Aergia: leveraging heterogeneity in federated learning systems

Cox, B.A.; Chen, Lydia Y.; Decouchant, J.E.A.P.

DOI
10.1145/3528535.3565238

Publication date
2022

Document Version
Final published version

Published in
Middleware '22: Proceedings of the 23rd conference on 23rd ACM/IFIP International Middleware Conference

Aergia: Leveraging Heterogeneity in Federated Learning Systems

Bart Cox
b.a.cox@tudelft.nl
Delft University of Technology
Delft, Netherlands

Lydia Y. Chen
lydiaychen@ieee.org
Delft University of Technology
Delft, Netherlands

Jérémie Decouchant
j.decouchant@tudelft.nl
Delft University of Technology
Delft, Netherlands

Abstract

Federated Learning (FL) is a popular deep learning approach that prevents centralizing large amounts of data, and instead relies on clients that update a global model using their local datasets. Classical FL algorithms use a central federator that, for each training round, waits for all clients to send their model updates before aggregating them. In practical deployments, clients might have different computing powers and network capabilities, which might lead slow clients to become performance bottlenecks. Previous works have suggested to use a deadline for each learning round so that the federator ignores the late updates of slow clients, or so that clients send partially trained models before the deadline. To speed up the training process, we instead propose Aergia, a novel approach where slow clients (i) freeze the part of their model that is the most computationally intensive to train; (ii) train the unfrozen part of their model; and (iii) offload the training of the frozen part of their model to a faster client that trains it using its own dataset. The offloading decisions are orchestrated by the federator based on the training speed that clients report and on the similarities between their datasets, which are privately evaluated thanks to a trusted execution environment. We show through extensive experiments that Aergia maintains high accuracy and significantly reduces the training time under heterogeneous settings by up to 27% and 53% compared to FedAvg and TiFL, respectively.

CCS Concepts: • **Computing methodologies** → **Distributed artificial intelligence**; • **Computer systems organization** → **Cloud computing**.

Keywords: Federated learning, Task Offloading, Stragglers

Middleware '22, November 7–11, 2022, Quebec, QC, Canada

ACM ISBN 978-1-4503-9340-9/22/11…$15.00
https://doi.org/10.1145/3528535.3565238

ACM Reference Format:
Bart Cox, Lydia Y. Chen, and Jérémie Decouchant. 2022. Aergia: Leveraging Heterogeneity in Federated Learning Systems. In *23rd ACM/IFIP International Middleware Conference (Middleware '22), November 7–11, 2022, Quebec, QC, Canada.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3528535.3565238

1 Introduction

Federated Learning (FL) is a decentralized and inherently privacy-preserving learning paradigm where clients collectively train a machine learning model [3, 22]. During a learning round, a federator selects a subset of the clients that return an update of the global model computed using their local dataset. Upon receiving client updates, the federator aggregates them into a global model update, which is then shared with all clients. Most of existing aggregation algorithms, including FedAvg [22] and FedProx [21], are synchronous, and require the federator to collect all updates from the selected clients before moving to the next training round.

In a practical FL system, clients might have heterogeneous computational resources and possess data that differ both in quantities and class distribution. It has been shown that both resource and data heterogeneity negatively impact the performance of a FL system [12, 14, 21, 40]. First, relying on a mix of weak and strong clients instead of homogeneous clients to train a model can significantly prolong the training time [6]. Second, a classification model trained with federated learning is less accurate when the client datasets are non independently and identically distributed (non-IID) [7].

To mitigate the impact of weak clients, also called stragglers, the state-of-the-art methods attempt to equalize the learning speed amongst the clients by (i) partitioning them based on offline profiling [6], or by (ii) dropping the updates of stragglers during the training rounds [20, 24]. The former approach may fall short in capturing transient heterogeneity caused by applications possibly collocated on the clients, whereas the latter might incur a severe accuracy degradation. Moreover, the impact of stragglers is further aggravated when encountering non-IID data among clients. Indeed, stragglers might possess a unique dataset that is critical to the overall model accuracy. In addition, due to the privacy preserving nature of FL, it is not really possible for the federator to infer the data distribution based only on the clients model updates [21, 33]. To limit the risk of model

divergence, prior studies aggregate the non-IID client data by adding a regularization term, like in FedProx [21], or by estimating their contributions, like in FedNova [33]. However, these works implicitly assume that the client nodes are homogeneous.

In this paper, we aim to accelerate the FL training of convolutional neural networks (CNN) in presence of stragglers and non-IID data. A CNN is composed of convolutional layers and fully connected layers [18], which respectively learn the representation of local data and map the extracted representation into classes. The local training of CNN entails forward and backward passes on both types of layers.

To retain the representation of the unique datasets of stragglers, we advocate to freeze their convolutional layers, and offload the computing and updating of the convolutional layers to strong clients. We propose Aergia[1], a federated learning algorithm that monitors the local training of selected clients and offloads part of the computing task of stragglers to strong clients that have spare and idle capacities. Aergia relies on a client matching algorithm that associates a straggler to a strong node based on an estimated performance gain and on the similarity between their datasets, since blindly offloading local models to nodes that have drastically different data distribution leads to weight divergence [7]. To ensure privacy, data similarities are securely evaluated using the clients' local data distributions (i.e., the number of labels per class) in an Intel SGX enclave [8], which is hosted by the federator.

We implement Aergia in PyTorch as a middleware running on top of Kubernetes. We evaluate Aergia on three datasets, namely MNIST, FMNIST, and Cifar-10, on different network architectures against four previous heterogeneity or non-IID aware aggregation solutions [6, 21, 22, 33]. Our FL systems consist of a mix of 24 weak, medium and strong nodes that use a different number of CPU cores. Our evaluation results show that Aergia achieves the highest accuracy within the lowest training time.

In a nutshell, this paper makes the following contributions:

- We explain how a straggler can offload the training of its model to a strong client.
- We present an algorithm that matches the performance profile and data similarity of clients.
- We design Aergia[2], a federated learning middleware for highly heterogeneous clients and non-IID data that leverages model training offloading and online client matching. Aergia relies on a trusted execution environment (an Intel SGX enclave) so that the federator can evaluate the similarity of client datasets without getting access to their private class distribution.
- We evaluate Aergia on a FL cluster built on top of Kubernetes. Our evaluation results on three datasets and several networks show that Aergia effectively leverages the spare computational capacity of strong clients to achieve high accuracy in low training time.

The remainder of this paper is organized as follows. §2 provides some background on Federated Learning, data and resource heterogeneity, as well as on their impact on training time and accuracy. §3 provides an overview of Aergia, while §4 describes its algorithms and implementations details. §5 presents our performance evaluation. §6 reviews the related work. Finally, §7 concludes this paper.

2 Background and Motivation

In this section, we first recall necessary background on deep learning models, which are core components of the federated learning paradigm, the practical heterogeneity challenges that federated learning faces and their impact on training time and accuracy.

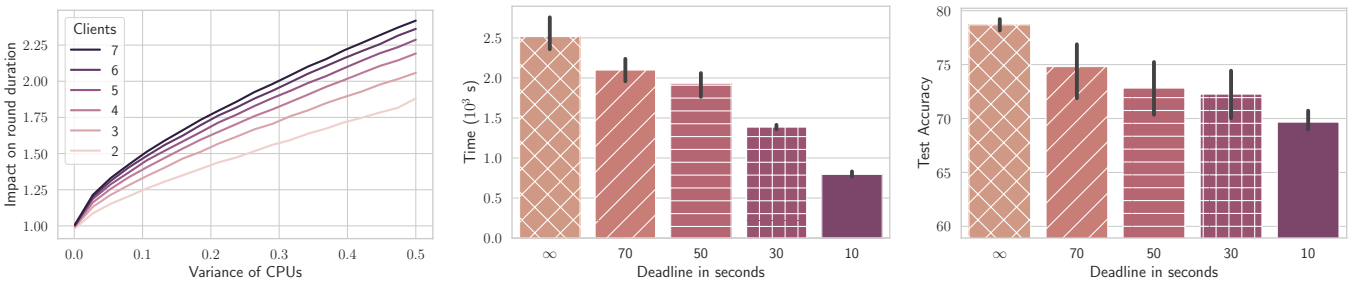
2.1 Premier on Convolutional Neural Networks

The state-of-the-art image classifier follows the structure of convolutional neural networks (CNN) [18], which consist of convolutional and fully connected layers. The former maps the image features into a compact representation, hence they are also referred to as feature layers. The latter are dense fully connected layers that classify the representation into one of the classes. The other difference between these two types of layers is their resource demands [9]: convolutional layers are computationally intensive while fully connected layers are memory intensive. The training time of a client's local model can be divided into two parts: the forward pass that computes the classification outcome of images, and the backward pass that back-propagates the model weights. Consequently, the training time of a typical CNN classifier can thus further be categorized into four parts: (i) ff: forward pass on feature layers, (ii) fc: forward pass on fully connected layers, (iii) bc: backward pass on fully connected layers, and (iv) bf: backward pass on feature layers.

2.2 Federated Learning

Federated learning (FL) [12, 14, 21, 40] is an emerging decentralized learning paradigm where K clients and a federator jointly train a machine learning model in T consecutive rounds while local data stays on premise. In this paper, we specifically consider an image classification model that maps images, x to one of C labels, denoted by y, through a function $f(\boldsymbol{w})$, parameterized by weights $\boldsymbol{w}$. Prior to the training, the federator initializes the model architecture, the objective function, the training algorithm, the training hyperparameters, and the aggregation protocol for the client's

[1] In Greek mythology, Aergia is the personification of sloth, idleness, indolence and laziness.
[2] https://github.com/bacox/fltk



(a) Impact of CPU heterogeneity among clients on training time (multiplicative factor compared to the homogeneous case).

(b) Total training duration in seconds without and with deadlines.

(c) Accuracy in a non-IID scenario without and with deadlines.

Figure 1. Heterogeneous computational powers among the clients increase the duration of the FL training process (Figure 1(a)). One could use deadlines so that the federator discards late updates in a round before starting the next one, which effectively reduces the training time (Figure 1(b)). However, using deadlines badly degrades the model accuracy, in particular in non-IID settings (Figure 1(c)).

local update[3]. We consider convolutional neural networks (CNN) [18] as the classifier model. The clients train the classifier model based on their own real data, which never leaves their premises, whereas the central server iteratively aggregates and distributes models submitted from clients until reaching the global model convergence.

Local Training. In each global round t, the clients receive the latest aggregated global model $f(w(t-1))$ from the federator and use their local data to perform local updates for E epochs, e.g., stochastic gradient decent (SGD) updates [23]. The cross entropy loss function [29, 30, 39] is widely adopted for classification problems. Specifically, a client k aims to find $\boldsymbol{w}_k(t)$ that minimizes the loss function:

$$\min_{\boldsymbol{w}_k(t)} f_k(\boldsymbol{w}_k(t); x_k, y_k),$$

using the n_k local data points (x_k, y_k), where x_k is an input data, e.g., images, and y_k is the class label. Upon finishing the local training, clients send their local model parameters, i.e., $\boldsymbol{w}_k(t)$, to the federator.

Model aggregation. After receiving all model updates from clients, the federator aggregates the clients' model parameters into the latest global model that is returned to the clients in the beginning of the next round. Specifically, at each round t, a subset of K clients is selected to do local training and send back their latest model weights, $w_k(t)$. The aggregation algorithms differ in the frequency and weights in aggregating the local models. FedSGD [4, 23] treats all local models equally, and trains the entire local data in one epoch. The gradients are sent to the federator for aggregation every epoch. To minimize the communication and avoid the divergence of local models, FedAvg [22] lets local models train for multiple epochs and then aggregate the models. Specifically, FedAvg calculates the global model of round t as the weighted average of all K local model weights:

$$\boldsymbol{w}(t) = \sum_{k=1}^{K} \frac{n_k}{\sum_{k=1}^{K} n_k} \boldsymbol{w}_k(t)$$

[3]We note that a variant of FL [26] does not rely on a federator and aggregates the model in a peer-to-peer manner. We do not consider this case here.

2.3 Sources of heterogeneity

Data heterogeneity. Clients possess different and unique privacy-sensitive datasets. A common assumption in the prior art is that client data are identically and independently distributed, which is the so called independent and identically distributed (IID) case. Taking the image data benchmark Cifar-10 [16] as an example, which contains 60,000 images from 10 classes, in the IID case each client would own an equal amount of images that would be equally distributed across classes. Recent studies point out that in practice distributed datasets are highly non-IID, and differ both in size and in distribution [1, 33]. For instance, it is easier to identify clients that own horse images than deer images (both are classes in Cifar-10). Consequently, unique images like deer are owned by a small client subset, whereas common images like horse have a higher probability to be equally distributed across all clients. Such non-IID data distribution, i.e., clients owning data in different quantities and distributions, have been shown to be challenging for FL and detrimental to the accuracy of the global models [12, 14, 40]. The heterogeneity of a non-IID dataset can be captured by its Earth Mover Distance (EMD) [28]. The higher the EMD of a dataset, the higher the heterogeneity of the client data distribution. To mitigate the accuracy degradation in FL due to non-IID data, related studies [21, 33] added regularization terms in the objective function, altering the aggregation algorithms, or augmenting the dataset.

Clients resource heterogeneity. Edge devices are highly heterogeneous in their computing and network resources [34]. Their hardware and software stacks evolve after each generation, i.e., every 5 to 6 years. It is challenging to find an optimal

(deep) learning model for a diversified set of devices. Due to the differences in the type and number of CPU cores and memory provisioning, the computation time of deep models on edge devices vary a lot. Moreover, the network connectivity of edge devices may often be unstable and expensive [7]. Instead of computation, communication is more a bottleneck for edge devices that have poor connectivity, questioning the effectiveness of aggregation algorithms relying on frequent communication. The heterogeneity of clients' resources essentially leads to (highly) unbalanced local training time and also long global training time, because the federator can only aggregate after receiving all the local models, including the slowest one, during the synchronous training. To alleviate either the communication or computation bottleneck, the prior art proposes to have asynchronous communication across global round [2], i.e., the federator aggregates the local updates as soon as a new update is received. The downside of asynchronous training is the risk of slow convergence and low accuracy of the global model. Thus, the state-of-the-art mainly addresses resource heterogeneity by equalizing the communication and computation time across clients so as to maintain the synchronous training and high model accuracy.

2.4 Motivation: Impact of heterogeneity on training time and accuracy

Resource heterogeneity in FL algorithms increases training time. `Aergia` addresses resource heterogeneity thanks to its model freezing and offloading approach. Several works also documented the negative effect of data heterogeneity among clients on FL accuracy [15, 21, 33], which further increases the effect of resource heterogeneity. `Aergia` mitigates this issue by taking into account the similarities between client datasets in its offloading scheduling algorithms.

We evaluate the influence of different degrees of CPU core heterogeneity among clients on the FL training time. We then evaluate the learning degradation that would be incurred when naively equalizing the training time of all rounds based on deadlines. For this experiment, we use the MNIST dataset, and refer the reader to Section 5.1 for additional experimental details.

Figure 1(a) shows how the overall training time of various sizes of FL clusters increases with the variance between the clients CPUs. We consider 1 to 13 clients. We set the average computational capacity per client to be 0.5 CPU, and assign cores to clients with a variation shown in the figures. A high variance implies a high difference among slow and fast clients. We compute the overall training time as follows: we add the time required for any pre-training requirements (such as offline profiling if the algorithm demands it) to the time required for all training rounds. The duration of a training round is measured by the Federator, using its local clock, from the moment the request is sent to the clients until the last participating client responds with its results. One can see that a small perturbation in clients CPU core, i.e., standard deviation, can significantly extend the training time because the federator has to wait for the stragglers because of the synchronous training process. The delay that is added to the total training time grows with the variance of capacity as well as with the cluster size.

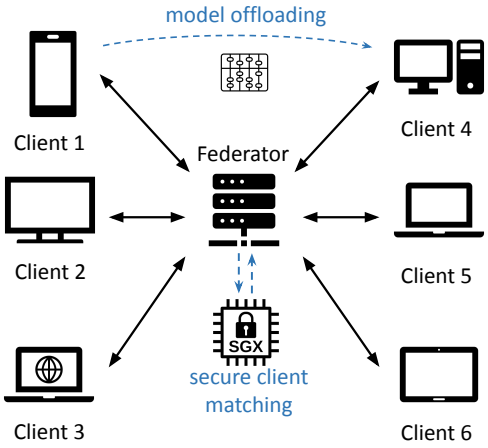


Figure 2. System architecture of `Aergia`.

The state-of-the-art minimizes the impact of resource heterogeneity of clients by equalizing their computation time through estimating clients speed and assigning different amounts of training loads. In Figures 1(b) and 1(c), we evaluate a naive solution - terminating the training per round according to the deadlines. Clients who cannot send back their model weights on time are not included in the aggregation, which effectively bounds the training time but severely degrades accuracy. In contrast, the sophisticated equalization algorithms carefully factor in the trade-off between the accuracy and training time, biasing toward achieving a high accuracy. Motivated by this result, we aim to derive an algorithm that equalizes the training time within and across training rounds.

3 Overview of `Aergia`

3.1 System Model

We consider a traditional federated learning architecture, which Figure 2 illustrates, where multiple clients are connected to a central server, and are also able to directly communicate with each other. We consider heterogeneous settings in the sense that clients might have different computational powers and be interconnected by network links that differ in terms of bandwidth and latency. We consider that the computational load of each client might evolve with time, e.g., because they are running other applications in addition to the training process that we aim at optimizing. We however assume that the network properties remain stable during the training process. We also assume all parties to be honest. We assume that communications are asynchronous but reliable,

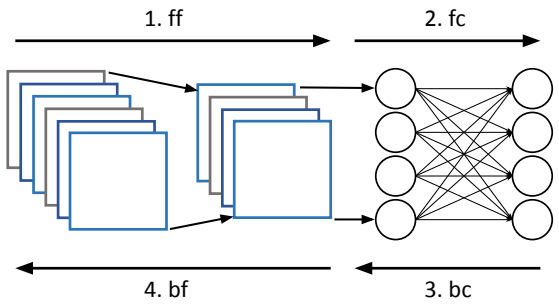


Figure 3. Model training phases during a local update: forward pass feature layers (ff), forward pass classifier layers (fc), backwards pass classifier layers (bc), and backwards pass feature layers (bf).

i.e., that there is no bound on the time it takes for messages to reach their destination but all messages eventually arrive. Clients are equipped with local clocks so that they can measure the time they require to train their models. We do not assume these clocks to be synchronized, but they do need to have similar frequencies. We do not require the federator to make use of a clock. We assume the federator to be correct. It is however equipped with an Intel SGX enclave that the clients can authenticate using remote attestation [8], and to which they send their encrypted and privacy-sensitive dataset class distribution. The enclave provides code integrity and confidentiality of the data it manipulates. The federator relies on the datasets similarity matrix that is computed by the enclave to refine its offloading decisions and maintain high accuracy.

Aergia allows the training process to become aware of the shortcomings and strengths of specific clients and organize their collaboration so that they rely on each other to reduce the overall training time. The central server plays a predominant role in the sense that it coordinates the overall training process and periodically identifies the clients that should offload part of their training tasks to more powerful clients through a model freezing method.

3.2 Leveraging the Heterogeneity of the Learning Phases

A CNN can be split into two sections. The first set of layers are the feature layers and the second part are the classifier layers. During training, for each minibatch the network performs a forward pass and a backward pass, respectively in the forward and in the backward propagation phases, for both sections of the layer. Figure 3 illustrates the four phases of a training cycle: forward pass feature layers (ff), forward pass classifier layers (fc), backwards pass classifier layers (bc), and backwards pass feature layers (bf).

We have profiled the time required by each of the four phases during the network training process with several datasets. To accurately measure the computing time, our profiling time presented here is under the single client scenario. Our results indicate that the time required to perform a cycle is not uniform across phases. Figure 4 shows that the majority of the time is spent in the backwards pass of the feature layers (from 52% to 75%). Aergia therefore focuses on offloading the backwards pass feature layers phase of the weak clients to stronger ones.

Aergia leverages parameter freezing, which is commonly used for transfer learning [25], personalisation tasks [1, 38] and to accelerate training [5, 37]. Freezing parameters in layers eliminates the needs to perform back propagation for the frozen parts of the network.

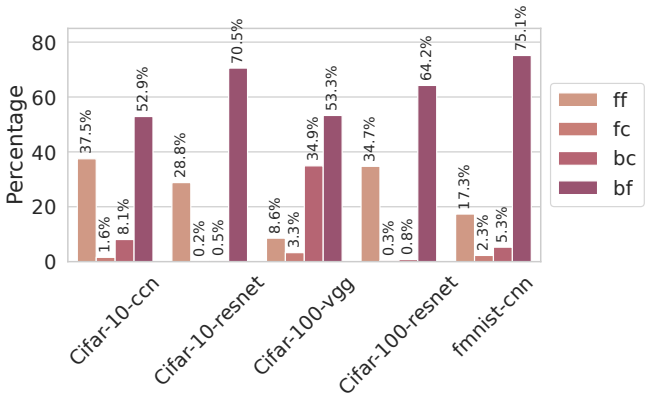


Figure 4. Profiling the different update parts of a network on several datasets shows that the execution time per phase is not uniform during a local update.

3.3 Round training with Model Freezing and Offloading

The client selection and aggregation in Aergia is done in the same way as in typical federated learning. For each local update, the server selects a subset of the devices and sends them the central model. Aergia differs from the standard FL starting from the moment where clients start training locally. In the following, we describe the various tasks executed during a round, starting from the reception of the global model by the clients. Section 4 provides a precise description and the implementation details of these tasks.

Early training and profiling. At the beginning of a training round, clients start by executing complete batches, i.e., they use the 4 training phases, and monitor the speed at which they execute each phase. Clients then report their performance measurements, obtained from the online profiler (which directly uses the clients' clock to measure processing times), to the central server, and continue fully training their model while waiting for the server's instructions.

Centralized scheduling. When it has received the performance measurements of all clients, the server identifies the clients that would slow down the entire training process

by sending their updated model later than others, and computes a schedule where slow clients offload part of their training process to more powerful and data compatible clients. This computation takes the clients dataset similarities into account by leveraging an Intel SGX enclave for privacy. The server indicates to the slow clients that they should offload their model to a stronger client, whose IP address is specified and who will train part of their model for them. The server also informs the stronger clients that should train the model of weaker clients. Messages from the server detail the global training round number so that clients can ignore late messages.

Model freezing, offloading and training. Clients are informed of the offloading they might have to execute according to the schedule the server computed, and the time at which all clients are expected to send their updated model to the server. Weak clients that need to offload their models freeze the first layers of their model, send their model to a stronger client and continue updating their model with a lighter procedure. Strong clients that are executing offloaded tasks return their own fully trained model and the (possibly partially trained) model they might have been updating for a weaker node at the time of the deadline.

Model aggregation. Upon receiving all model updates, the server recombines the models of clients that offloaded their training process: the first layers are received from the stronger client to which their training was offloaded, and the remaining layers are received directly from the weaker client. Based on the reconstructed models and based on the models that were fully trained by clients, the server uses the classical FL averaging method to compute the next global model, and start the next global update.

4 System Details of Aergia

In this section, we detail the key components of Aergia. We first describe Aergia's model freezing and offloading algorithm. We then detail how the federator relies on profiling results communicated by clients to schedule the models freezing and offloading that should happen during a global round. We finally explain how the federator leverages an SGX enclave to compute the clients dataset similarities to refine its schedule.

4.1 Model Freezing and Offloading

Figure 5 illustrates the potential impact of task offloading on the duration of a learning round with heterogeneous clients, and allows us to explain how model freezing and offloading intervene in Aergia. At the beginning of a round, the server sends the current global model to the clients, after which they compute their local model using their own datasets. In this example, clients 3 and 4 are more powerful than clients 1 and 2, which impacts the time it takes them to execute a training phase (numbered from 1 to 4). In this

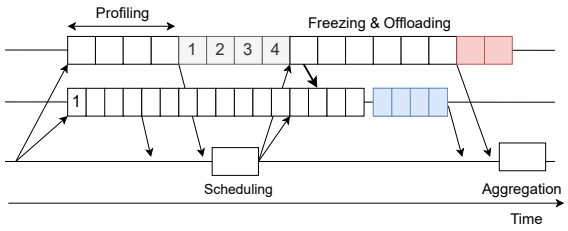


Figure 5. Illustration of model freezing and offloading. Client 1 freezes and offloads its model to client 2 to accelerate the training process.

example, the clients profile their performance over a number of local batch updates (Aergia uses 100 batches), and report their performance metrics to the server afterwards. After having collected all performance measurements, the server determines which clients should freeze the second part of their model and offload its training to other clients. The server then informs the clients to freeze their model and to offload the training of their last layers to a selected strong client. Symmetrically, the federator also communicates with the strong clients about the offloading decision so that they are aware of the identity of the weak client whose model they should train, and know the number of local training batches they should use to train it. While they are waiting for the server's scheduling decision, clients keep training their models using all four phases. In Figure 5, clients 1 and 2 are informed that they should respectively freeze and offload their models to clients 3 and 4. When clients 3 and 4 finish executing their local updates (here after 3 batches), they train the offloaded models they have received on their local datasets. All clients eventually return the local models they train (offloaded or not) to the server, which finally aggregates them. In this example, clients 1 and 2 execute their rounds faster than they would have without offloading because they save the execution of two phases. Note that scheduling decisions are cryptographically signed by the federator for authenticity, and that they contain a monotonically increasing sequence number so that they cannot be replayed and so that messages sent by the federator that arrive late (i.e., in the next round) are ignored.

4.2 Online Profiling

Minimizing the maximum training time of all clients during a local update is equivalent to minimizing the variance between the client training times. To reach this goal, Aergia gathers information about the training speed of each client over the first local updates of a global training round. This information is initially not available to the federator, and might also evolve over time as clients might dedicate a various proportion of their computing power to the learning task. During the profiling phase, clients keep track of the execution time of the forward- and backward propagation of

each layer. These execution times are then communicated to the server and allows it to obtain a clear picture of the current computational power of the clients, and identify the clients that are slowing down the whole learning process. The profiling phase should include sufficiently enough batches to report accurate numbers, but not too many so that there is still headroom to optimize the remaining part of the learning process (whose entire duration depends on client dataset sizes). Aergia profiles the training process during 100 local batch updates out of 1600 local batch updates[4].

4.3 Centralized Scheduling

The federator in Aergia executes a scheduling algorithm that considers the training process of each client as two consecutive tasks that can potentially be executed on different clients. The first task of each client is always executed locally while the second task can be offloaded to another client. Based on the performance metrics gathered by the profiler, the offloading scheduling algorithm decides whether the second task of each client should be offloaded to another client. In Figure 5, clients 1 and 2 respectively offload the second task of their training process to clients 3 and 4. Clients 3 and 4 execute these offloaded tasks as soon as they receive them and are done computing their own model updates. Overall, thanks to the offloaded tasks, the training time of the overall round can be reduced.

When it has received the performance measurements of all clients, the server identifies the clients that would slow down the entire training process by sending their updated model later than others, and computes a schedule where slow clients offload part of their training process to more powerful and data compatible clients. The server directly informs the slow clients that should offload their training process to another client.

The main objective of the server is to minimize computational variance by correctly matching a weak client with a strong client. The central server is during the local training of the clients informed about the training performance of the clients. The clients gather these performance estimates with the online profiler while training locally on the data. Note that the profile has a very low overhead as it only represents 0.58% of the total training time. Using the performance indicators, the central server matches underperforming clients with strong clients following Algorithm 1. We estimate a *mean compute time* for each round based on the performance data of each client that is active in the current round. The central server uses the *mean compute time* (*mct*) as a target time for the offloading schedule. Using *mct* and the performance indicators of each client, the weak and strong clients are identified (Line 13 and 14). Pairs of weak and strong clients are created such that the compute time of the weak clients approaches the initial *mean compute time*. Similarly, the additional offloading work of the strong client should be bounded by *mean compute time*, i.e., its original compute time and offloading time should not exceed the *mean compute time*. Clients are sorted based on their expected training duration. The set of clients who are not able to meet the imposed *mean compute time* are the weak clients. For these clients, a suitable strong client needs to be identified. The scheduling algorithm matches clients starting by the weakest

[4]The number of local upates used for the profiler depends on the size of the dataset and the batch size. For our experiments we use 100 local batch updates to limit the profiling overhead and still achieve a good profiling accuracy.

Algorithm 1 Scheduling models freezing and offloading with m clients

1: **Inputs:**
2: $t_{j,\{1,2,3\}}$, $t_{j,4}$: training times of client j on tasks $(1, 2, 3)$ and 4, for $1 \leq j \leq m$.
3: ru_j: remaining local updates of client j.
4: $S_{i,j}$: Similarity values between clients i and j, describing pair wise data similarity.
5: f: Similarity factor.
6:
7: **Outputs:**
8: *deadlines*: array of times at which a client must send the partially or completely trained models to the server
9: *sending*: an array that details the stronger client to which a client should offload its model to (possibly None).
10:
11: **Algorithm:**
12: $mct = \frac{1}{|M|} \sum_{m=1}^{M} ru_m * (t_{m,\{1,2,3\}} + t_{m,4})$
13: $sending = \{x | \forall x \in M, ru_x * (t_{x,\{1,2,3\}} + t_{x,4}) > mct\}$
14: $receiving = \{y | \forall y \in M, ru_y * (t_{y,\{1,2,3\}} + t_{y,4}) \leq mct\}$
15: $sort\_ascending(sending)$
16: $sort\_descending(receiving)$
17: $deadlines = []$
18: **for** each $c \in sending$:
19: $selected\_client = None$
20: $offloading\_cost = \infty$
21: $op = 0$
22: **for** each $k \in receiving$:
23: $ct, d = calc\_op(t_{c\{1,2,3,4\}}, t_{k\{1,2,3,4\}}, t_{k\{4\}}, ru_c, ru_k)$
24: $cost_{temp} = ct * (1 + \log(S_{c,k} * f + 1))$
25: **if** $cost_{temp} < offloading\_cost$:
26: $offloading\_cost = cost_{temp}$
27: $selected\_client$ = k
28: $op = d$
29: $receiving = receiving - selected\_client$
30: $deadlines[c] = (selected\_client, op)$
31: **if** $receiving = \emptyset$:
32: **break**
33: **return** $(deadlines, sending)$

Algorithm 2 *calc_op*(): Calculating the optimal offloading point (*op*) between two nodes

Inputs:
t_a: training time for client a.
t_B: training time for client b.
x_b: training time of only conv layer for client b.
r_a: remaining local updates of client a.
r_b: remaining local updates of client b.

Outputs:
ct: Estimated duration between two nodes.
d: Number of local updates to be executed before offloading to the other node.

opt_time(t_a, t_b, x_b, r_a, r_b**):**
$ct = \infty$
for $d \in 1 \ldots \min(r_a, r_b)$:
  $current\_ct = \max((r_a - d) * t_a + d * x_b, (r_b - d) * t_b)$
  if $current\_ct > ct$:
    return ct, d
  $ct = current\_ct$
return ct, d

ones because the global training time in a round is determined by the weakest client. For a possible offloading option, the algorithm evaluates the optimal offloading time (Line 23). The optimal offloading time is calculated using the performance indicators of two nodes as can be seen in Algorithm 2. A strong client can only be used one time per round for offloading. Once a strong client is assigned to a weak client, the strong client is removed from the list of possible receiving nodes (Line 29). One can modify Algorithms 1 and 2 to support heterogeneous network transmission latencies and bandwidths, which is straightforward and has been omitted for space reasons.

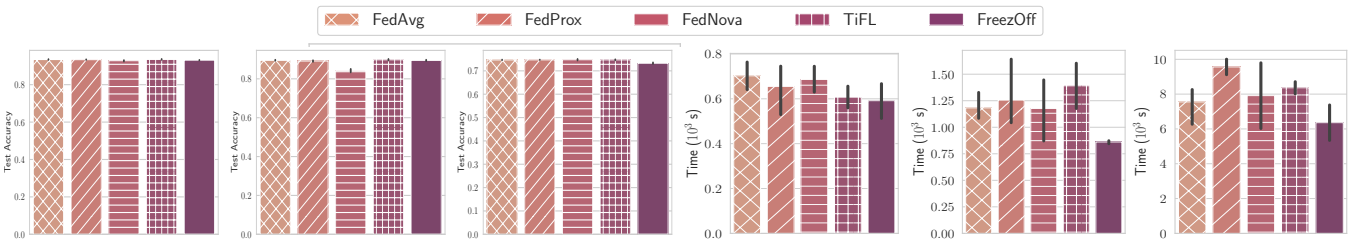
Globally minimizing the time required to train all client models during a FL round can be seen as a $Qm|r_j|C_{max}$ job scheduling problem, which can be solved using dynamic programming [17]. Our algorithm is a variant of the greedy longest-processing-time-first (LPT) algorithm that distributes the training tasks over the clients, which has been demonstrated to produce schedules that are close to optimal [11]. In addition, LPT scales linearly with the number of clients, and therefore does not significantly slow down the federator.

Training procedure. The scheduling algorithm is used by the federator to orchestrate the training process. The actions of the Federator during a training round consist of the following steps. First, the client selection, in the same manner as with `FedAvg`, is performed. The selected clients are by the Federator instructed to start training for T rounds with the online profiler active. Each client after P number of local batch updates informs the federator about the performance metrics that are gathered with the profiler. Simultaneously, the clients stop the online profiler and continue training until further instruction. With all the performance metrics that the federator receives, an offloading schedule is created using Algorithm 1. The clients are informed by the federator about the offloading schedule. The model aggregation is performed when the federator has received a training result from all selected clients. This process is repeated for T rounds.

4.4 Refining Schedules with Data Heterogeneity

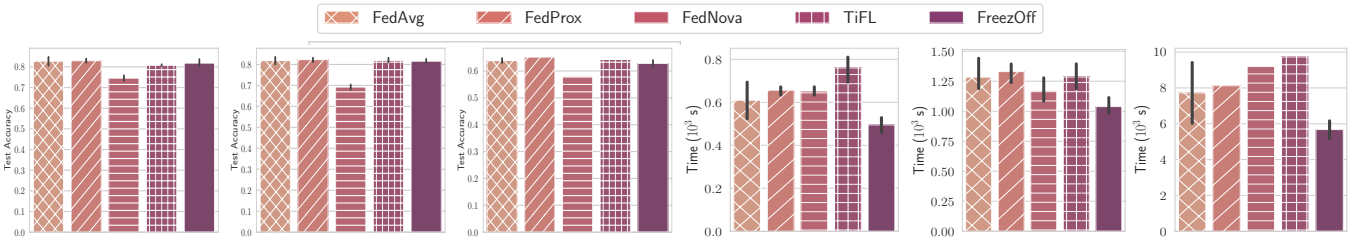
So far, the scheduling algorithm we have presented does not take into account the fact that clients might possess non-IID local datasets. In practice, offloading the model of a client to a stronger client with a vastly different dataset would be detrimental to the training accuracy. The central server therefore also takes into account the similarity of two datasets to compute the model freezing and offloading that should take place during a round. The similarity between datasets is computed by the federator using the data distribution over the classes. We implemented this process using a trusted execution environment, namely Intel SGX, but it could also be implemented using cryptographic methods such as homomorphic encryption. We demonstrate that using dataset similarities allows `Aergia` to improve its model accuracy and leave such privacy-preserving extensions to future work. The local data distribution of a client is privacy sensitive, therefore, we calculate the similarity matrix S inside a trusted execution environment (i.e., an SGX enclave) of the federator node. Clients send an encrypted vector that contains the number of labels per class to the enclave. The trusted execution environment computes the pair-wise similarity between pairs of clients based on the EMD metric. The result is a similarity matrix S, a $m \times m$ triangular matrix where m is the number of clients. The similarity matrix S enables `Aergia` to improve the offloading algorithm without publicly sharing sensitive information.

The cost function on line 24 of Algorithm 1 takes the data similarity of two clients into account. The similarity matrix S is calculated before the training starts, and it contains the pair wise similarity of the local data using the EMD metric. Variable f on line 24 of Algorithm 1 is used to control the effect of the inter client similarity. With $f = 0$, the scheduling only takes the performance metrics into account and ignores the local data similarities, as discussed in section 4.3. As soon as $f > 0$, the inter-client data similarity is taken into consideration while creating the schedule. A larger f increases the influence of data similarity when determining the offloading decisions. Using this simple objective function, the scheduling algorithm associates each weak client to a suitable stronger client that will partially train its model on a dataset that is sufficiently similar to its own.



(a) Accuracy MNIST **(b)** Accuracy FMNIST **(c)** Accuracy Cifar-10 **(d)** Time MNIST **(e)** Time FMNIST **(f)** Time Cifar-10

Figure 6. IID Data set. The time reported in subfigures 6(d), 6(e), 6(f) is the training time in seconds used to complete 100 communication rounds.



(a) Accuracy MNIST **(b)** Accuracy FMNIST **(c)** Accuracy Cifar-10 **(d)** Time MNIST **(e)** Time FMNIST **(f)** Time Cifar-10

Figure 7. Non-IID Data set. The time reported in subfigures 7(d), 7(e), 7(f) is the training time in seconds used to complete 100 communication rounds.

5 Evaluation

In this section, we summarize the evaluation results of Aergia on a testbed, where heterogeneous clients own diversified data sets and learn the CNN classifier in a federated manner. We compare the overall training time and accuracy against four state of the art solutions on three datasets. We also conduct a sensitivity analysis to demonstrate the effectiveness of Aergia under different degrees of non-IIDness.

5.1 Evaluation Setup

The central federator performs the default FL tasks (client selection and model aggregation) and also computes the client matching for the offloading of model training. During the local training, the client sends the federator performance metrics of the local training. With this information, the federator can spot stragglers that slow down the training process. Using these performance metrics, the federator matches stragglers with powerful clients in order to offload computational tasks. The matched clients communicate directly and transfer the tasks to each other.

Baselines. We compare Aergia to four baselines: FedAvg [22], FedNova [33], FedProx [21], and TiFL [6]. FedAvg, FedNova, and FedProx implicitly assume that client nodes are homogeneous and that communication is synchronous. FedProx minimizes the amount of drift a client can obtain during local training to improve the convergence when learning over non-IID data. FedNova normalizes the client updates at the aggregation stage. TiFL handles stragglers on a global level by grouping the clients in tiers based on their computation speed.

Testbed. The evaluations are performed on a testbed aimed to run FL systems in a distributed environment. The testbed is implemented in Python and built on top of PyTorch. Each node is fully isolated from each other and can only communicate through messages. All communication is asynchronous and peer to peer and based on RPC. This means that nodes can message each other directly without relying on the Federator as a relay (Most FL systems assume a star network). In our evaluation we assume a fully connected network. The testbed allows all nodes in the system to run independent and isolated from each other. We use an Aurora R13 desktop with a 5.20 GHz i9 CPU with 24 cores and 64 GB RAM. We use up to one client per CPU core. We implemented the dataset similarity computation that runs in an Intel SGX enclave in C++ and used Graphene [31].

Datasets. We evaluate Aergia using three datasets: MNIST, FMNIST and Cifar-10. The MNIST and FMNIST datasets contain 60,000 training images and 10,000 test images with a dimension of 28x28 pixels. The Cifar-10 dataset has a total of 60,000 images of 32x32 pixels split over 50,000 training images and 10,000 test images.

Networks There are three networks used to evaluate the datasets. For the MNIST dataset we use a three layer CNN with two convolutional layers and a single fully connected

layer. For FMNIST we use the same model as used to evaluate MNIST. Lastly, an eight layer CNN is used to evaluate Cifar-10 consisting of six convolutional layers and two fully connected layers.

Heterogeneous Resource Setup. In the real world, machines are rarely equal in computing power. Therefore, using large discrete steps, such as 0.25, 0.5, 0.75 and 1.0 CPU, to define resource heterogeneity between nodes is not realistic. We use a total of 24 cores to evaluate all the algorithms we consider. To reproduce a real-world scenario, the CPU speed of each client is selected uniformly at random as a fraction that ranges between 0.1 and 1.0 of the original CPU speed. We use Docker containers to isolate the nodes on different cores and control their CPU speed. This allows for an realistic approximation of the aforementioned scenario.

Heterogeneous Data Distribution. The clients that participate in the Federated Learning process may have non-uniform data distributions. To evaluate performance under non-IID heterogeneity, clients sample 3 classes out of the 10 available.Clients are then biased towards their local dataset during training. Local client datasets are disjoint, i.e., they do not share images. This approach is used for all three datasets when evaluating the non-IID scenario.

5.2 Accuracy and training times

We divide our evaluation into two scenarios: IID and non-IID on three datasets. We report the accuracy and training time in Figures 6-7.

IID: When the local training data has a IID distribution, the effects of the resource heterogeneity are negligible. In figure 6(b)-a we compare the accuracy of `FedAvg`, `FedNova`, `FedProx`, `TiFL`, and `Aergia` on the FMNIST dataset. The accuracy after 100 rounds is comparable among the aforementioned algorithms. In contrary to the accuracy values, the results of the real-time duration show that there is a noticeable difference between `Aergia` and the rest. On average, `Aergia` is able to do the same amount of training as `FedAvg` and `TiFL` in 27% and 45 % less time respectively.

Non-IID: The impact of non-IID data is intensified by the system heterogeneity and leads to a significant longer training times. This effect can be seen in Figure 7(f). `Aergia` is able to reduce the training time per round and thereby reduce the overall global training time by 27% compared to `FedAvg` and 53% compared to `TiFL`. The same trend is visible in Figures 7(d) and 7(e). The accuracy reached in non-IID scenarios is comparable between the `Aergia` and the other algorithms with the exception of `FedNova`. Figures 7(e) and 7(f) show that `TiFL` is unable to prevent the slowdown of the FL system, in contrary to what is reported in [6]. We think this is caused by the higher variation of CPU power in this scenario. `TiFL` is in this scenario not able to reduce the variance in CPU using their tier solution due to a higher intra-tier CPU variance. Overall we observe that `Aergia` is able to reduce the training time up to 53% while achieving the comparable accuracy as the state of the art non-IID aware aggregation algorithms.

To better understand the performance advantages of `Aergia`, we zoom into the detailed performance of FMNIST, i.e., the time distribution per training round. Figure 8 shows the density of the rounds duration during the training process for `Aergia` and the baselines we consider. `Aergia`'s distribution is shifted to the left compared to all baselines, which indicates its ability to minimize the duration of rounds during training.

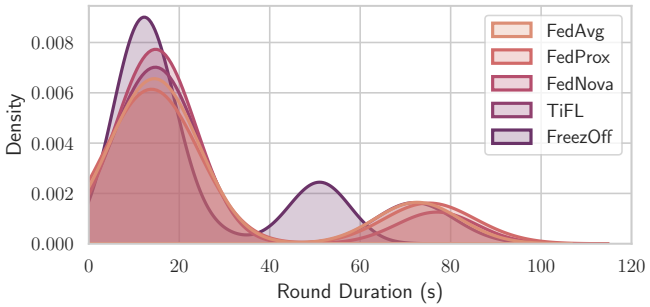


Figure 8. Density of the duration of rounds.

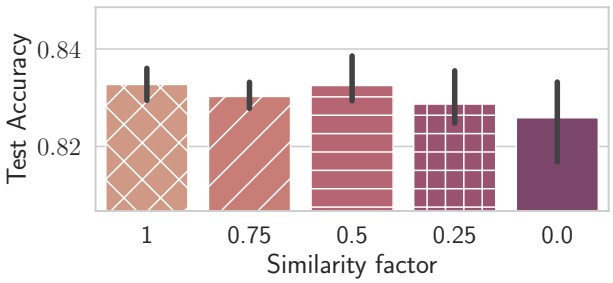
5.3 Rounds duration and impact of the similarity factor

`Aergia` uses the similarity factor f in Line 24 in Algorithm 1 to control the impact of the inter-client data similarity. A value of 0 means that the inter-client similarity has no effect on the scheduling while $f > 0$ increases the impact of similarity in Algorithm 1. A higher value of f restricts the number of favourable strong offloading options. Figure 9(b) shows that the average round time decreases when similarity is not taken into account. A low values of f has impact on the global model accuracy as can be seen in Figure 9(a). A positive value of f increases the global model accuracy in `Aergia`.

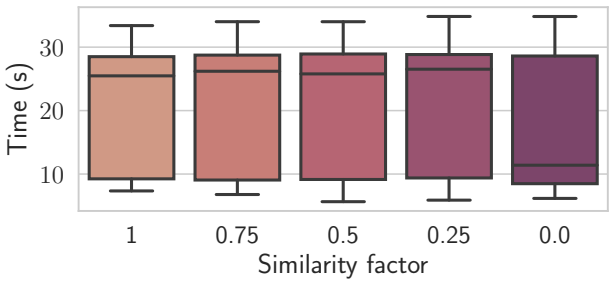
We evaluate learning FMNIST on 24 clients, 3 of which are selected in each round. Figures 9(a) and 9(b) report the impact of the similarity factor parameter on accuracy and on the training time, respectively. With a similarity factor of 0, the scheduling algorithm ignores the inter-client similarity and uses performance indicators only. With a positive similarity factor, the inter-client similarity has more impact on the scheduling decisions. Increasing the similarity factor can restrict the available offloading options and thereby increases the average round time.

5.4 Impact of the Degree of Non-IIDness

The degree of data non-IIDness has an impact on the convergence speed during training. We control the number of classes a client can own as a measure to evaluate and create non-IID data. The lower the number of sampled classes the



(a) Impact of the client similarity factor (parameter f in Algorithm 1) on accuracy when offloading.



(b) Impact of the client similarity factor (parameter f in Algorithm 1) on the training time.

Figure 9. Impact of similarity factor on accuracy and training time.

higher the degree of non-IID in the clients local data. For instance, non-IID(2) indicates each client only has 2 classes of data points out of 10 available classes. In Figure 10 we show the effect of the degree of non-IIDness on the Aergia's algorithm. All variations train for the same amount of rounds. The difference in completion time is low but the difference in accuracy is apparent. A high degree of non-IID data results in a decrease of model accuracy. This is comparable to previously reported results [6].

Profiler The profiling time of the online profiler impacts the error in the performance indicators used for scheduling. A longer profiling time offers better performance indicators, but a longer profiling time reduces the training speed and increases the training time. The profiler has a negligible overhead of 0.22% ± 0.09 on average for all models.

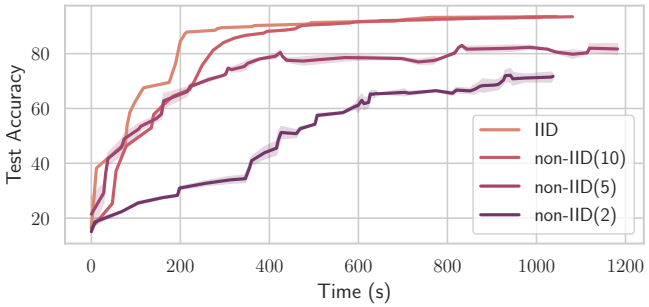


Figure 10. Test accuracy depending on the level of dataset non-IIDness measured during the training process. The level of non-IID limits the number of sampled classes in a clients' local training data. The levels of non-IIDness follow the setup used in [6].

6 Related Work

Devices may differ in computational and communication capabilities due to hardware (CPU, memory) and network connectivity. Challenges such as straggler mitigation, client drift, and the effect of non-IID data are exacerbated by the aforementioned system-level characteristics. FedProx [21] limits how far the local model of the client is allowed to drift from the global model. The objective function is altered with a parameter μ to penalize drift from the global model. Their theory shows that choosing $\mu > 0$ should improve convergence when learning over non-IID data, but practice shows that this is not the case for non-IID label distributions. The solution of Li et al. [21] does not account for stragglers.

	Data heterogeneity aware	Resource heterogeneity aware	Minimize training time
FedAvg [22]	-	-	✗
FedProx [21]	+	-	✗
FedNova [33]	+	-	✗
TiFL [6]	+	+	✓
Aergia	++	++	✓

Table 1. FL solutions for heterogeneous settings.

FedNova [33] limits the effect of heterogeneous clients in the system by altering the global aggregation rule. Clients that perform more steps return a larger update and thus more significantly impact the global model, even when the datasets of all clients have the same size. Although Wang et al. [33] mitigate the effect that heterogeneous nodes have on the model accuracy, it is not able to remove the under representation of stragglers in federated learning systems. Table 1 summarizes our analysis of the existing works that consider heterogeneous client resources or datasets, and shows that Aergia is the only algorithm that minimizes the actual training time, thanks to its model freezing and offloading mechanisms, and maintains high accuracy even in presence of non-IID datasets, thanks to its similarity computation that is executed in an SGX enclave.

6.1 Task Offloading

Dong et al. [10] use an edge server equipped with a Trusted Execution Environment (TEE) to offload tasks of edge devices. Because of the limited capabilities of the TEE, this

method is only tested on the MNIST dataset on a very small network. Besides that, the data is also randomly (but equally) distributed over the clients. The main goal of this work is to mitigate the straggler problem by utilizing the more powerful edge servers for task offloading. EAFL [13] offloads federated learning tasks from edge clients to nearby edge servers. To lighten to computational burden of weak clients, EAFL partially offloads data to an edge server, while `Aergia` offloads models, which is more privacy preserving. FedAdapt [35] leverages split learning to allow an IoT device to train its model with the help of the central server by exchanging partially trained models during each round. In comparison, we leverage model freezing and allow clients to offload training tasks to each other, which is more scalable, and consider the data heterogeneity issue, without which accuracy would be decreased.

6.2 Federated Learning with deadlines

One solution to mitigate stragglers is for the federated learning system to impose a deadline for the clients to submit their locally trained models. While this reduces the overall idle in the whole system, it often diminishes the contribution of the straggler by dropping late submission. Li et al. [20] use the imposed deadline to tune the local settings such as CPU frequency on each client to meet the deadline with the least amount of energy consumption. This approach lowers the amount of dropped clients, however the effect of non-IID data is not considered. Nishio and Yonetani [24] use resource aware client selection (FedCS) to limit the amount of participating stragglers. They show that this solution works mainly works in an IID data settings in large networks. When dealing with non-IID data, models trained with FedCS incur a significant drop in accuracy.

6.3 Stragglers

Different techniques can be used to minimize the impact of stragglers. `TiFL` [6] groups clients in tiers based on their performance characteristics. Each round a different tier is chosen for client selection. This reduces the variance between training times within each round.

Sageflow [27] uses periodic aggregation rounds to limit the decrease of training time per round caused by stragglers. Late contributions send in by stragglers are incorporated into the global model corrected with a staleness factor. The downside of Sageflow is that the method uses public data distributed across the clients.

Lee et al. [19] use adaptive deadlines to accelerate the convergence to the global model. By calculating a deadline per round based on the participating devices, the mean idle time per device is minimized. This solution still can cause stragglers to be dropped by the system, since the deadline is calculated based on the fastest participating client plus the tolerated waiting time. The authors note that their solution has no guarantees when applied on non-IID data.

6.4 Parameter freezing

Parameter freezing is a well known technique in transfer learning that is used to fine tune the model. An alternative use for Parameter Freezing is to speed up the training process. Chen et al. [7] limit the communication between the federated and edge device by freezing parameters in the early stage of the training process. However, challenges are that the local parameters excluded from global synchronization may diverge on different clients, and meanwhile some parameters may only temporally stabilize. Adaptive Parameter Freezing (APF) proposes to fix (freeze) the non-synchronized stable parameters in intermittent periods.

In order to reduce the communication cost, parameter freezing can be used. Brock et al. [5] use parameter freezing to speed up the computations by slowly freezing out the first few layers. This avoids the cost of computing the gradients and speeds up convergence. This approach is coarse and degrades the accuracy of model in spite of a speedup in training. Chen et al. [7] use parameter freezing to lower the communication cost when sending the model of weights. By freezing stale parameters the number model weights that are shared with the server are reduced. While this significantly reduces data transfers, it does not alleviate the computational burden of stragglers.

Veit and Belongie [32] use partial execution of networks to speed up the model execution during inference. By excluding some layers in the network from execution based on the input image, the inference execution time is reduced while retaining accuracy. Similarly, Wu et al. [36] use layer dropping in residual networks to create dynamic execution path to reduce the execution time during inference.

7 Conclusion

In practical settings, Federated Learning (FL) is largely impacted by the heterogeneity of clients, as each training round needs to wait for slow clients, which are also called stragglers. In this work, we have proposed `Aergia` a novel FL algorithm that leverages model freezing and offloading. In this protocol, slow clients freeze the first layers of their model and offload it to a faster client that trains these layers using its own dataset. `Aergia` uses a simple, yet fast and scalable, scheduling algorithm to regularly organize the offloading of models between clients. The central server uses its own dataset to decide which clients have compatible datasets to maintain high accuracy when organizing model freezes and offloadings. Our experiments on three datasets demonstrate that `Aergia` reduces the overall training time by up to 53% and maintains high accuracy.

References

[1] Manoj Ghuhan Arivazhagan, Vinay Aggarwal, Aaditya Kumar Singh, and Sunav Choudhary. 2019. Federated Learning with Personalization Layers. *CoRR* abs/1912.00818 (2019). arXiv:1912.00818 http://arxiv.org/abs/1912.00818

[2] Dmitrii Avdiukhin and Shiva Kasiviswanathan. 2021. Federated Learning under Arbitrary Communication Patterns. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event* (July 18 - 24, 2021) *(Proceedings of Machine Learning Research)*, Marina Meila and Tong Zhang (Eds.), Vol. 139. PMLR, 425–435. https://proceedings.mlr.press/v139/avdiukhin21a.html

[3] Kallista A. Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloé Kiddon, Jakub Konečný, Stefano Mazzocchi, Brendan McMahan, Timon Van Overveldt, David Petrou, Daniel Ramage, and Jason Roselander. 2019. Towards Federated Learning at Scale: System Design. In *Proceedings of Machine Learning and Systems 2019, MLSys 2019, Stanford, California, USA, March 31 - April 2, 2019*, Ameet Talwalkar, Virginia Smith, and Matei Zaharia (Eds.). mlsys.org, 374–388. https://proceedings.mlsys.org/book/271.pdf

[4] Kallista A. Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. 2017. Practical Secure Aggregation for Privacy-Preserving Machine Learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, Texas, USA, October 30 - November 03, 2017*, Bhavani Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu (Eds.). ACM, 1175–1191. https://doi.org/10.1145/3133956.3133982

[5] Andrew Brock, Theodore Lim, James M. Ritchie, and Nick Weston. 2017. FreezeOut: Accelerate Training by Progressively Freezing Layers. *CoRR* abs/1706.04983 (2017). arXiv:1706.04983 http://arxiv.org/abs/1706.04983

[6] Zheng Chai, Ahsan Ali, Syed Zawad, Stacey Truex, Ali Anwar, Nathalie Baracaldo, Yi Zhou, Heiko Ludwig, Feng Yan, and Yue Cheng. 2020. TiFL: A Tier-based Federated Learning System. In *HPDC '20: The 29th International Symposium on High-Performance Parallel and Distributed Computing, Stockholm, Sweden, June 23-26, 2020*, Manish Parashar, Vladimir Vlassov, David E. Irwin, and Kathryn Mohror (Eds.). ACM, 125–136. https://doi.org/10.1145/3369583.3392686

[7] Chen Chen, Hong Xu, Wei Wang, Baochun Li, Bo Li, Li Chen, and Gong Zhang. 2021. Communication-Efficient Federated Learning with Adaptive Parameter Freezing. In *41st IEEE International Conference on Distributed Computing Systems, ICDCS 2021, Washington DC, USA, July 7-10, 2021*. IEEE, 1–11. https://doi.org/10.1109/ICDCS51616.2021.00010

[8] Victor Costan and Srinivas Devadas. 2016. Intel SGX Explained. *IACR Cryptol. ePrint Arch.* (2016), 86. http://eprint.iacr.org/2016/086

[9] Bart Cox, Jeroen Galjaard, Amirmasoud Ghiassi, Robert Birke, and Lydia Y. Chen. 2021. Masa: Responsive Multi-DNN Inference on the Edge. In *19th IEEE International Conference on Pervasive Computing and Communications, PerCom 2021, Kassel, Germany, March 22-26, 2021*. IEEE, 1–10. https://doi.org/10.1109/PERCOM50583.2021.9439111

[10] Shifu Dong, Deze Zeng, Lin Gu, and Song Guo. 2020. Offloading Federated Learning Task to Edge Computing with Trust Execution Environment. In *17th IEEE International Conference on Mobile Ad Hoc and Sensor Systems, MASS 2020, Delhi, India, December 10-13, 2020*. IEEE, 491–496. https://doi.org/10.1109/MASS50613.2020.00066

[11] Ronald L. Graham. 1969. Bounds on Multiprocessing Timing Anomalies. *SIAM Journal of Applied Mathematics* 17, 2 (1969), 416–429. https://doi.org/10.1137/0117039

[12] Kevin Hsieh, Amar Phanishayee, Onur Mutlu, and Phillip B. Gibbons. 2020. The Non-IID Data Quagmire of Decentralized Machine Learning. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event (Proceedings of Machine Learning Research)*, Vol. 119. PMLR, 4387–4398. http://proceedings.mlr.press/v119/hsieh20a.html

[13] Zhongming Ji, Li Chen, Nan Zhao, Yunfei Chen, Guo Wei, and F. Richard Yu. 2021. Computation Offloading for Edge-Assisted Federated Learning. *IEEE Trans. Veh. Technol.* 70, 9 (2021), 9330–9344. https://doi.org/10.1109/TVT.2021.3098022

[14] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista A. Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D'Oliveira, Hubert Eichner, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaïd Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konečný, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Hang Qi, Daniel Ramage, Ramesh Raskar, Mariana Raykova, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. 2021. Advances and Open Problems in Federated Learning. *Found. Trends Mach. Learn.* 14, 1-2 (2021), 1–210. https://doi.org/10.1561/2200000083

[15] Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank J. Reddi, Sebastian U. Stich, and Ananda Theertha Suresh. 2020. SCAFFOLD: Stochastic Controlled Averaging for Federated Learning. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event (Proceedings of Machine Learning Research)*, Vol. 119. PMLR, 5132–5143. http://proceedings.mlr.press/v119/karimireddy20a.html

[16] Alex Krizhevsky and Geoffrey Hinton. 2009. *Learning multiple layers of features from tiny images.* Technical Report 0. University of Toronto, Toronto, Ontario.

[17] Eugene L. Lawler, Jan Karel Lenstra, Alexander H. G. Rinnooy Kan, and David B. Shmoys. 1993. Chapter 9 Sequencing and scheduling: Algorithms and complexity. In *Logistics of Production and Inventory*, Stephen C. Graves, Alexander H. G. Rinnooy Kan, and Paul Herbert Zipkin (Eds.). Handbooks in Operations Research and Management Science, Vol. 4. North-Holland, 445–522. https://doi.org/10.1016/s0927-0507(05)80189-6

[18] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324. https://doi.org/10.1109/5.726791

[19] Jaewook Lee, Haneul Ko, and Sangheon Pack. 2022. Adaptive Deadline Determination for Mobile Device Selection in Federated Learning. *IEEE Trans. Veh. Technol.* 71, 3 (2022), 3367–3371. https://doi.org/10.1109/TVT.2021.3136308

[20] Li Li, Haoyi Xiong, Zhishan Guo, Jun Wang, and Cheng-Zhong Xu. 2019. SmartPC: Hierarchical Pace Control in Real-Time Federated Learning System. In *IEEE Real-Time Systems Symposium, RTSS 2019, Hong Kong, SAR, China, December 3-6, 2019*. IEEE, 406–418. https://doi.org/10.1109/RTSS46320.2019.00043

[21] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. 2020. Federated Optimization in Heterogeneous Networks. In *Proceedings of Machine Learning and Systems 2020, MLSys 2020, Austin, Texas, USA, March 2-4, 2020*, Inderjit S. Dhillon, Dimitris S. Papailiopoulos, and Vivienne Sze (Eds.). mlsys.org, 429–450. https://proceedings.mlsys.org/book/316.pdf

[22] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. 2017. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017, 20-22 April 2017, Fort Lauderdale, Florida, USA (Proceedings of Machine Learning Research)*, Aarti Singh and Xiaojin (Jerry) Zhu (Eds.), Vol. 54. PMLR, 1273–1282. http://proceedings.mlr.press/v54/mcmahan17a.html

[23] H. Brendan McMahan, Eider Moore, Daniel Ramage, and Blaise Agüera y Arcas. 2016. Federated Learning of Deep Networks using Model Averaging. *CoRR* abs/1602.05629 (2016). arXiv:1602.05629 http://arxiv.org/abs/1602.05629

[24] Takayuki Nishio and Ryo Yonetani. 2019. Client Selection for Federated Learning with Heterogeneous Resources in Mobile Edge. In *2019 IEEE International Conference on Communications, ICC 2019, Shanghai, China, May 20-24, 2019*. IEEE, 1–7. https://doi.org/10.1109/ICC.2019.8761315

[25] Maxime Oquab, Léon Bottou, Ivan Laptev, and Josef Sivic. 2014. Learning and Transferring Mid-level Image Representations Using Convolutional Neural Networks. In *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, Ohio, USA, June 23-28, 2014*. IEEE Computer Society, 1717–1724. https://doi.org/10.1109/CVPR.2014.222

[26] Róbert Ormándi, István Hegedüs, and Márk Jelasity. 2013. Gossip learning with linear models on fully distributed data. *Concurr. Comput. Pract. Exp.* 25, 4 (2013), 556–571. https://doi.org/10.1002/cpe.2858

[27] Jungwuk Park, Dong-Jun Han, Minseok Choi, and Jaekyun Moon. 2021. Sageflow: Robust Federated Learning against Both Stragglers and Adversaries. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan (Eds.). 840–851. https://proceedings.neurips.cc/paper/2021/hash/076a8133735eb5d7552dc195b125a454-Abstract.html

[28] Yossi Rubner, Carlo Tomasi, and Leonidas J. Guibas. 1998. A Metric for Distributions with Applications to Image Databases. In *Proceedings of the Sixth International Conference on Computer Vision (ICCV-98), Bombay, India, January 4-7, 1998*. IEEE Computer Society, 59–66. https://doi.org/10.1109/ICCV.1998.710701

[29] Shizhao Sun, Wei Chen, Liwei Wang, Xiaoguang Liu, and Tie-Yan Liu. 2016. On the Depth of Deep Neural Networks: A Theoretical View. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, Dale Schuurmans and Michael P. Wellman (Eds.). AAAI Press, 2066–2072. http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12073

[30] Yichuan Tang. 2013. Deep Learning using Support Vector Machines. *CoRR* abs/1306.0239 (2013). arXiv:1306.0239 http://arxiv.org/abs/1306.0239

[31] Chia-che Tsai, Donald E. Porter, and Mona Vij. 2017. Graphene-SGX: A Practical Library OS for Unmodified Applications on SGX. In *2017 USENIX Annual Technical Conference, USENIX ATC 2017, Santa Clara, California, USA, July 12-14, 2017*, Dilma Da Silva and Bryan Ford (Eds.). USENIX Association, 645–658. https://www.usenix.org/conference/atc17/technical-sessions/presentation/tsai

[32] Andreas Veit and Serge J. Belongie. 2018. Convolutional Networks with Adaptive Inference Graphs. In *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part I (Lecture Notes in Computer Science)*, Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss (Eds.), Vol. 11205. Springer, 3–18. https://doi.org/10.1007/978-3-030-01246-5_1

[33] Jianyu Wang, Qinghua Liu, Hao Liang, Gauri Joshi, and H. Vincent Poor. 2020. Tackling the Objective Inconsistency Problem in Heterogeneous Federated Optimization. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (Eds.). 7611–7623. https://proceedings.neurips.cc/paper/2020/hash/564127c03caab942e503ee6f810f54fd-Abstract.html

[34] Carole-Jean Wu, David Brooks, Kevin Chen, Douglas Chen, Sy Choudhury, Marat Dukhan, Kim M. Hazelwood, Eldad Isaac, Yangqing Jia, Bill Jia, Tommer Leyvand, Hao Lu, Yang Lu, Lin Qiao, Brandon Reagen, Joe Spisak, Fei Sun, Andrew Tulloch, Peter Vajda, Xiaodong Wang, Yanghan Wang, Bram Wasti, Yiming Wu, Ran Xian, Sungjoo Yoo, and Peizhao Zhang. 2019. Machine Learning at Facebook: Understanding Inference at the Edge. In *25th IEEE International Symposium on High Performance Computer Architecture, HPCA 2019, Washington, DC, USA, February 16-20, 2019*. IEEE, 331–344. https://doi.org/10.1109/HPCA.2019.00048

[35] Di Wu, Rehmat Ullah, Paul Harvey, Peter Kilpatrick, Ivor T. A. Spence, and Blesson Varghese. 2021. FedAdapt: Adaptive Offloading for IoT Devices in Federated Learning. *CoRR* abs/2107.04271 (2021). arXiv:2107.04271 https://arxiv.org/abs/2107.04271

[36] Zuxuan Wu, Tushar Nagarajan, Abhishek Kumar, Steven Rennie, Larry S. Davis, Kristen Grauman, and Rogério Schmidt Feris. 2018. BlockDrop: Dynamic Inference Paths in Residual Networks. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, Utah, USA, June 18-22, 2018*. Computer Vision Foundation / IEEE Computer Society, 8817–8826. https://doi.org/10.1109/CVPR.2018.00919

[37] Xueli Xiao, Thosini Bamunu Mudiyanselage, Chunyan Ji, Jie Hu, and Yi Pan. 2019. Fast Deep Learning Training through Intelligently Freezing Layers. In *2019 International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), iThings/GreenCom/CPSCom/SmartData 2019, Atlanta, Georgia, USA, July 14-17, 2019*. IEEE, 1225–1232. https://doi.org/10.1109/iThings/GreenCom/CPSCom/SmartData.2019.00205

[38] Jie Zhang, Song Guo, Xiaosong Ma, Haozhao Wang, Wenchao Xu, and Feijie Wu. 2021. Parameterized Knowledge Transfer for Personalized Federated Learning. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan (Eds.). 10092–10104. https://proceedings.neurips.cc/paper/2021/hash/5383c7318a3158b9bc261d0b6996f7c2-Abstract.html

[39] Zhilu Zhang and Mert R. Sabuncu. 2018. Generalized Cross Entropy Loss for Training Deep Neural Networks with Noisy Labels. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett (Eds.). 8792–8802. https://proceedings.neurips.cc/paper/2018/hash/f2925f97bc13ad2852a7a551802feea0-Abstract.html

[40] Yue Zhao, Meng Li, Liangzhen Lai, Naveen Suda, Damon Civin, and Vikas Chandra. 2018. Federated Learning with Non-IID Data. *CoRR* abs/1806.00582 (2018). arXiv:1806.00582 http://arxiv.org/abs/1806.00582