2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA) | 978-1-7281-8470-8/20/$31.00 ©2020 IEEE | DOI: 10.1109/ICMLA51294.2020.00185

FedAR: Activity and Resource-Aware Federated Learning Model for Distributed Mobile Robots

Ahmed Imteaj and M. Hadi Amini
School of Computing and Information Sciences, College of Engineering and Computing, Florida International University
Sustainability, Optimization, and Learning for InterDependent networks laboratory (solid lab)
Miami, FL, USA
{aimte001, moamini}@fiu.edu

***Abstract*—Smartphones, autonomous vehicles, and the Internet-of-things (IoT) devices are considered the primary data source for a distributed network. Due to a revolutionary breakthrough in internet availability and continuous improvement of the IoT devices capabilities, it is desirable to store data locally and perform computation at the edge, as opposed to share all local information with a centralized computation agent. A recently proposed Machine Learning (ML) algorithm called *Federated Learning (FL)* paves the path towards preserving data privacy, performing distributed learning, and reducing communication overhead in large-scale machine learning (ML) problems. This paper proposes an FL model by monitoring client activities and leveraging available local computing resources, particularly for resource-constrained IoT devices (e.g., mobile robots), to accelerate the learning process. We assign a trust score to each FL client, which is updated based on the client's activities. We consider a distributed mobile robot as an FL client with resource limitations either in memory, bandwidth, processor, or battery life. We consider such mobile robots as FL clients to understand their resource-constrained behavior in a real-world setting. We consider an FL client to be untrustworthy if the client infuses incorrect models or repeatedly gives slow responses during the FL process. After disregarding the ineffective and unreliable client, we perform local training on the selected FL clients. To further reduce the straggler issue, we enable an asynchronous FL mechanism by performing aggregation on the FL server without waiting for a long period to receive a particular client's response.**

***Index Terms*—Federated Learning, resource-limitations, trust, mobile robot, straggler, Internet-of-Things, distributed computing, model aggregation.**

I. Introduction

A. Motivation

The ubiquitous nature of IoT devices causes huge data streams due to their widespread applications. Storing and processing such vast amounts of data in a centralized location is costly, highly insecure, and time-consuming. Further, in some cases, the nature of local data is sensitive and requires further security measures to ensure user confidentiality and data privacy while exchanging data for computation and decision-making purposes. Sensitive data such as captured images, browsing history, personal information, and location-based services can be utilized for recommendations, social advertising, prediction, or incurring any potential privacy risks. Such sensitive information is unsafe to share with the server if potential privacy issues are not checked [1]. Besides, as privacy preservation awareness is rising, legal laws and restrictions, i.e., General Data Protection Regulation (GDPR), are becoming prominent, which reduce the feasibility of data aggregation [2]. The conventional centralized ML techniques can not be implemented in a distributed fashion due to infrastructure fallibility, including intermittent or weak network connectivity, limited bandwidth, or response delay [3]. To tackle the above-mentioned issues, an alternative distributed ML paradigm named *Federated Learning (FL)* is proposed in [4] that performs on-device training based on client's local data, pushes client's training parameters at the edge, and learns from the global model. The popularity of FL applications is increasing due to their high-acceptability, particularly in improving user privacy. The FL process enables network clients to generate a joint ML model that enhances privacy by not exposing any clients' private data. Another important factor of FL is that it works with nonindependent and identically distributed (non-IID) data samples observed in real-world settings [5], [6]. That means the FL algorithm is capable of handling the changes over time in the distribution of collected data samples. However, during an FL process, the client selection is crucial as a straggler, and an unreliable client can retard the learning process and prolong the model convergence time. Any client that is selected for a training phase is called a participant. A participant may turn into a straggler if the participant is underpowered in system requirements for a particular model training. Most of the existing FL approach avoids the straggler issue by assuming all the FL clients as proficient nodes and randomly selecting clients for the training rounds. However, the presence of such straggler clients has a vital impact on the overall performance of learning model [7]. Further, there is a possibility of selecting a vulnerable FL client, specifically in an FL-based IoT (FL-IoT) environment, where the devices are more prone to susceptibility. Therefore, we need to monitor the clients' available resources, behaviors, and contributions towards training a learning model.

In our proposed approach, we consider mobile robots integrated with comparatively low processing units, limited memory, bandwidth, and battery power instead of assuming proficient clients with sufficient resource availability. We give instructions to each registered mobile robot for collecting data samples. We track the robot clients through remote sensing mechanisms, and each client gets triggered upon receiving a

978-1-7281-8470-8/20/$31.00 ©2020 IEEE
DOI 10.1109/ICMLA51294.2020.00185

particular notification from the server [8]. Before executing a training round, we check the resource availability following the system requirement for model training and examine each interested client's trust score, which is updated based on their previous training performance. By enabling asynchronous FL, we further reduce the straggler effect by ignoring the unresponsive clients.

B. Literature Reviews

The FL research domain is increasingly evolving due to its capability to handle non-IID data, preserve privacy, and reduce communication overhead. Prior works from various disciplines, including distributed systems, ML, databases, cryptography, and data mining, focused on improving the FL method. An overview of the FL system design is presented in [9], where they discussed the complexity of FL design by posing challenges of FL implementation and experimental evaluation. In consequence, some open-source frameworks are proposed to handle FL mechanisms such as TensorFlow Federated (TFF) designed by Google [10], Federated AI Technology Enabler (FATE) from WeBank [11], LEAF [12], and PySyft [13].

FL strategy fits best in such applications, where we need to perform model training using sensitive data. Hence, on-device training is preferable rather than passing data to the cloud. The authors in [14] proposed an FL-based model training for predicting heart attack in a patient. By configuring the necessary setup and integrating wearable devices, they observed the patient's health data and performed FL-based on-device model training. Besides, a Federated Transfer Learning (FTL) framework is presented in [1]. In that paper, they considered each hospital as an FL client and performed collaboration among neighbor hospitals to improve the FL model. In [15], FL-based privacy-preserving named entity recognition (NER) is implemented that can identify different medicinal attributes (e.g., drug names, its reactions, and symptoms) by analyzing medical texts and performing classification.

Device-centric application data is utilized to build a recommendation system that aims to predict a rating for an item or user preference. For such a recommendation system, usually, one user data is shared with other users, and privacy is violated in many cases. A federated meta-learning-based recommendation system is proposed in [16] considering the privacy issues. Each FL client shares their generated algorithm with the server to carry out an effective global model. Similarly, privacy-preserving news recommendation system [17] and personalized recommendation system [18] are constructed based on the FL concept. Another interesting application of FL is user-typing-behavior-based next-word prediction designed by [19]. Their application can read a user's mind while typing by analyzing the user's messages and browsing history. A similar type of application to predict emoji on a mobile keyboard is presented in [20]. Moreover, wake-word detector-based applications are prevalent nowadays that consider user voice or speech for training an FL model that can recognize the user's command. The author in [21] designed such an FL-based wake-word detector application by using a crowd-sourced dataset that does not expose a user's voice during training. The ranking search result is another recent application of FL [9], [22], where the user query and preference are not shared with any other entity, and the search result is generated based on FL training. A content suggestion framework is designed on the concept of user activities, i.e., clicking or ignoring a content, which is discussed in [23]. This application tracks and stores the user-click information on content suggestions, and constructs a model accordingly. Both training and inference are performed on-device in their work, and only the model parameters are dispatched to the server. However, most of the existing FL-based applications assumed that the FL clients are resource-boundless and trustworthy, i.e., any FL client can be selected for the training phase. However, if we consider a real-life FL-IoT setting, the FL client is often resource-bounded and prone to vulnerability. This paper aims to consider both resource-constrained and reliability issues in an FL environment. According to the best of our knowledge, there is no existing work that examines the resource status and scrutinizing the client activity within an FL environment.

C. Contributions

The main contributions of this paper can be listed as follows:

- We propose a novel model capable of giving a reward or a punishment to each participating FL client based on their performance during a training period.
- We propose a strategy to choose only capable clients before starting the FL training phase.
- We consider mobile robots to collect federated data in a distributed fashion and perform FL training simulation for a resource-constrained IoT environment.
- We consider an asynchronous FL scheme to mitigate the straggler effect on a resource-constrained FL-IoT environment.

D. Organization

The rest of this paper is organized as follows. Section 2 highlights the overview of FL. Section 3 elaborately explains our novel FL framework to minimize prediction loss and obtain high accuracy. Section 4 presents our experiment results, followed by section 5, that concludes the paper.

II. Background of Federated Learning

Federated Learning (FL) is a distributed ML algorithm that enables on-device learning of clients within a network instead of passing raw data to the server. Based on the clients' models, or algorithms, the FL algorithm learns a single, global model. The advantage of FL method is that both federated data and models of each client remain local. In a distributed system, federated data refers to a collection of extracted data elements hosted over a cluster of devices. For instance, any applications running on a mobile device can store those application data locally, or, using distributed sensing, mobile nodes can sense sensor data and store those without sharing with a centralized location. To initialize the learning process, federated data is required, and we need to prepare a dataset which holds data

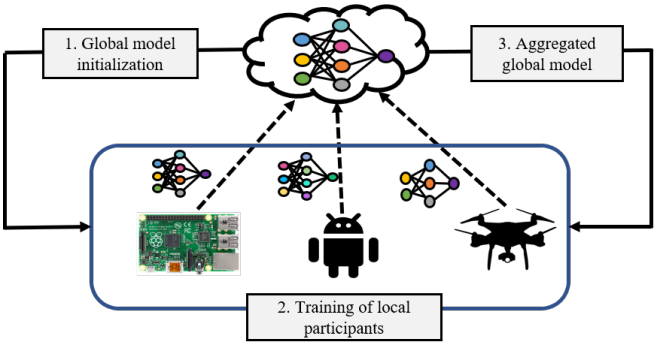

Fig. 1: FL procedure considering N participants.

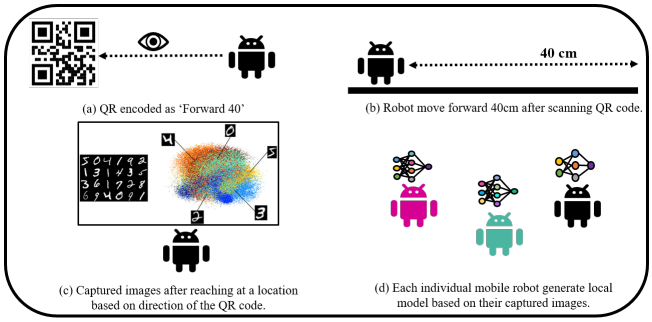

Fig. 2: Real-time data collection procedure in an FL environment.

from multiple sources or agents. In a typical FL scenario, we may need to deal with a very large number of user nodes, but only a small portion of them may be available at a certain time to perform training sessions. For instance, if the client nodes are mobile robots, then the devices can only be considered for a training round when they are charged, and active.

The author in [24] introduced the FL term and proposed a popular FL algorithm named Federated Averaging (FedAvg). The model training of FL is done in several communication rounds until desirable model accuracy is obtained. To decrease network traffic, the training period on each client side is carried out through different batches. Further, only local model parameters or statistical summary are shared with the server, i.e., client data does not need to leave its own device.

The FL method illustrated in Figure **1** comprises three steps: ***Step 1 (FL Task and Global Model Initialization)***: In the initial step, the FL server initializes a task with specific requirements, and an initial global model. The global model is defined by applying initial hyper-parameters and a learning rate. In each communication round, the server selects a subset of clients, known as participants. After that, the server transfers the global model to all the selected participants.
Step 2 (On-device Training): Each selected client performs an on-device training using their local data and updates own local model parameters by learning from the global model. The main goal of each local client is to find optimal parameters to minimize the loss function. Each client discovers the local optimal parameters by applying stochastic gradient descent (SGD) on their local available data.
Step 3 (Perform Aggregation): The server waits to receive a local model update from each training participant. Upon receiving the model parameters, the server performs an aggregation and updates the global model. The latest global model broadcasts again to all newly selected participants.

The local model and global model update, i.e., step 2 and step 3, are repeated until the loss function is minimized, or the model reaches a convergence.

III. System Description

In this paper, we assume that we have a resource-constrained FL environment, where a client may have a straggler effect, and any interested client may provide inappropriate model information during the training process. The main reason of straggler effect is the resource shortage issue that leads us to think about a strategy to avoid an unreliable or inefficient client during the learning process. Moreover, the existence of unreliable or inconsistent clients can prolong the convergence time and may create a negative impact on overall model accuracy. To understand the resource-constrained FL behavior, we focus on collecting real-time federated data collection, handling resource heterogeneity, and monitoring client activity during an FL process.

A. Real-time Data Collection through Distributed Sensing

In a federated dataset, we need to have a column of client id that represents which client possesses the corresponding row information. To collect federated data using distributed mobile robots (clients), we need to send a set of instructions to those clients. One naive strategy is to collect data by controlling the robots remotely. However, such a strategy is not convenient if we have many clients, and we need to keep track on each client's information. We design an approach for controlling mobile robots through which any robot can be tracked, and acknowledged about required instructions in a convenient way. In this approach, any new client interested in joining an FL network needs to commit registration by providing their credentials. We generate a token and store the generated token in our server. A token is a unique identifier that helps to recognize each robot client in a distinguishable way. The token can be used as the client id in our federated dataset. More details can be found in our prior work [8]. We infuse functionalities to each robot so that they can identify a push notification when it is sent from the server. The push notification is used to trigger a robot and pass the commands as a Quick Response (QR) code. We use QR code to hide raw instructions. We enable a QR code scanning mechanism to each mobile robot to read the instructions given as a form of a QR code and understand the meaning of the QR code. After successfully scanning the QR code, a robot client can get the instructions as a form of an array. The first element of the resulting array is the key, and the following elements are the parameters. For instance, an instruction of $Forward$ 40 10 0 means move forward with speed $40ms^{-1}$ for 10 seconds with an angle of 0 $degree$. In case we have multiple instructions within a QR code, we split those by a colon separator. By reaching at a desired place, the robot can capture images or

can sense the environment. After that, the collected data are properly labeled. Each distributed robot can train themselves by the labeled data, and generate a local model that is shared with the server. In Figure **2**, we presented an IoT based data collection overview for an FL environment.

B. Construction of a Trust and Resource-aware Framework

1) Publish FL Task: An FL process can be considered a monopoly market, where a task publisher acts as a monopolist operator and a set of mobile robots $\mathcal{N} = \{1, \ldots, N\}$ act as the clients. Each client $n \in \mathcal{N}$ uses its local training dataset of size s_n for being a part of the FL task. Each of the training data has an input-output pair, where the input is a vector that contains data sample features, and the output indicates a label for each input vector. The task publisher broadcasts an FL task with minimum resource requirements (e.g., memory, battery, and processing power) and a minimum trust score to qualify for the task participation.

2) Check Resource Availability of Interested Clients: Upon receiving information about interested clients' resource availability, the server filters out the candidates that do not meet the task requirement. Only the robot that satisfies the requirements has a chance to join the training phase. In Algorithm **1**, we have a function $CheckResource$ that takes an input of memory ($\mathcal{M}$), bandwidth ($\mathcal{B}$), and battery life ($\mathcal{E}$) in line **14**. We generate resource availability for the three give inputs (line **15**) and compare with the task requirement $\mathcal{L}_{Req}$ (line **16**). If available resources satisfy the task requirement, then we add that client to a list RA (line **17-18**).

3) Calculate Trust of FL-client: In our proposed FedAR model, we choose a fraction of clients that may change at each training round according to resource availability (e.g., memory availability) and the client's updated trust value. To construct a trust model, we pass the iteration number/communication round (i), client id (m), global model parameters (w_i), threshold time t for sending back local model, and model deviation γ. Each FL participant needs to send back their local model parameter within a given period of t that is set by the task publisher. However, we may encounter various situations (e.g., fast or slow response) during the training process and may receive different FL participants' responses (e.g., inappropriate model information). If a client joins the FL network for the first time, we set its trust score as $C_m = 50$. We add a trust score $C_{Interested}$ with a value of 1 to each of the clients interested in being a part of a training phase, successfully meets the trust and resource requirement for that task, but could not join the training round. We provide $C_{Interested}$ to encourage the trustworthy and capable candidate to participate in a future task. After selecting the FL client, we reward each client that accomplishes a task, and if it fails, we decrease its trust value as a punishment. Every participant client in a training round must submit their model within a given time because it is not feasible for the server to wait for a client for an infinite amount of time. The task publisher can set the threshold time for a task. In case an FL participant gives responses within the desired period, we provide a reward to that client (C_{Reward} with a trust value of 8). On the other hand, when an FL client fails to accomplish its task on time, we set a penalty on that client's trust score ($C_{Penalty}$ with a trust value of -2). We check the past performance of that client, and if the client repeatedly failed to respond on time ($20\% - 50\%$ of its participation), then we add a C_{Blame} trust value to that client's trust score (where C_{Blame} = -8). In case the client's straggling effect is observed above 50% of its overall participation, or if the client sends a model that has high deviation compare to the other client's models, we add a C_{Ban} trust value to that client's trust score (where C_{Blame} = -16). In Table **I**, we presented the assigned trust score of different events.

TABLE I: Trust values considering different factors in an FL environment.

Factor	Value
$C_{initial}$	50
C_{Reward}	8
$C_{Interested}$	1
$C_{Penalty}$	-2
C_{Blame}	-8
C_{Ban}	-16

The details of our trust and resource-aware model are presented in Algorithm **1**. In line **1**, we receive communication round as i, client id as m, global model parameter as w_i, maximum time t to finish that task, and threshold of model diversity γ. The threshold time to perform a task can be changed in different iterations by the task publisher based on the client's performance. We also do not consider a fixed threshold because, in the initial period of training, a model deviation can be larger for all clients than the global model, while after some iteration, it is supposed to get reduced to some extent. After sending all the function parameters, line **2-4** checks whether a client sends back its optimized local model within a preset time t. If the client sends its optimized local model within time, then we set that client's unsuccessful record (U_m^i) for that training round as 0 and give that client a reward score (C_{Reward}), which is added to the client's trust score. In case a client can not send back its local model within time t, we set that client's unsuccessful record (U_m^i) for that training round as 1 (line **6**). We check how frequently that client encounters unsuccessful or ineffective local model generation. If that unsuccessful record indicates below 20% successful task completion of the overall client participation history, then we add a penalty score ($C_{Penalty}$) to that client as a sign of warning (line **7-8**). In case we get more than 20%, but less than 50% unsuccessful record history for a participated client, then we set a blame score (C_{Blame}) to the existing trust score of that client (line **9-10**). Moreover, if a client is responsible for giving a slow response for more than 50% of its overall task participation history or sending back a model with large diversity compared to other client's local models, we assign a Ban score (C_{Ban}) to the existing trust score of that client (line **11-12**). Finally, in line **13**, we append the updated trust score of each client to a list.

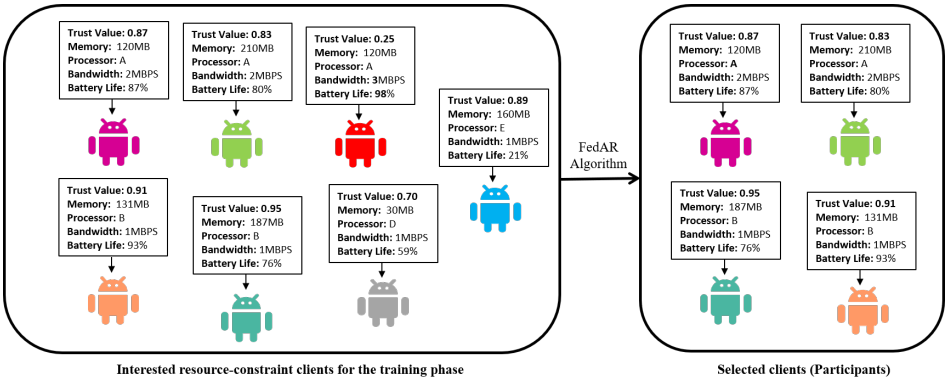

Fig. 3: Selection of proficient and trustworthy clients for training phase.

Algorithm 1: Activity and Resource-Aware Model. The global model of i^{th} training round is represented by G^i, D^i_m indicates the local model of a mobile robot m on i^{th} iteration, unsuccessful record of a client m on iteration i is denoted by $(U^i_m.)$, trust score C_m for m^{th} client, t represents timeout, and γ indicates deviation.

UpdateTrustScore (i, m, w_i, t, γ)**:**
if m *sends model* w^i *within* t **then**
  set $U^i_{ID(m)} = 0$
  set $C_m = C_m + C_{Reward}$
else
  set $U^i_{ID(m)} = 1$
  if $\frac{1}{i}\sum_{p=1}^{i} U_m^p < 0.2$ **then**
    set $T^i_{ID(m)} = C_{Penalty}$
  if $\frac{1}{i}\sum_{p=1}^{i} U_m^p < 0.5$ *and* $\frac{1}{i}\sum_{p=1}^{i} U_m^p \geq 0.2$ **then**
    set $C_m = C_m + C_{Blame}$
  else if $\frac{1}{i}\sum_{p=1}^{i} U_m^p \geq 0.5$ *or* G^i-$D^i_m > \gamma$ **then**
    set $C_m = C_m + C_{Ban}$
Append C_m to $TrustList$
CheckResource $(\mathcal{M}_m, \mathcal{B}_m, \mathcal{E}_m)$**:**
Resource availability, $\mathcal{R}_m = f(\mathcal{M}_m, \mathcal{B}_m, \mathcal{E}_m)$
Compare $\mathcal{R}_m$ with $\mathcal{L}_{Req}$
if $\mathcal{R}_m$ *satisfied* $\mathcal{L}_{Req}$ **then**
  Add $\mathcal{R}_m$ to RA list
Return C and RA

4) Select Client for Training Phase: In a typical FL scenario, we may need to deal with many user nodes; however, only a small portion of them may be available at a specific time to perform training sessions. For instance, if the client nodes are mobile robots, they can be considered for training round only when charged and active. In our FL-IoT environment, all client data are locally available, and we need to choose a subset of clients to participate in the training process. This subset of clients is changed in each round, considering the trust score and resource availability. After calculating the trust value and checking out resource availability, we select the eligible candidates as participants. Each participant generates their local optimal decisions after giving consent to participate in a task according to its resource conditions and local dataset content. The effectiveness of the local model update directly depends on the local dataset accuracy [25]. After selecting the clients, we can reuse them again and again until our target model reaches convergence. In Figure **3**, we presented the client selection process of our proposed FedAR method.

5) Creating Model with Forward Pass Method: Initially, we prepared input samples of client images to be grouped as batches. We identify the TensorFlow variables that are required in constructing our model. To represent the entire dataset, we consider a data structure that holds variables such as model weights, bias, and various counters and cumulative statistics updated during training. We define a forward pass method to compute loss with these variables and model parameters, make predictions, and update the statistics of sample batches of input. After that, we define a function that is responsible for returning a set of local metrics. Each client devices send their local metrics to the central server. The server performs aggregation upon receiving the values in a federated learning evaluation process. To the end, we build a model representation to use with TensorFlow-Federated (TFF) and apply Keras optimizer on the model. Finally, the constructed model and optimizer instances are fed to the FedAR algorithm, and the process is initialized with federated train data to generate output metrics loss and accuracy. The overall working process is depicted in Figure **4**.

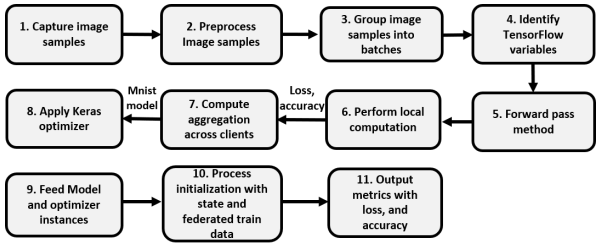

Fig. 4: FL-based image classification approach considering TensorFlow framework.

6) Evaluate Local Model Quality: After client selection, each client can be trained with FL optimization technique

(e.g., FL-SGD). The task publisher disseminates the initial shared global model parameters to all selected clients. Each mobile robot trains its model using local data and uploads its model parameters in the server. To ensure the reliability of the model update, we leverage model quality evaluation by applying FoolsGold scheme [26] that identifies unreliable participants by observing their local model updates' gradient diversity and helps to avoid poisoning attack. If a local client's model update performance lower than a specified threshold or a client repeatedly sent similar gradient updates, then the task publisher rejects the client update and does not update the global model. With the resource checking, unreliable client handle strategy, and malicious attacker detection approach, weak clients, and unreliable model updates are not considered during the learning process. The task publisher only considers the reliable local model update and performs a federated averaging strategy [24] to generate an updated global model.

7) Leveraging Asynchronous FL to Accelerate Convergence: This paper assumes that we have resource-constrained clients within the learning environment; therefore, performing synchronous federated learning can lead the server to wait for a long time to update the global model. In a synchronous FL, the server performs aggregation only after receiving a response from all the available clients. However, if a client has a straggler effect, then the client's slow response time can harm the overall model convergence. For such a scenario, synchronous FL does not guarantee convergence [7], [9]. On the other hand, in asynchronous FL, the server performs aggregation every time it receives a model from a client (See Figure **5**), i.e., the task publisher does not have to wait for a particular client. Hence, the straggler effect does not hamper the overall model effectiveness, and it guarantees convergence [27]–[29]. We applied the asynchronous FL strategy in our proposed FedAR algorithm due to the uncertain response time of the heterogeneous resource clients. Each time the server updates its global model, the server sends the model back to all the available participants for the next iteration until the updated global model satisfies a convergence condition set by the task publisher.

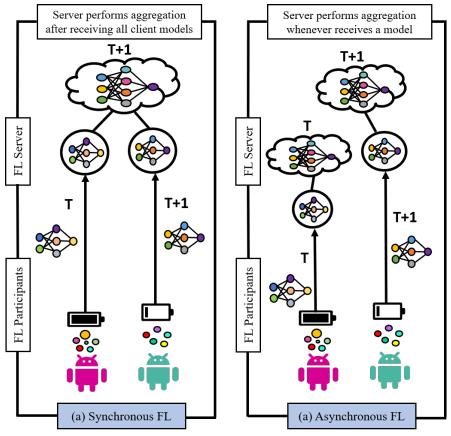

Fig. 5: Synchronous and asynchronous FL approach.

8) Update Trust Models based on Performance: After successfully executing an FL task, the server assigns trust value to each participant based on their performances. If we update the trust value after reaching model convergence, then there remains a possibility of repeatedly selecting stragglers and unreliable clients for the training process. Rather than, we update each FL participant's trust score after completing every iteration, which eventually helps to avoid straggler or inconsistent client in the further iteration.

Algorithm 2: FedAR: Activity and Resource-aware Federated learning. The $\mathcal{S}$ eligible clients are indexed by u; $\mathcal{B}$ is the local minibatch size, $\mathcal{F}$ is the client fraction, E is the number of local epochs, η is the learning rate, and t represents timeout.

1 **Registration:** Each client commits registration
2 Initialize Trust score to newly registered clients
3 Disseminate system requirements to all clients
4 **Server executes:** initialize w_0
5 Reveal resource availability by each interested client.
6 **for** *each round* $m = 1, 2, \ldots$ **do**
7 RA_m = **CheckResource** $(\mathcal{M}_m, \mathcal{B}_m, \mathcal{E}_m)$
8 Sort available client based on $TrustList$ and RA, and store in $\mathcal{S}$
9 $\mathcal{C} \leftarrow$ Top $\mathcal{S} \cdot \mathcal{F}$ clients
10 $M_m \leftarrow$ (random set of $\mathcal{C}$ clients)
11 **for** *each client* $u \in M_m$ *in parallel* **do**
12 $w^u_{m+1} \leftarrow$ ClientUpdate (u, w_m)
13 **if** *Model received from* u *within time* t **then**
14 $w_{m+1} \leftarrow w_{m+1} + \frac{n_m}{n} w^u_{m+1}$
15 **UpdateTrustScore** (m, u, w_m, t, γ)
16 **ClientUpdate**(k, w) : // Run on client k
17 $\mathcal{B} \leftarrow$ (split $\mathcal{P}_k$ into batches of size $\mathcal{B}$)
18 **for** *each local epoch* i *from 1 to* E **do**
19 **for** *batch* $b \in \mathcal{B}$ **do**
20 $w \leftarrow w - \eta \nabla \ell(w; b)$
21 return w to server

C. Proposed FedAR Algorithm

We presented our proposed *FedAR* scheme in Algorithm **2**. Initially, each client needs to register themselves to join in a network (line **1**). We initialize each newly joined FL client with a trust value (line **2**). The server disseminates a task with a system requirement to each available client and initializes the task parameters (line **3-4**). In (line **5**), the server receives available resource information from all clients.

In each communication round of the training phase, the available resource of each client is compared with the system requirement, and eligible clients are included in a list RA (line **6-7**). We sort the interested clients according to their trust score and resource availability, and store the eligible candidates into a list $\mathcal{S}$ (line **8**). We select a fraction of the eligible candidate in line **9**, and randomly select a subset of eligible clients (line

10). For each selected client, we pass the latest global model (line **11-12**). We assume, there are M clients, each client have n_u local data, and the overall data n is partitioned among the M clients with a set of indexes $\mathcal{P}_u$ on client u, where $n_u = |\mathcal{P}_u|$. Each client splits their local data into batches, performs SGD for each of the batches, and sends back the local model parameter to the server (line **16-21**). If the server receives the client's model parameter, it immediately updates the global model by performing aggregation (line **13-14**). Finally, in line **15**, the trust score is updated based on a client's response within a threshold time of t.

IV. Experimented Results

A. Simulation Settings

To create a resource-bounded real-time FL environment, we considered twelve distributed mobile robots that can follow a set of given instructions. We integrated variant sizes of memory, processor, and battery to each mobile robot to simulate the heterogeneous environment in terms of hardware configurations. We use a combination of a popular digit classification dataset called MNIST [30] and our captured digit images using distributed mobile robots. We consider eight reliable and consistent robots and four unreliable robots. Among the four unreliable robots, two of them have issues regarding resource scarcity, while the other two generate low-quality models that can be considered a poisoning attack. We provided ten classes of digits for the unreliable workers and deliberately modified some of the training samples to mislead our FL model training process. The strength of the poisoning attack depends on the modification percentage of the labels. The robot clients use a batch size of twenty and compute five local iterations to accomplish a local SGD update. We set the same transmission rates for all the existing robot clients during the local model update to attain simplicity during execution. Therefore, we maintain the same energy consumption and transmission rates for all robot clients.

B. Performance Evaluation

We evaluate an FL setting's performance by considering different chunks of data samples for the distributed mobile robots (see Table **II**). As we mentioned before, we consider four unreliable mobile robots with resource limitations and assign fewer image samples and classes to those clients (i.e., Robot 3, Robot 5, Robot 6, Robot 9 in Table **II**). We randomly apply either Softmax or Relu activation and set instructions to each robot to collect image samples from the environment. The data label and image sample number assigned to each of the robots are presented in Table **II**. Each image is simply a matrix of pixels where each pixel indicates color density. We flatten the 28x28 image samples into 784-element arrays. After that, we shuffle the samples and group them into batches. We shuffle our images to avoid the risk of creating batches that are not representative of the overall dataset. We group the images into batches to consider mini-batch of data samples while applying local SGD. We utilize an FL framework TensorFlow 1.12.0 to evaluate the digit classification task. We apply forward pass method to perform local computation, apply a $keras$ optimizer function to obtain an optimized result, and track loss using $SparseCategoricalCrossentropy$ loss function.

TABLE II: Model architectures of FL mobile robots.

FL Client	Emnist Labels	Activation Function	Number of Image Samples
Robot 1	0-9	Softmax	1000
Robot 2	0-9	ReLu	1000
Robot 3	0,1,2,3	Softmax	400
Robot 4	0-9	Softmax	1000
Robot 5	4,5,6	ReLu	300
Robot 6	7,8,9	ReLu	300
Robot 7	0-9	Softmax	1000
Robot 8	0-9	ReLu	1000
Robot 9	5,6,8	Softmax	300
Robot 10	0-9	Softmax	1000
Robot 11	0-9	ReLu	1000
Robot 12	0-9	Softmax	1000

We simulated our model's behavior with the variance of batch size and local epoch of each robot client (see Figure **6**). In Figure **6**, we use E to represent the local epoch of each client and B to indicate the data samples' batch size. We can see that the accuracy of FL increases (i.e., prediction loss decreases) as the communication round goes up. We observe that when we have a batch size of 10 and a local epoch of 20, we obtain better model accuracy for our data sample. The learning period continues until we reach a target convergence.

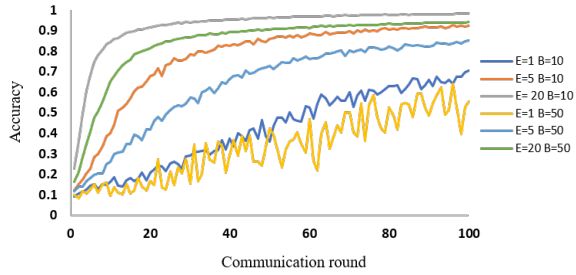

Fig. 6: FL accuracy of mobile robots with variance in batch size and local epoch.

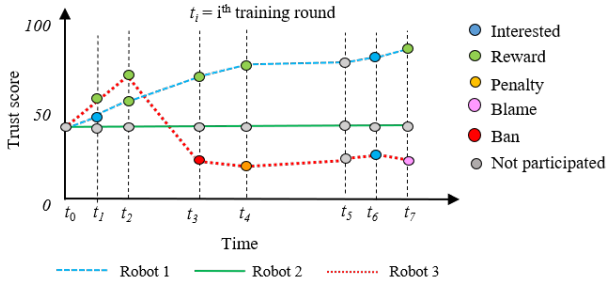

Fig. 7: Activity dependent trust scores of three mobile robots.

To observe the clients' trust score update, we considered different events, e.g., successful task completion, improper model parameter infusion, delay in sending a response, interest

to be a part of a training phase, and not able to participate due to unavailable resources. In Figure **7**, we visualized the trust score update of three different mobile robots in different periods. Besides, we simulated the straggler effect on the FL process by considering different straggler robots that can not send back their local model update within a given time and eventually reduce the global model's overall accuracy. Figure **8** depicted that less number of straggler robots accelerates the FL accuracy.

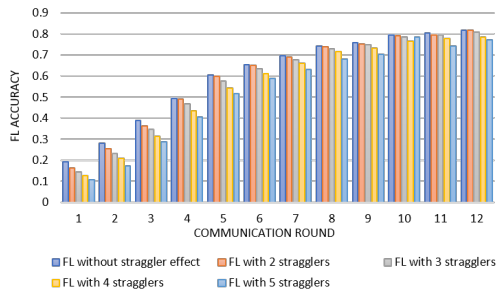

Fig. 8: FL performance in presence of straggler effect.

V. Conclusion

This paper proposes a trust and resource-aware FL framework to deal with the untrustworthy and resource-constrained FL environment. For our simulation settings, we consider distributed mobile robots that have reliability and resource limitation issues. Instead of assuming all the FL clients are consistent and resource-efficient, we consider additional steps to check each client's resources and previous training performance. We enable a feature of assigning a trust score to each robot client, and avoiding the inconsistent and inadequate resource clients from the training process. To further handle the straggler effect, we selected the most proficient and reliable clients for an FL task and applied asynchronous FL to reduce the convergence time.

References

[1] Q. Yang, Y. Liu, Y. Cheng, Y. Kang, T. Chen, and H. Yu, "Federated learning," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 13, no. 3, pp. 1–207, 2019.

[2] L. Lyu, H. Yu, and Q. Yang, "Threats to federated learning: A survey," *arXiv preprint arXiv:2003.02133*, 2020.

[3] H. Li, K. Ota, and M. Dong, "Learning iot in edge: Deep learning for the internet of things with edge computing," *IEEE network*, vol. 32, no. 1, pp. 96–101, 2018.

[4] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial Intelligence and Statistics*, 2017, pp. 1273–1282.

[5] H. B. McMahan, E. Moore, D. Ramage, and B. A. y Arcas, "Federated learning of deep networks using model averaging," 2016.

[6] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy-preserving machine learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1175–1191.

[7] A. Imteaj, U. Thakker, S. Wang, J. Li, and M. H. Amini, "Federated learning for resource-constrained iot devices: Panoramas and state-of-the-art," *arXiv preprint arXiv:2002.10610*, 2020.

[8] A. Imteaj and M. H. Amini, "Distributed sensing using smart end-user devices: pathway to federated learning for autonomous iot," in *2019 International Conference on Computational Science and Computational Intelligence (CSCI)*. IEEE, 2019, pp. 1156–1161.

[9] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konečnỳ, S. Mazzocchi, H. B. McMahan *et al.*, "Towards federated learning at scale: System design," *arXiv preprint arXiv:1902.01046*, 2019.

[10] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[11] WeBank, "Fate: An industrial grade federated learning framework," 2018. [Online]. Available: https://fate.fedai.org

[12] S. Caldas, P. Wu, T. Li, J. Konečnỳ, H. B. McMahan, V. Smith, and A. Talwalkar, "Leaf: A benchmark for federated settings," *arXiv preprint arXiv:1812.01097*, 2018.

[13] T. Ryffel, A. Trask, M. Dahl, B. Wagner, J. Mancuso, D. Rueckert, and J. Passerat-Palmbach, "A generic framework for privacy preserving deep learning," *arXiv preprint arXiv:1811.04017*, 2018.

[14] L. Huang, Y. Yin, Z. Fu, S. Zhang, H. Deng, and D. Liu, "Loadaboost: Loss-based adaboost federated machine learning on medical data," *arXiv preprint arXiv:1811.12629*, 2018.

[15] S. Ge, F. Wu, C. Wu, T. Qi, Y. Huang, and X. Xie, "Fedner: Privacy-preserving medical named entity recognition with federated learning." *CoRR*, 2020.

[16] F. Chen, Z. Dong, Z. Li, and X. He, "Federated meta-learning for recommendation," *arXiv preprint arXiv:1802.07876*, 2018.

[17] T. Qi, F. Wu, C. Wu, Y. Huang, and X. Xie, "Fedrec: Privacy-preserving news recommendation with federated learning," *arXiv*, pp. arXiv–2003, 2020.

[18] M. Ammad-Ud-Din, E. Ivannikova, S. A. Khan, W. Oyomno, Q. Fu, K. E. Tan, and A. Flanagan, "Federated collaborative filtering for privacy-preserving personalized recommendation system," *arXiv preprint arXiv:1901.09888*, 2019.

[19] A. Hard, K. Rao, R. Mathews, S. Ramaswamy, F. Beaufays, S. Augenstein, H. Eichner, C. Kiddon, and D. Ramage, "Federated learning for mobile keyboard prediction," *arXiv preprint arXiv:1811.03604*, 2018.

[20] S. Ramaswamy, R. Mathews, K. Rao, and F. Beaufays, "Federated learning for emoji prediction in a mobile keyboard," *arXiv preprint arXiv:1906.04329*, 2019.

[21] D. Leroy, A. Coucke, T. Lavril, T. Gisselbrecht, and J. Dureau, "Federated learning for keyword spotting," in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2019, pp. 6341–6345.

[22] F. Hartmann, S. Suh, A. Komarzewski, T. D. Smith, and I. Segall, "Federated learning for ranking browser history suggestions," *arXiv preprint arXiv:1911.11807*, 2019.

[23] T. Yang, G. Andrew *et al.*, "Applied federated learning: Improving google keyboard query suggestions," *arXiv:1812.02903*, 2018.

[24] H. B. McMahan, E. Moore *et al.*, "Communication-efficient learning of deep networks from decentralized data," *arXiv:1602.05629*, 2016.

[25] M. Shayan, C. Fung, C. J. Yoon, and I. Beschastnikh, "Biscotti: A ledger for private and secure peer-to-peer machine learning," *arXiv preprint arXiv:1811.09904*, 2018.

[26] C. Fung, C. J. M. Yoon, and I. Beschastnikh, "Mitigating sybils in federated learning poisoning," 2018.

[27] X. Lian, W. Zhang, C. Zhang, and J. Liu, "Asynchronous decentralized parallel stochastic gradient descent," *arXiv preprint arXiv:1710.06952*, 2017.

[28] S. Zheng, Q. Meng, T. Wang, W. Chen, N. Yu, Z.-M. Ma, and T.-Y. Liu, "Asynchronous stochastic gradient descent with delay compensation," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 4120–4129.

[29] C. Xie, S. Koyejo, and I. Gupta, "Asynchronous federated optimization," *arXiv preprint arXiv:1903.03934*, 2019.

[30] N. I. of Standards and Technology, "Emnist dataset." [Online]. Available: https://www.nist.gov/srd/nist-special-database-19