR-10 presents a more rigorous training job. In LEAF with a straggler, Async-DFL achieves more than 2× and 3× convergence speedup, compared with Sync-DFL and CFL, respectively, when all methods achieve the same 70 percent and 75 percent accuracies. In CIFAR-10 with testbed, compared with CFL and Sync-DFL, Async-DFL takes 1.5× and 3× less convergence time respectively to achieve the same maximum testing accuracy (75 percent). For a larger simulated network, Async-DFL achieves 7×speedup compared to both CFL and Sync-DFL.

Conclusion and Future Directions

To cope with the growing need for AIoT systems, the article examines and evaluates the efficacy of two different versions of decentralized federated learning models by building a composable generic decentralized FL framework. The experiment results demonstrate the superior convergence performance of DFL in multihop IoT networks compared to classic CFL. Moreover, the preliminary results show that Async-DFL can accelerate the model convergence speed while being very resilient and robust in heterogeneous IoT environments with the inevitable presence of stragglers. Async-DFL shows great potential to achieve the optimal trade-off between model convergence speed and model quality for further large-scale AIoT networks, which is worthy of further investigation.

Moreover, we will investigate semi-asynchronous DFL algorithms, where each worker only synchronizes with a subset of its one-hop neighbors, which can also dynamically change according to model quality and channel condition. Moreover, the current DFL algorithms generally assume all the workers are willing to cooperate. This assumption may not hold in practical settings; therefore, incentive schemes need to be developed for enhancing the cooperative gain. In addition, this article focuses on the algorithmic foundation of DFL. It will be worth theoretically investigating the convergence bounds of Sync-DFL and Async-DFL.

Acknowledgments

This work is funded by Intel/NSF joint grant 2003198 and NSF 2008447.

References

[1] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A Survey," *Computer Networks*, vol. 54, no. 15, 2010, pp. 2787–2805; https://www.sciencedirect.com/science/article/pii/S1389128610001568.
[2] J. Zhang and D. Tao, "Empowering Things With Intelligence: A Survey of the Progress, Challenges, and Opportunities in Artificial Intelligence of Things," *CoRR*, vol. abs/2011.08612, 2020; https://arxiv.org/abs/2011.08612.
[3] J. Konečný *et al.*, "Federated Learning: Strategies for Improving Communication Efficiency," *CoRR*, vol. abs/1610.05492, 2016; http://arxiv.org/abs/1610.05492.
[4] X. Lian *et al.*, "Can Decentralized Algorithms Outperform Centralized Algorithms? A Case Study for Decentralized Parallel Stochastic Gradient Descent," *Proc. 31st Int'l. Conf. Neural Info. Processing Systems*, ser. NIPS'17. Curran Associates Inc., 2017, p. 5336–46.
[5] J. Cao *et al.*, "Hadfl: Heterogeneityaware Decentralized Federated Learning Framework," *Proc. 2021 58th ACM/IEEE Design Automation Conf.*, IEEE, 2021.
[6] M. Chen *et al.*, "Wireless Communications for Collaborative Federated Learning in the Internet of Things," *CoRR*, vol. abs/2006.02499, 2020; https://arxiv.org/abs/2006.02499.
[7] M. Chen *et al.*, "Distributed Learning in Wireless Networks: Recent Progress and Future Challenges," *IEEE JSAC*, 2021.
[8] W. Y. B. Lim *et al.*, "Federated Learning in Mobile Edge Networks: A Comprehensive Survey," *IEEE Commun. Surveys & Tutorials*, vol. 22, no. 3, 2020, pp. 2031–63.
[9] X. Lian *et al.*, "Asynchronous Decentralized Parallel Stochastic Gradient Descent," *Proc. Int'l. Conf. Machine Learning*, PMLR, 2018, pp. 3043–52.
[10] C. He *et al.*, "Fedml: A Research Library and Benchmark for Federated Machine Learning," *CoRR*, vol. abs/2007.13518, 2020; http://arxiv.org/abs/2007.13518.
[11] A. G. Howard *et al.*, "Mobilenets: Efficient Convolutional Neural Networks for Mobile Vision Applications," 2017; http://arxiv.org/ abs/1704.04861.
[12] P. Pinyoanuntapong *et al.*, "Edgeml: Toward Network-Accelerated Federated Learning over Wireless Edge," *CoRR*, vol. abs/2111.09410, 2021; http://arxiv.org/abs/2111.09410.
[13] P. Pinyoanuntapong *et al.*, "Fedair: Towards Multi-Hop Federated Learning Over-the-Air," *Proc. IEEE SPAWC*, 2020.
[14] P. Pinyoanuntapong *et al.*, "Sim-to-Real Transfer in Multi-Agent Reinforcement Networking for Federated Edge Computing," *CoRR*, vol. abs/2110.08952, 2021; http://arxiv.org/abs/2110.08952.
[15] R. R. Fontes *et al.*, "Mininet-Wifi: Emulating Software-Defined Wireless Networks," *Proc. 2015 11th Int'l. Conf. Network and Service Management*, 2015, pp. 384–89.

Biographies

Pinyarash Pinyoanuntapong is a Ph.D. student in the Department of Computer Science, University of North Carolina at Charlotte. He received his B.Eng. degree in computer engineering in 2016 and Master of Science in computer networks in 2017 from Wichita State University. His research interests include reinforcement learning, federated learning, and wireless networks.

Houston Huff is a Ph.D. student in the College of Computing and Informatics at the University of North Carolina at Charlotte. He received his B.S. degree in computer science with a minor in physics in 2020, and is currently in his first year of Ph.D. study. His research interests include augmented/virtual reality, machine learning, network communication, and artificial intelligence.

Minwoo Lee is an assistant professor in the Department of Computer Science, University of North Carolina at Charlotte. He received a Ph.D. degree in computer science from Colorado State University in 2017. His current research interests include foundational machine learning problems including adaptive systems, transfer learning, knowledge representation, robust knowledge augmentation, interactive learning, and evidence-driven explanation and reasoning.

Chen Chen is an assistant professor at the Center for Research in Computer Vision, University of Central Florida. He received his Ph.D. degree from the Department of Electrical Engineering, University of Texas at Dallas in 2016 where he received the David Daniel Fellowship (Best Doctoral Dissertation Award). His research interests include computer vision, efficient deep learning, and federated learning. He is an Associate Editor of *IEEE Transactions on Circuits and Systems for Video Technology*, the *Journal on Real-Time Image Processing*, and the *IEEE Journal on Miniaturization for Air and Space Systems*.

Pu Wang received his B.Eng. degree in electrical engineering from Beijing Institute of Technology, China, in 2003, and his M.Eng. degree in electrical and computer engineering from Memorial University of Newfoundland , Canada, in 2008. He received his Ph.D. degree in electrical and computer engineering from the Georgia Institute of Technology, Atlanta in August 2013. Currently, he is an associate professor with the Department of Computer Science at the University of North Carolina at Charlotte. His current research interests focus on AI for networked systems, including reinforcement learning for networking optimization, distributed/federated learning over wireless edge computing, deep learning for wireless/radar sensing, and swarming intelligence for multi-robot systems.