FAVAS: Federated AVeraging with ASynchronous clients

Anonymous Author(s)
Affiliation
Address
email

Abstract

In this paper, we propose a novel centralized Asynchronous Federated Learning (FL) framework, FAVAS for training Deep Neural Networks (DNNs) in resource-constrained environments. Despite its popularity, "classical" federated learning faces the increasingly difficult task of scaling synchronous communication over large wireless networks. Moreover, clients typically have different computing resources and therefore computing speed, which can lead to a significant bias (in favor of "fast" clients) when the updates are asynchronous. Therefore, practical deployment of FL requires to handle users with strongly varying computing speed in communication/resource constrained setting. We provide convergence guarantees for FAVAS in a smooth, non-convex environment and carefully compare the obtained convergence guarantees with existing bounds, when they are available. Experimental results show that the FAVAS algorithm outperforms current methods on standard benchmarks.

1 Introduction

Federated learning, a promising approach for training models from networked agents, involves the collaborative aggregation of locally computed updates, such as parameters, under centralized orchestration (Konečný et al., 2015; McMahan et al., 2017; Kairouz et al., 2021). The primary motivation behind this approach is to maintain privacy, as local data is never shared between agents and the central server (Zhao et al., 2018; Horváth et al., 2022). However, communication of training information between edge devices and the server is still necessary. The central server aggregates the local models to update the global model, which is then sent back to the devices. Federated learning helps alleviate privacy concerns, and it distributes the computational load among networked agents. However, each agent must have more computational power than is required for inference, leading to a computational power bottleneck. This bottleneck is especially important when federated learning is used in heterogeneous, cross-device applications.

Most approaches to centralized federated learning (FL) rely on synchronous operations, as assumed in many studies (McMahan et al., 2017; Wang et al., 2021). At each global iteration, a copy of the current model is sent from the central server to a selected subset of agents. The agents then update their model parameters using their private data and send the model updates back to the server. The server aggregates these updates to create a new shared model, and this process is repeated until the shared model meets a desired criterion. However, device heterogeneity and communication bottlenecks (such as latency and bandwidth) can cause delays, message loss, and stragglers, and the agents selected in each round must wait for the slowest one before starting the next round of computation. This waiting time can be significant, especially since nodes may have different computation speeds.

To address this challenge, researchers have proposed several approaches that enable asynchronous communication, resulting in improved scalability of distributed/federated learning (Xie et al., 2019;

Chen et al., 2020, 2021; Xu et al., 2021). In this case, the central server and local agents typically operate with inconsistent versions of the shared model, and synchronization in lockstep is not required, even between participants in the same round. As a result, the server can start aggregating client updates as soon as they are available, reducing training time and improving scalability in practice and theory.

Contributions. Our work takes a step toward answering this question by introducing `FAVAS`, a centralized federated learning algorithm designed to accommodate clients with varying computing resources and support asynchronous communication.

- In this paper, we introduce a new algorithm called `FAVAS` that uses an unbiased aggregation scheme for centralized federated learning with asynchronous communication. Our algorithm does not assume that clients computed the same number of epochs while being contacted, and we give non-asymptotic complexity bounds for `FAVAS` in the smooth nonconvex setting. We emphasize that the dependence of the bounds on the total number of agents n is improved compared to Zakerinia et al. (2022) and does not depend on a maximum delay.
- Experimental results show that our approach consistently outperforms other asynchronous baselines on the challenging TinyImageNet dataset (Le and Yang, 2015).

Our proposed algorithm `FAVAS` is designed to allow clients to perform their local steps independently of the server's round structure, using a fully local, possibly outdated version of the model. Upon entering the computation, all clients are given a copy of the global model and perform at most $K \geq 1$ optimization steps based on their local data. The server randomly selects a group of s clients in each server round, which, upon receiving the server's request, submit an *unbiased* version of their progress. Although they may still be in the middle of the local optimization process, they send reweighted contributions so that fast and slow clients contribute equally. The central server then aggregates the models and sends selected clients a copy of the current model. The clients take this received server model as a new starting point for their next local iteration.

2 Related Works

Federated Averaging (FedAvg), also known as local SGD, is a widely used approach in federated learning. In this method, each client updates its local model using multiple steps of stochastic gradient descent (SGD) to optimize a local objective function. The local devices then submit their model updates to the central server for aggregation, and the server updates its own model parameters by averaging the client models before sending the updated server parameters to all clients. FedAvg has been shown to achieve high communication efficiency with infrequent synchronization, outperforming distributed large mini-batches SGD (Lin et al., 2019).

However, the use of multiple local epochs in FedAvg can cause each device to converge to the optima of its local objective rather than the global objective, a phenomenon known as client drift. This problem has been discussed in previous work; see (Karimireddy et al., 2020). Most of these studies have focused on synchronous federated learning methods, which have a similar update structure to FedAvg (Wang et al., 2020; Karimireddy et al., 2020; Qu et al., 2021; Makarenko et al., 2022; Mao et al., 2022; Tyurin and Richtárik, 2022). However, synchronous methods can be disadvantageous because they require all clients to wait when one or more clients suffer from high network delays or have more data, and require a longer training time. This results in idle time and wasted computing resources.

Moreover, as the number of nodes in a system increases, it becomes infeasible for the central server to perform synchronous rounds among all participants, and synchrony can degrade the performance of distributed learning. A simple approach to mitigate this problem is node sampling, e.g. Smith et al. (2017); Bonawitz et al. (2019), where the server only communicates with a subset of the nodes in a round. But if the number of stragglers is large, the overall training process still suffers from delays.

Synchronous FL methods are prone to stragglers. One important research direction is based on FedAsync (Xie et al., 2019) and subsequent works. The core idea is to update the global model immediately when the central server receives a local model. However, when staleness is important, performance is similar to FedAvg, so it is suboptimal in practice. ASO-Fed (Chen et al., 2020) proposes to overcome this problem and handles asynchronous FL with local streaming data by introducing memory-terms on the local client side. AsyncFedED (Wang et al., 2022) also relies on the FedAsync instantaneous update strategy and also proposes to dynamically adjust the learning rate and the number of local epochs to staleness. Only one local updated model is involved in FedAsync-like global model aggregations. As a result, a larger number of training epochs are required and the frequency of communication between the server and the workers increases greatly, resulting in massive bandwidth consumption. From a different perspective, QuAFL (Zakerinia et al., 2022) develops a concurrent algorithm that is closer to the FedAvg strategy. QuAFL incorporates both asynchronous and compressed communication with convergence guarantees. Each client must compute K local steps and can be interrupted by the central server at any time. The client updates its model with the (compressed) central version and its current private model. The central server randomly selects s clients and updates the model with the (compressed) received local progress (since last contact) and the previous central model. QuAFL works with old variants of the model at each step, which slows convergence. However, when time, rather than the number of server rounds, is taken into account, QuAFL can provide a speedup because the asynchronous framework does not suffer from delays caused by stragglers. A concurrent and asynchronous approach aggregates local updates before updating the global model: FedBuff (Nguyen et al., 2022) addresses asynchrony using a buffer on the server side. Clients perform local iterations, and the base station updates the global model only after Z different clients have completed and sent their local updates. The gradients computed on the client side may be stale. The main assumption is that the client computations completed at each step come from a uniform distribution across all clients. Fedbuff is asynchronous, but is also sensitive to stragglers (must wait until Z different clients have done all local updates). Similarly, Koloskova et al. (2022) focus on Asynchronous SGD, and provide guarantees depending on some τ_{max}. Similar to Nguyen et al. (2022) the algorithm is also impacted by stragglers, during the transitional regime at least. A recent work by Fraboni et al. (2023) extend the idea of Koloskova et al. (2022) by allowing multiple clients to contribute in one round. But this scheme also favors fast clients. Liu et al. (2021) does not run on buffers, but develops an Adaptive Asynchronous Federated Learning (AAFL) mechanism to deal with speed differences between local devices. Similar to FedBuff, in Liu et al. (2021)'s method, only a certain fraction of the locally updated models contribute to the global model update. Most convergence guarantees for asynchronous distributed methods depend on staleness or gradient delays (Nguyen et al., 2022; Toghani and Uribe, 2022; Koloskova et al., 2022). Only Mishchenko et al. (2022) analyzes the asynchronous stochastic gradient descent (SGD) independently of the delays in the gradients. However, in the heterogeneous (non-IID) setting, convergence is proved up to an additive term that depends on the dissimilarity limit between the gradients of the local and global objective functions.

3 Algorithm

We consider optimization problems in which the components of the objective function (i.e., the data for machine learning problems) are distributed over n clients, i.e.,

$$\min_{w\in\mathbb{R}^d} R(w);\;\; R(w) = \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}_{(x,y)\sim p^i_{\mathrm{data}}}[\ell(\mathrm{NN}(x,w),y)],$$

where d is the number of parameters (network weights and biases), n is the total number of clients, ℓ is the training loss (e.g., cross-entropy or quadratic loss), $\mathrm{NN}(x,w)$ is the DNN prediction function, p^i_{data} is the training distribution on client i. In FL, the distributions p^i_{data} are allowed to differ between clients (statistical heterogeneity).

Each client maintains three key values in its local memory: the local model w^i, a counter q^i, and the value of the initial model with which it started the iterations w^i_{init}. The counter q^i is incremented for each SGD step the client performs locally until it reaches K, at which point the client stops updating its local model and waits for the server request. Upon the request to the client i, the local model and counter q^i are reset. If a server request occurs before the K local steps are completed, the client simply pauses its current training process, reweights its gradient based on the number of local epochs (defined by E^i_{t+1}), and sends its current *reweighted* model to the server.

In Zakerinia et al. (2022), we identified the client update $w^i = \frac{1}{s+1}w_{t-1} + \frac{s}{s+1}w^i$ as a major shortcoming. When the number of sampled clients s is large enough, $\frac{s}{s+1}w^i$ dominates the update and basically no server term are taken into consideration. This leads to a significant client drift. As a

Algorithm 1: `FAVAS` over T iterations. In red are highlighted the differences with QuAFL.

Input : Number of steps T, LR η, Selection Size s, Maximum local steps K ;

`/* At the Central Server */`

1 **Initialize**
2 Initialize parameters w_0;
3 Server sends w_0 to all clients;
4 **end**
5 **for** $t = 1, \ldots, T$ **do**
6 Generate set $\mathcal{S}_t$ of s clients uniformly at random;
7 **for** *all clients* $i \in \mathcal{S}_t$ **do**
8 Server receives $w^i_{unbiased}$ from client i;
9 **end**
10 Update central server model $w_t \leftarrow \frac{1}{s+1} w_{t-1} + (\frac{1}{s+1} \sum_{i \in \mathcal{S}_t} w^i_{unbiased})$;
11 **for** *all clients* $i \in \mathcal{S}_t$ **do**
12 Server sends w_t to client i;
13 **end**
14 **end**

`/* At Client i */`

15 **Initialize**
16 Client receives w_0 and K from the Server;
17 **Local variables** $w^i = w_0, q^i = 0$;
18 **end**
19 **Loop**
20 Run `ClientLocalTraining()` concurrently;
21 **When** *Contacted by the Server* **do**
22 Interrupt `ClientLocalTraining()`;
23 Define α^i following (3) ;
24 Send $w^i_{unbiased} := w^i_{init} + \frac{1}{\alpha^i}(w^i - w^i_{init})$ to the server;
25 Receive w_t from the server;
26 Update $w^i_{init} \leftarrow w_t, w^i \leftarrow w_t, q^i \leftarrow 0$;
27 Restart `ClientLocalTraining()` from zero with updated variables;
28 **end**
29 **end**
30 **function** `ClientLocalTraining()`:
31 **while** $q^i < K$ **do**
32 Compute local stochastic gradient $\widetilde{g}^i$ at w^i;
33 Update local model $w^i \leftarrow w^i - \eta \widetilde{g}^i$;
34 Update local counter $q^i \leftarrow q^i + 1$;
35 **end**
36 `Wait()`;
37 **end function**

consequence, QuAFL does not perform well in the heterogeneous case (see Section 5). Second, one can note that the updates in QuAFL are biased in favor of fast clients. Indeed each client computes gradients at its own pace and can reach different numbers of epochs while being contacted by the central server. It is assumed that clients compute the *same* number of local epochs in the analysis from Zakerinia et al. (2022), but it is not the case in practice. As a consequence, we propose `FAVAS` to deal with asynchronous updates without favoring fast clients. A first improvement is to update local weight directly with the received central model. Details can be found in Algorithm 1. Another idea to tackle gradient unbiasedness is to reweight the contributions from each of the s selected clients: these can be done either by dividing by the (proper) number of locally computed epochs, or by the expected value of locally computed epochs. In practice, we define the reweight $\alpha^i = \mathbb{E}[E^i_{t+1} \wedge K]$, or $\alpha^i = \mathbf{P}(E^i_{t+1} > 0)(E^i_{t+1} \wedge K)$, where $\wedge$ stands for $\min$. We assume that the server performs a number of training epochs $T \geq 1$. At each time step $t \in \{1, \ldots, T\}$, the server has a model w_t. At initialization, the central server transmits identical parameters w_0 to all devices. At each time step t, the central server selects a subset $\mathcal{S}_t$ of s clients uniformly at random and requests their local models. Then, the requested clients submit their *reweighted* local models back to the server. When all requested models arrive at the server, the server model is updated based on a simple average (see Line 10). Finally, the server multicasts the updated server model to all clients in $\mathcal{S}_t$. In particular, all clients $i \notin \mathcal{S}_t$ continue to run their individual processes without interruption.

Remark 1. *In `FAVAS`'s setting, we assume that each client $i \in \{1, ..., n\}$ keeps a full-precision local model w^i. In order to reduce the computational cost induced by the training process, `FAVAS` can also be implemented with a quantization function Q. First, each client computes backpropagation with respect to its quantized weights $Q(w^i)$. That is, the stochastic gradients are unbiased estimates of $\nabla f_i\left(Q\left(w^i\right)\right)$. Moreover, the activations computed at forward propagation are quantized. Finally, the stochastic gradient obtained at backpropagation is quantized before the SGD update. In our supplementary experiments, we use the logarithmic unbiased quantization method of Chmiel et al. (2021).*

Table 1: How long one has to wait to reach an ϵ accuracy for non-convex functions. For simplicity, we ignore all constant terms. Each constant $C\_$ depends on client speeds and represents the unit of time one has to wait in between two consecutive server steps. L is the Lipschitz constant, and $F := (f(w_0) - f_*)$ is the initial conditions term. a_i, b are constants depending on client speeds statistics, and defined in Theorem 3.

Method	Units of time
FedAvg	$\left(\frac{FL\sigma^2+(1-\frac{s}{n})KG^2}{sK}\epsilon^{-2} + FL^{\frac{1}{2}}G\epsilon^{-\frac{3}{2}} + LFB^2\epsilon^{-1}\right) C_{FedAvg}$
FedBuff	$\left(FL(\sigma^2+G^2)\epsilon^{-2} + FL((\frac{\tau_{max}^2}{s^2}+1)(\sigma^2+nG^2))^{\frac{1}{2}}\epsilon^{-\frac{3}{2}} + FL\epsilon^{-1}\right) C_{FedBuff}$
AsyncSGD	$\left(FL(3\sigma^2+4G^2)\epsilon^{-2} + FLG(s\tau_{avg})^{\frac{1}{2}}\epsilon^{-\frac{3}{2}} + (s\tau_{max}F)^{\frac{1}{2}}\epsilon^{-1}\right) C_{AsyncSGD}$
QuAFL	$\frac{1}{E^2}FLK(\sigma^2+2KG^2)\epsilon^{-2} + \frac{n\sqrt{n}}{E\sqrt{Es}}FKL(\sigma^2+2KG^2)^{\frac{1}{2}}\epsilon^{-\frac{3}{2}} + \frac{1}{E\sqrt{s}}n\sqrt{n}FBK^2L\epsilon^{-1}$
`FAVAS`	$FL(\sigma^2\sum_i^n \frac{a_i}{n} + 8G^2b)\epsilon^{-2} + \frac{n}{s}FL^2(K^2\sigma^2 + L^2K^2G^2 + s^2\sigma^2\sum_i^n \frac{a_i}{n} + s^2G^2b)^{\frac{1}{2}}\epsilon^{-\frac{3}{2}} + nFB^2KLb\epsilon^{-1}$

4 Analysis

In this section we provide complexity bounds for `FAVAS` in a smooth nonconvex environment. We introduce an abstraction to model the stochastic optimization process and prove convergence guarantees for `FAVAS`.

Preliminaries. We abstract the optimization process to simplify the analysis. In the proposed algorithm, each client asynchronously computes its own local updates without taking into account the server time step t. Here in the analysis, we introduce a different, but statistically equivalent setting. At the beginning of each server timestep t, each client maintains a local model w_{t-1}^i. We then assume that all n clients *instantaneously* compute local steps from SGD. The update in local step q for a client i is given by:

$$\widetilde{h}_{t,q}^i = \widetilde{g}^i\left(w_{t-1}^i - \sum_{s=1}^{q-1}\eta\widetilde{h}_{t,s}^i\right),$$

where $\widetilde{g}^i$ represents the stochastic gradient that client i computes for the function f_i. We also define n independent random variables $E_t^1, \ldots, E_t^n$ in $\mathbb{N}$. Each random variable E_t^i models the number of local steps the client i could take before receiving the server request. We then introduce the following random variable: $\widetilde{h}_t^i = \sum_{q=1}^{E_t^i}\widetilde{h}_{t,q}^i$. Compared to Zakerinia et al. (2022), we do not assume that clients performed the same number of local epochs. Instead, we reweight the sum of the gradients by weights α^i, which can be either *stochastic* or *deterministic*:

$$\alpha^i = \begin{cases} \mathbf{P}(E_{t+1}^i > 0)(E_{t+1}^i \wedge K) & \textit{stochastic} \text{ version,} \\ \mathbb{E}[E_{t+1}^i \wedge K] & \textit{deterministic} \text{ version.} \end{cases} \tag{1}$$

And we can define the *unbiased* gradient estimator: $\check{h}_t^i = \frac{1}{\alpha^i}\sum_{q=1}^{E_t^i \wedge K}\widetilde{h}_{t,q}^i$.

Finally, a subset $\mathcal{S}_t$ of s clients is chosen uniformly at random. This subset corresponds to the clients that send their models to the server at time step t. In the current notation, each client $i \in \mathcal{S}_t$ sends the value $w_{t-1}^i - \eta\check{h}_t^i$ to the server. We emphasise that in our abstraction, all clients compute E_t^i local updates. However, only the clients in $\mathcal{S}_t$ send their updates to the server, and each client $i \in \mathcal{S}_t$ sends only the K first updates. As a result, we introduce the following update equations:

$$\begin{cases} w_t = \frac{1}{s+1}w_{t-1} + \frac{1}{s+1}\sum_{i\in\mathcal{S}_t}(w_{t-1}^i - \eta\frac{1}{\alpha^i}\sum_{s=1}^{E_t^i\wedge K}\widetilde{h}_{t,s}^i), \\ w_t^i = w_t, \quad \text{for } i \in \mathcal{S}_t, \\ w_t^i = w_{t-1}^i, \quad \text{for } i \notin \mathcal{S}_t. \end{cases}$$

Assumptions and notations.

A1. *Uniform Lower Bound: There exists $f_* \in \mathbb{R}$ such that $f(x) \geq f_*$ for all $x \in \mathbb{R}^d$.*

A2. *Smooth Gradients: For any client i, the gradient $\nabla f_i(x)$ is L-Lipschitz continuous for some $L > 0$, i.e. for all $x, y \in \mathbb{R}^d$: $\|\nabla f_i(x) - \nabla f_i(y)\| \leq L\|x - y\|$.*

A3. *Bounded Variance: For any client i, the variance of the stochastic gradients is bounded by some $\sigma^2 > 0$, i.e. for all $x \in \mathbb{R}^d$: $\mathbb{E}[\left\|\widetilde{g}^i(x) - \nabla f_i(x)\right\|^2] \leq \sigma^2$.*

A4. *Bounded Gradient Dissimilarity: There exist constants $G^2 \geq 0$ and $B^2 \geq 1$, such that for all $x \in \mathbb{R}^d$: $\sum_{i=1}^n \frac{\|\nabla f_i(x)\|^2}{n} \leq G^2 + B^2\|\nabla f(x)\|^2$.*

We define the notations required for the analysis. Consider a time step t, a client i, and a local step q. We define

$$\mu_t = \left(w_t + \sum_{i=1}^n w_t^i\right)/(n+1)$$

the average over all node models in the system at a given time t. The first step of the proof is to compute a preliminary upper bound on the divergence between the local models and their average. For this purpose, we introduce the Lyapunov function: $\Phi_t = \|w_t - \mu_t\|^2 + \sum_{i=1}^n \left\|w_t^i - \mu_t\right\|^2$.

Upper bounding the expected change in potential. A key result from our analysis is to upper bound the change (in expectation) of the aforementioned potential function Φ_t:

Lemma 2. *For any time step $t > 0$ we have:*

$$\mathbb{E}\left[\Phi_{t+1}\right] \leq (1-\kappa)\,\mathbb{E}\left[\Phi_t\right] + 3\frac{s^2}{n}\eta^2 \sum_{i=1}^n \mathbb{E}\left\|\check{h}_{t+1}^i\right\|^2, \quad \textit{with } \kappa = \frac{1}{n}\left(\frac{s(n-s)}{2(n+1)(s+1)}\right).$$

The intuition behind Lemma 2 is that the potential function Φ_t remains concentrated around its mean, apart from deviations induced by the local gradient steps. The full analysis involves many steps and we refer the reader to Appendix B for complete proofs. In particular, Lemmas 16 and 18 allow us to examine the scalar product between the expected node progress $\sum_{i=1}^n \check{h}_t^i$ and the true gradient evaluated on the mean model $\nabla f(\mu_t)$. The next theorem allows us to compute an upper-bound on the averaged norm-squared of the gradient, a standard quantity studied in nonconvex stochastic optimization.

Convergence results. The following statement shows that `FAVAS` algorithm converges towards a first-order stationary point, as T the number of global epochs grows.

Theorem 3. *Assume **A1** to **A4** and assume that the learning rate η satisfies $\eta \leq \frac{1}{20B^2bKLs}$. Then `FAVAS` converges at rate:*

$$\frac{1}{T}\sum_{t=0}^{T-1} \mathbb{E}\left\|\nabla f\left(\mu_t\right)\right\|^2 \leq \frac{2(n+1)F}{Ts\eta} + \frac{Ls}{n+1}\left(\frac{\sigma^2}{n}\sum_i^n a^i + 8G^2b\right)\eta + L^2s^2\left(\frac{720\sigma^2}{n}\sum_i^n a^i + 5600bG^2\right)\eta^2,$$

with $F := (f(\mu_0) - f_*)$, and

$$\begin{cases} a^i, b = \frac{1}{\mathbf{P}(E_{t+1}^i>0)^2}\left(\frac{\mathbf{P}(E_{t+1}^i>0)}{K^2} + \mathbb{E}\left[\frac{\mathbb{1}(E_{t+1}^i>0)}{E_{t+1}^i \wedge K}\right]\right), \max_i\left(\frac{1}{\mathbf{P}(E_{t+1}^i>0)}\right) \textbf{\textit{for }} \alpha^i = \mathbf{P}(E_{t+1}^i > 0)(E_{t+1}^i \wedge K), \\ a^i, b = \frac{1}{\mathbb{E}[E_{t+1}^i \wedge K]} + \frac{\mathbb{E}[(E_{+1}^i \wedge K)^2]}{K^2\mathbb{E}[E_{t+1}^i \wedge K]}, \max_i\left(\frac{\mathbb{E}[(E_{t+1}^i \wedge K)^2]}{\mathbb{E}[E_{t+1}^i \wedge K]}\right) \quad \textbf{\textit{for }} \alpha^i = \mathbb{E}[E_{t+1}^i \wedge K]. \end{cases}$$

Note that the previous convergence result refers to the average model μ_t. In practice, this does not pose much of a problem. After training is complete, the server can ask each client to submit its final model. It should be noted that each client communicates $\frac{sT}{n}$ times with the server during training. Therefore, an additional round of data exchange represents only a small increase in the total amount of data transmitted.

The bound in Theorem 3 contains 3 terms. The first term is standard for a general non-convex target and expresses how initialization affects convergence. The second and third terms depend on the statistical heterogeneity of the client distributions and the fluctuation of the minibatch gradients. Table 1 compares complexity bounds along with synchronous and asynchronous methods.One can note the importance of the ratio $\frac{s}{n}$. Compared to Nguyen et al. (2022) or Koloskova et al. (2022), `FAVAS` can potentially suffer from delayed updates when $\frac{s}{n} \ll 1$, but `FAVAS` does *not* favor fast clients at all. In practice, it is not a major shortcoming, and `FAVAS` is more robust to fast/slow clients distribution than FedBuff/AsyncSGD (see Figure 2). We emphasize both FedBuff and AsyncSGD rely on strong assumptions: neither the queuing process, nor the transitional regime are taken into account in their analysis. In practice, during the first iterations, only fast clients contribute. It induces a serious bias. Our experiments indicate that a huge amount of server iterations has to be accomplished to reach the stationary regime. Still, under this regime, slow clients are contributing with delayed information. Nguyen et al. (2022); Koloskova et al. (2022) propose to uniformly bound this delay by some quantity τ_{max}. We keep this notation while reporting complexity bounds in Table 1, but argue nothing guarantee τ_{max} is properly defined (i.e. finite). All analyses except that of Mishchenko et al. (2022) show that the number of updates required to achieve accuracy grows linearly with τ_{max}, which can be very adverse. Specifically, suppose we have two parallel workers - a fast machine that takes only 1 unit of time to compute a stochastic gradient, and a slow machine that takes 1000 units of time. If we use these two machines to implement FedBuff/AsyncSGD, the gradient delay of the slow machine will be one thousand, because in the 1 unit of time we wait for the slow machine, the fast machine will produce one thousand updates. As a result, the analysis based on τ_{max} deteriorates by a factor of 1000.

In the literature, guarantees are most often expressed as a function of server steps. In the asynchronous case, this is *inappropriate* because a single step can take very different amounts of time depending on the method. For example, with FedAvg or Scaffold (Karimireddy et al., 2020), one must wait for the slowest client for each individual server step. Therefore, we introduce in Table 1 constants $C\_$ that depend on the client speed and represent the unit of time to wait between two consecutive server steps. Finally, optimizing the value of the learning rate η with Lemma 12 yields the following:

Corollary 4. *Assume **A1** to **A4**. We can optimize the learning rate by Lemma 12 and* `FAVAS` *reaches an ϵ precision for a number of server steps T greater than (up to numerical constants):*

$$\frac{FL(\frac{\sigma^2}{n}\sum_i^n a^i + 8G^2 b)}{\epsilon^2} + (n+1)\left(\frac{FL^2(K^2\sigma^2 + L^2K^2G^2 + \frac{s^2\sigma^2}{n}\sum_i^n a^i + s^2G^2 b)^{\frac{1}{2}}}{s\epsilon^{\frac{3}{2}}} + \frac{FB^2KLb}{\epsilon}\right),$$

where $F = (f(\mu_0) - f_*)$, and (a^i, b) are defined in Theorem 3.

The second term in Corollary 4 is better than the one from the QuAFL analysis (n^3 of Zakerinia et al., 2022). Although this $(n+1)$ term can be suboptimal, note that it is only present at second order from ϵ and therefore becomes negligible when ϵ goes to 0 (Lu and De Sa, 2020; Zakerinia et al., 2022).

Remark 5. *Our analysis can be extended to the case of quantized neural networks. The derived complexity bounds also hold for the case when the quantization function Q is biased. We make only a weak assumption about Q (we assume that there is a constant r_d such that for any $x \in \mathbb{R}^d$ $\|Q(x) - x\|^2 \leq r_d$), which holds for standard quantization methods such as stochastic rounding and deterministic rounding. The only effect of quantization would be increased variance in the stochastic gradients. We need to add to the upper bound given in Theorem 3 an "error floor" of $12L^2 r_d$, which remains independent of the number of server epochs. For stochastic or deterministic rounding, $r_d = \Theta(d\frac{1}{2^{2b}})$, where b is the number of bits used. The error bound is the cost of using quantization as part of the optimization algorithm. Previous works with quantized models also include error bounds (Li et al., 2017; Li and Sa, 2019).*

5 Numerical Results

We test `FAVAS` on three image classification tasks: MNIST (Deng, 2012), CIFAR-10 (Krizhevsky et al., 2009), and TinyImageNet (Le and Yang, 2015). For the MNIST and CIFAR-10 datasets, two training sets are considered: an IID and a non-IIID split. In the first case, the training images are randomly distributed among the n clients. In the second case, each client takes two classes (out of the ten possible) without replacement. This process leads to heterogeneity among the clients.

The standard evaluation measure for FL is the number of server rounds of communication to achieve target accuracy. However, the time spent between two consecutive server steps can be very different for asynchronous and synchronous methods. Therefore, we compare different synchronous and asynchronous methods w.r.t. *total simulation time* (see below). We also measured the loss and accuracy of the model in terms of server steps and total local client steps (see Appendix C.3). In all experiments, we track the performance of each algorithm by evaluating the server model against an unseen validation dataset. We present the test accuracy and variance, defined as $\sum_{i=1}^n \|w_t^i - w_t\|^2$.

We decide to focus on non-uniform timing experiments as in Nguyen et al. (2022), and we base our simulation environment on QuAFL's code[1]. After simulating n clients, we randomly group them into fast or slow nodes. We assume that at each time step t (for the central server), a set of s clients is randomly selected without replacement. We assume that the clients have different computational speeds, and refer to Appendix C.2 for more details. We assume that only one-third of the clients are slow, unless otherwise noted. We compare `FAVAS` with the classic synchronous approach FedAvg (McMahan et al., 2017) and two newer asynchronous metods QuAFL (Zakerinia et al., 2022) and FedBuff (Nguyen et al., 2022). Details on implementing other methods can be found in Appendix C.1.

We use the standard data augmentations and normalizations for all methods. `FAVAS` is implemented in Pytorch, and experiments are performed on an NVIDIA Tesla-P100 GPU. Standard multiclass cross entropy loss is used for all experiments. All models are fine-tuned with $n = 100$ clients, $K = 20$ local epochs, and a batch of size 128. Following the guidelines of Nguyen et al. (2022), the buffer size in FedBuff is set to $Z = 10$. In FedAvg, the total simulated time depends on the maximum number of local steps K and the slowest client runtime, so it is proportional to the number of local steps and the number of global steps. In QuAFL and `FAVAS` on the other hand, each global step has a predefined duration that depends on the central server clock. Therefore, the global steps have similar durations and the total simulated time is the sum of the durations of the global steps. In FedBuff, a global step requires filling a buffer of size Z. Consequently, both the duration of a global step and the total simulated time depend on Z and on the proportion of slow clients (see Appendix C.2 for a detailed discussion).

We first report the accuracy of a shallow neural network trained on MNIST. The learning rate is set to 0.5 and the total simulated time is set to 5000. We also compare the accuracy of a Resnet20 (He et al., 2016) with the CIFAR-10 dataset (Krizhevsky et al., 2009), which consists of 50000 training images and 10000 test images (in 10 classes). For CIFAR-10, the learning rate is set to 0.005 and the total simulation time is set to 10000. In Figure 1, we show the test accuracy of `FAVAS` and competing

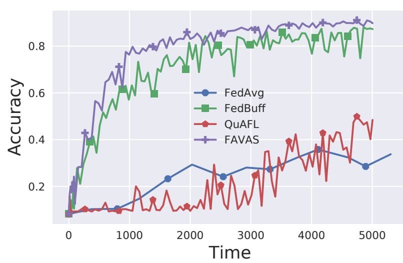


Figure 1: Test accuracy on the MNIST dataset with a non-IID split in between $n = 100$ total nodes, $s = 20$.

Table 2: Final accuracy on the test set (average and standard deviation over 10 random experiments) for the MNIST classification task. The last two columns correspond to Figures 1 and 2.

Methods	IID split	non-IID split ($\frac{2}{3}$ fast clients)	non-IID split ($\frac{1}{9}$ fast clients)
FedAvg	93.4 ± 0.3	38.7 ± 7.7	44.8 ± 6.9
QuAFL	92.3 ± 0.9	40.7 ± 6.7	45.5 ± 4.0
FedBuff	$\mathbf{96.0} \pm 0.1$	85.1 ± 3.2	67.3 ± 5.5
`FAVAS`	95.1 ± 0.1	$\mathbf{88.9} \pm 0.9$	$\mathbf{87.3} \pm 2.3$

methods on the MNIST dataset. We find that `FAVAS` and other asynchronous methods can offer a significant advantage over FedAvg when time is taken into account. However, QuAFL does not appear to be adapted to the non-IID environment. We identified client-side updating as a major shortcoming. While this is not severe when each client optimizes (almost) the same function, the QuAFL mechanism suffers from significant client drift when there is greater heterogeneity between clients. FedBuff is efficient when the number of stragglers is negligible compared to n. However, FedBuff is sensitive to the fraction of slow clients and may get stuck if the majority of clients are classified as slow and a few are classified as fast. In fact, fast clients will mainly feed the buffer, so the central updates will be heavily biased towards fast clients, and little information from slow clients will be considered. Figure 2 illustrates this phenomenon, where one-ninth of the clients are classified as fast. To provide a fair comparison, Table 2 gives the average performance of 10 random experiments with the different methods on the test set.

In Figure 3a, we report accuracy on a non-IID split of the CIFAR-10 dataset. FedBuff and `FAVAS` both perform better than other approaches, but FedBuff suffers from greater variance. We explain this limitation by the bias FedBuff provides in favor of fast clients. We also tested `FAVAS` on the TinyImageNet dataset (Le and Yang, 2015) with a ResNet18. TinyImageNet has 200 classes and each

[1]https://github.com/ShayanTalaei/QuAFL

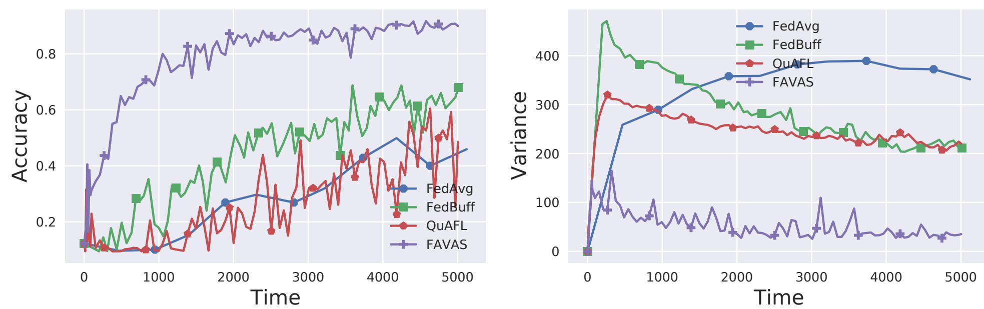
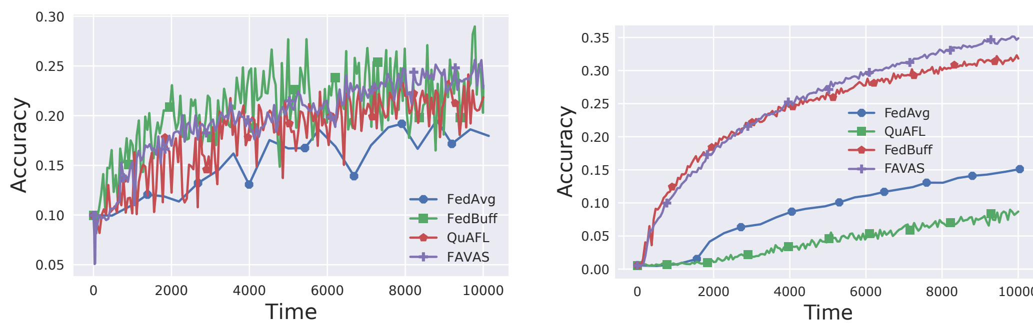


Figure 2: Test accuracy and variance on the MNIST dataset with a non-IID split between $n = 100$ total nodes. In this particular experiment, one-ninth of the clients are defined as fast.



(a) CIFAR-10 (non-IID)

(b) TinyImageNet (IID)

Figure 3: Test accuracy on CIFAR-10 and TinyImageNet datasets with $n = 100$ total nodes. Central server selects $s = 20$ clients at each round.

class has 500 (RGB) training images, 50 validation images and 50 test images. To train ResNet18, we follow the usual practices for training NNs: we resize the input images to 64×64 and then randomly flip them horizontally during training. During testing, we center-crop them to the appropriate size. The learning rate is set to 0.1 and the total simulated time is set to 10000. Figure 3b illustrates the performance of `FAVAS` in this experimental setup. While the partitioning of the training dataset follows an IID strategy, TinyImageNet provides enough diversity to challenge federated learning algorithms. Figure 3b shows that `FAVAS` scales much better on large image classification tasks than any of the methods we considered.

Remark 6. *We also evaluated the performance of `FAVAS` with and without quantization. We ran the code [2] from LUQ (Chmiel et al., 2021) and adapted it to our datasets and the FL framework. Even when the weights and activation functions are highly quantized, the results are close to their full precision counterpart (see Figure 7 in Appendix C).*

6 Conclusion

We have presented `FAVAS` the first (centralised) Federated Learning method of federated averaging that accounts for asynchrony in resource-constrained environments. We established complexity bounds under verifiable assumptions with explicit dependence on all relevant constants. Empirical evaluation shows that `FAVAS` is more efficient than synchronous and asynchronous state-of-the-art mechanisms in standard CNN training benchmarks for image classification.

[2]https://openreview.net/forum?id=clwYez4n8e8

References

Bonawitz, K., Eichner, H., Grieskamp, W., Huba, D., Ingerman, A., Ivanov, V., Kiddon, C., Konečnỳ, J., Mazzocchi, S., McMahan, B., et al. (2019). Towards federated learning at scale: System design. *Proceedings of Machine Learning and Systems*, 1:374–388.

Chen, Y., Ning, Y., Slawski, M., and Rangwala, H. (2020). Asynchronous online federated learning for edge devices with non-iid data. In *2020 IEEE International Conference on Big Data (Big Data)*, pages 15–24. IEEE.

Chen, Z., Liao, W., Hua, K., Lu, C., and Yu, W. (2021). Towards asynchronous federated learning for heterogeneous edge-powered internet of things. *Digital Communications and Networks*, 7(3):317–326.

Chmiel, B., Banner, R., Hoffer, E., Yaacov, H. B., and Soudry, D. (2021). Logarithmic unbiased quantization: Simple 4-bit training in deep learning. *arXiv preprint arXiv:2112.10769*.

Deng, L. (2012). The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE signal processing magazine*, 29(6):141–142.

Fraboni, Y., Vidal, R., Kameni, L., and Lorenzi, M. (2023). A general theory for federated optimization with asynchronous and heterogeneous clients updates. *Journal of Machine Learning Research*, 24(110):1–43.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Horváth, S., Sanjabi, M., Xiao, L., Richtárik, P., and Rabbat, M. (2022). Fedshuffle: Recipes for better use of local work in federated learning. *arXiv preprint arXiv:2204.13169*.

Kairouz, P., McMahan, H. B., Avent, B., Bellet, A., Bennis, M., Bhagoji, A. N., Bonawitz, K., Charles, Z., Cormode, G., Cummings, R., et al. (2021). Advances and open problems in federated learning. *Foundations and Trends® in Machine Learning*, 14(1–2):1–210.

Karimireddy, S. P., Kale, S., Mohri, M., Reddi, S., Stich, S., and Suresh, A. T. (2020). Scaffold: Stochastic controlled averaging for federated learning. In *International Conference on Machine Learning*, pages 5132–5143. PMLR.

Koloskova, A., Stich, S. U., and Jaggi, M. (2022). Sharper convergence guarantees for asynchronous sgd for distributed and federated learning. *arXiv preprint arXiv:2206.08307*.

Konečnỳ, J., McMahan, B., and Ramage, D. (2015). Federated optimization: Distributed optimization beyond the datacenter. *arXiv preprint arXiv:1511.03575*.

Krizhevsky, A., Hinton, G., et al. (2009). Learning multiple layers of features from tiny images.

Le, Y. and Yang, X. (2015). Tiny imagenet visual recognition challenge. *CS 231N*, 7(7):3.

Li, H., De, S., Xu, Z., Studer, C., Samet, H., and Goldstein, T. (2017). Training quantized nets: A deeper understanding. *Advances in Neural Information Processing Systems*, 30.

Li, Z. and Sa, C. D. (2019). Dimension-free bounds for low-precision training.

Lin, T., Stich, S. U., Patel, K. K., and Jaggi, M. (2019). Don't use large mini-batches, use local sgd. In *International Conference on Learning Representations*.

Liu, J., Xu, H., Wang, L., Xu, Y., Qian, C., Huang, J., and Huang, H. (2021). Adaptive asynchronous federated learning in resource-constrained edge computing. *IEEE Transactions on Mobile Computing*.

Lu, Y. and De Sa, C. (2020). Moniqua: Modulo quantized communication in decentralized sgd. In *International Conference on Machine Learning*, pages 6415–6425. PMLR.

Makarenko, M., Gasanov, E., Islamov, R., Sadiev, A., and Richtarik, P. (2022). Adaptive compression for communication-efficient distributed training. *arXiv preprint arXiv:2211.00188*.

Mao, Y., Zhao, Z., Yan, G., Liu, Y., Lan, T., Song, L., and Ding, W. (2022). Communication-efficient federated learning with adaptive quantization. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 13(4):1–26.

McMahan, B., Moore, E., Ramage, D., Hampson, S., and y Arcas, B. A. (2017). Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR.

Mishchenko, K., Bach, F., Even, M., and Woodworth, B. (2022). Asynchronous sgd beats minibatch sgd under arbitrary delays. *arXiv preprint arXiv:2206.07638*.

Nguyen, J., Malik, K., Zhan, H., Yousefpour, A., Rabbat, M., Malek, M., and Huba, D. (2022). Federated learning with buffered asynchronous aggregation. In *International Conference on Artificial Intelligence and Statistics*, pages 3581–3607. PMLR.

Qu, L., Song, S., and Tsui, C.-Y. (2021). Feddq: Communication-efficient federated learning with descending quantization. *arXiv preprint arXiv:2110.02291*.

Smith, V., Chiang, C.-K., Sanjabi, M., and Talwalkar, A. S. (2017). Federated multi-task learning. *Advances in neural information processing systems*, 30.

Toghani, M. T. and Uribe, C. A. (2022). Unbounded gradients in federated leaning with buffered asynchronous aggregation. *arXiv preprint arXiv:2210.01161*.

Tyurin, A. and Richtárik, P. (2022). Dasha: Distributed nonconvex optimization with communication compression, optimal oracle complexity, and no client synchronization. *arXiv preprint arXiv:2202.01268*.

Wang, J., Charles, Z., Xu, Z., Joshi, G., McMahan, H. B., Al-Shedivat, M., Andrew, G., Avestimehr, S., Daly, K., Data, D., et al. (2021). A field guide to federated optimization. *arXiv preprint arXiv:2107.06917*.

Wang, J., Liu, Q., Liang, H., Joshi, G., and Poor, H. V. (2020). Tackling the objective inconsistency problem in heterogeneous federated optimization. *Advances in neural information processing systems*, 33:7611–7623.

Wang, Q., Yang, Q., He, S., Shui, Z., and Chen, J. (2022). Asyncfeded: Asynchronous federated learning with euclidean distance based adaptive weight aggregation. *arXiv preprint arXiv:2205.13797*.

Xie, C., Koyejo, S., and Gupta, I. (2019). Asynchronous federated optimization. *arXiv preprint arXiv:1903.03934*.

Xu, C., Qu, Y., Xiang, Y., and Gao, L. (2021). Asynchronous federated learning on heterogeneous devices: A survey. *arXiv preprint arXiv:2109.04269*.

Zakerinia, H., Talaei, S., Nadiradze, G., and Alistarh, D. (2022). Quafl: Federated averaging can be both asynchronous and communication-efficient. *arXiv preprint arXiv:2206.10032*.

Zhao, Y., Li, M., Lai, L., Suda, N., Civin, D., and Chandra, V. (2018). Federated learning with non-iid data. *arXiv preprint arXiv:1806.00582*.