Federated Learning with Buffered Asynchronous Aggregation

John Nguyen **Kshitiz Malik** **Hongyuan Zhan** **Ashkan Yousefpour**
Mike Rabbat **Mani Malek** **Dzmitry Huba**
Meta AI

Abstract

Scalability and privacy are two critical concerns for cross-device federated learning (FL) systems. In this work, we identify that synchronous FL — synchronized aggregation of client updates in FL — cannot scale efficiently beyond a few hundred clients training in parallel. It leads to diminishing returns in model performance and training speed, analogous to large-batch training. On the other hand, asynchronous aggregation of client updates in FL (i.e., asynchronous FL) alleviates the scalability issue. However, aggregating individual client updates is incompatible with Secure Aggregation, which could result in an undesirable level of privacy for the system. To address these concerns, we propose a novel buffered asynchronous aggregation method, FedBuff, that is agnostic to the choice of optimizer, and combines the best properties of synchronous and asynchronous FL. We empirically demonstrate that FedBuff is 3.3× more efficient than synchronous FL and up to 2.5× more efficient than asynchronous FL, while being compatible with privacy-preserving technologies such as Secure Aggregation and differential privacy. We provide theoretical convergence guarantees in a smooth non-convex setting. Finally, we show that under differentially private training, FedBuff can outperform FedAvgM at low privacy settings and achieve the same utility for higher privacy settings.

1 Introduction

Federated Learning (FL) is a distributed learning paradigm that aims to train a shared model across participants while training data stays on the participant devices. In this work, we focus on cross-device FL where participants are edge devices (Kairouz et al. (2019)), and in particular, aim to address the following two challenges:

Challenge 1: Scalability. In large-scale cross-device FL settings, the number of clients can be in the millions, and only a small fraction of the client population may be available at any given time for training (Wang et al. (2021)). Additionally, client devices may have limited communication bandwidth and compute power. In these settings, an important parameter is *concurrency*: the number of clients training concurrently (i.e., *clients-per-round* or *cohort size*). There is a fundamental limitation when increasing concurrency in synchronous FL training: a diminishing return in the speed and quality of training. In this paper, we propose a novel buffered asynchronous aggregation optimization that makes it possible to train using significantly higher concurrency, improving the performance and efficiency of FL.

Challenge 2: Privacy. Inference attacks, methods trying to recover information from gradients, can expose sensitive information about the participating clients (Melis et al., 2019; Geiping et al., 2020). Given this privacy concern, secure aggregation (SecAgg) (Karl et al. (2020); Bonawitz et al. (2016)) and differential privacy (DP) (Kairouz et al. (2021); McMahan et al. (2018)) provide protection against inference attacks (Watson et al., 2021; Carlini et al., 2020). Using SecAgg, an honest-but-curious server cannot see the individual client updates, while DP can protect clients' data from observations based on the inputs and the output of the computation. With SecAgg, DP clipping and noise addition can be performed on server, providing a better privacy-utility trade-offs. For many real-world cross-device FL applications, compatibility with such privacy enhancing technologies is vital.

Our proposal: FedBuff. Motivated by these challenges, we propose and analyze FedBuff, a novel asynchronous federated optimization framework using buffered asynchronous aggregation. In FedBuff,

Proceedings of the 25th International Conference on Artificial Intelligence and Statistics (AISTATS) 2022, Valencia, Spain. PMLR: Volume 151. Copyright 2022 by the author(s).

clients train and communicate asynchronously with the server. Unlike other asynchronous methods, the server aggregates K client updates in a secure buffer before performing a server update. This secure buffer can be implemented by using Trusted Execution Environments (TEEs) (Karl et al., 2020; Mo et al., 2021).

Contributions. We highlight the main contribution:

• We propose FedBuff, a novel asynchronous federated optimization framework with buffered asynchronous aggregation to achieve scalability and privacy against the **honest-but-curious** threat model through secure aggregation and differential privacy.

• We provide a convergence analysis for FedBuff in the smooth non-convex setting. When clients take Q local SGD steps, FedBuff requires $\mathcal{O}\left(1/(\epsilon^2 Q)\right)$ server iterations to reach ϵ accuracy (Section 4).

• Empirically, we show that FedBuff is up to 3.8× more efficient than competing synchronous FL algorithms, even without penalizing synchronous FL algorithms for stragglers. We also demonstrate that FedBuff is up to 2.5× more efficient than the closest asynchronous FL algorithm in the literature, FedAsync (Xie et al., 2019). Our extensive empirical evaluation finds that $K = 10$ is a good setting across benchmarks and does not require tuning.

• To the best of our knowledge, we are the first to propose an asynchronous federated optimization framework that is compatible with SecAgg and global user-level DP. Under differentially private training, FedBuff can outperform both synchronous FL with amplified DP-SGD and DP-FTRL (differentially private Follow-the-Regularized-Leader) at low privacy settings, and be competitive for high privacy settings.

2 Background

Synchronous FL. Significant attention has been paid towards synchronous FL methods (SyncFL), as they are perhaps easier to analyze and implement. SyncFL methods are also better suited for privacy – training and aggregating updates over a large number of clients render most inference attacks ineffectual (Melis et al., 2019; Zhu and Han, 2020; Geiping et al., 2020; Lam et al., 2021). However, synchronous FL methods are prone to stragglers, proceeding at the pace of the slowest client. Bonawitz et al. (2019) proposed using over-selection to tap 30% more clients than the target cohort size and wait for the fastest replies to overcome this issue. However, over-selection comes at the cost of wasting clients' resources and introduces selection bias. We study these problems in Appendix C.2 and C.3.

In SyncFL optimization, FedAvg, a generalization of local SGD, has been shown to work well empirically (McMahan et al., 2016). FedProx (Li et al., 2018) improves upon FedAvg by adding a proximal term μ to the local SGD optimizer. FedAvgM (Hsu et al., 2019) further improves convergence by adding server-side momentum. Adaptive methods such as FedAdam (Reddi et al., 2020) are effective in cross-device FL settings and have comparable performance to FedAvgM. These optimizers often focus on heterogeneity, asymptotic convergence, and communication efficiency in *low concurrency* settings. In this paper, we focus on *high concurrency* settings. In these settings, SyncFL is not scalable and is inefficient. Typically, the optimal server learning rate increases with concurrency; aggregating over more users has a variance-reducing effect, enabling the server to take larger steps. Consequently, higher concurrency reduces the number of rounds needed to reach a target accuracy because of a larger server learning rate. However, to have stable, convergent training dynamics, the server learning rate cannot be increased indefinitely; eventually it saturates, resulting in a sublinear speed-up similar to in large-batch training (Goyal et al. (2017); Ott et al. (2018); You et al. (2019, 2018, 2017); Shallue et al. (2018)). As a result, SyncFL systems cannot accelerate training through parallelism beyond a few hundred clients and exhibit decreasing efficiency with increasing concurrency (Figure 1).

Asynchronous FL. Asynchronous FL methods are a good match for cross-device FL settings, where clients have different compute power and intermittent availability (Wang et al., 2021). Most asynchronous FL (AsyncFL) works, such as the works (Xie et al., 2019; van Dijk et al., 2020; Chai et al., 2020; Chen et al., 2019; Wu et al., 2020; Li et al., 2021) have been focused on solving the straggler problem by designing asynchronous FL algorithms. However, these proposals include aspects that make them impractical for real-world at-scale FL settings. For instance, Chai et al. (2020); Li et al. (2021) profile client speed, Chen et al. (2019) broadcast the model updates to all clients, Xie et al. (2019) update the server model on every client, placing a significant burden on the clients and server, and van Dijk et al. (2020) assume all clients have the same speed.

In *"fully"* AsyncFL methods (e.g., Xie et al. (2019)), every client update results in a server model update. This has implications for privacy and scalability. Considering privacy, when every client update forces a server update, SecAgg cannot be used; secure aggregation's benefit is in hiding individual updates by combining them in an aggregate. Additionally, providing user-level DP in AsyncFL is only feasible with local differential privacy (LDP), where the client clips the model update and adds noise locally to it before sending it to

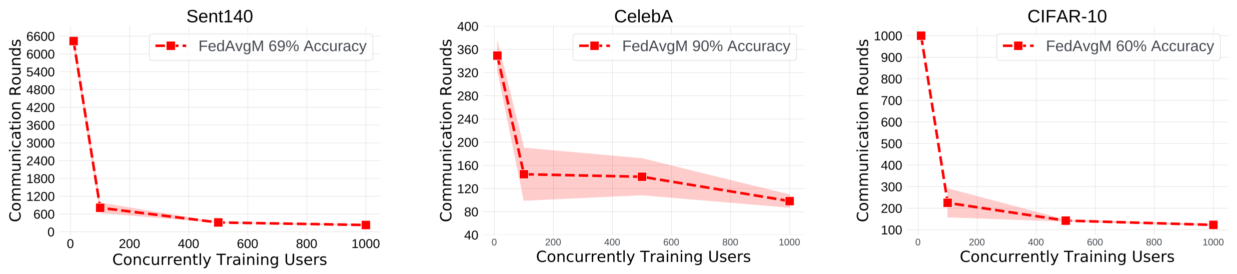


Figure 1: The number of communication rounds to reach a target accuracy with varying levels of concurrency. SyncFL algorithms such as FedAvgM (Hsu et al., 2019) shows diminishing returns from increasing concurrency beyond 100. For example, increasing concurrency by 10x (100 –> 1000) decreases the number of communication rounds by less than 2x. This is analogous to large-batch training, where increasing the batch size eventually gives diminishing returns.

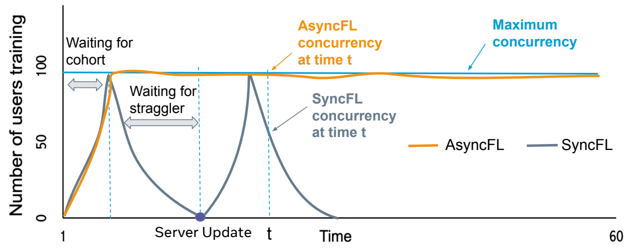


Figure 2: Training progress for asynchronous and synchronous FL, and the associated delays. Synchronous FL proceeds in *rounds*. The number of active clients increases at the beginning of a round as clients join the cohort, and it falls gradually towards the end of the round due to stragglers. In asynchronous FL, the number of active clients stays relatively constant over time; as clients finish training and upload their results, other clients take their place.

the server. LDP for high dimensional data has been criticized for poor privacy-utility trade-off (Erlingsson et al. (2020); Bittau et al. (2017)).

Secure Aggregation. SecAgg is a privacy enhancing technology based on cryptographic primitives (Bonawitz et al., 2016; Bell et al., 2020; So et al., 2021) or hardware-based Trusted Execution Environment (TEE) (Karl et al., 2020). SecAgg enhances privacy by obfuscating a client's update with many other clients' updates, protecting against the honest-but-curious server threat model (Bonawitz et al., 2016). FedBuff is compatible with SecAgg.

Differential Privacy. DP (Dwork et al., 2014) provides a rigorous formulation of the release of information derived from private data. In the context of machine learning, differentially private training (Abadi et al., 2016) limits what can be learned about the original training data.

Definition 1. *A randomized mechanism M: $U \mapsto R$ satisfies (ϵ, δ)-DP if for all adjacent datasets $D, D' \in U$ and for any subset of outputs $S \subseteq R$, the following holds:*

$$Pr[M(D) \in S] \leq e^{\epsilon} \cdot Pr[M(D') \in S] + \delta.$$

The definition of adjacent datasets D, D' is domain and application dependent. In the context of cross-device FL, we consider D, D' to be two datasets of training examples, where each example is associated with a client. Then, D and D' are adjacent if D' can be formed by adding or removing all of the examples associated with a single client from D (i.e., *user-level privacy* (McMahan et al., 2018)). In this paper, we consider the *global DP* (GDP) setting, where a trusted server collects, clips, aggregates the client updates. The server then adds noise to the aggregated updates. Compared to LDP, GDP provides a better privacy-utility trade-off for high dimensional data. This setting of DP relies on a SecAgg on the server, as server is responsible for operation of DP.

3 FedBuff: Federated Learning with Buffered Asynchronous Aggregation

We consider the following optimization problem:

$$\min_{w \in \mathbb{R}^d} f(w) := \frac{1}{m} \sum_{i=1}^{m} p_i F_i(w) \tag{1}$$

where m is the total number of clients and the function F_i measures the loss of a model with parameters w on the ith client's data, and $p_i > 0$ weighs the importance of the data from client i. The goal is to find a model that fits all clients' data well on (weighted) average. In FL, F_i is only accessible by client i. For simplicity, in this paper, we focus on the unweighted setting, $p_i = 1$ for all i, although the analysis is easily extendable to the more general case with non-uniform weights.

SyncFL methods need to aggregate and synchronize clients after each round. Hence, concurrency in SyncFL is equal to the number of clients who participate in a given round. In asynchronous methods, concurrency is the number of clients training at a given point in time (Figure 2). In FedBuff (Algorithm 1), clients enter and finish local training asynchronously. However, the server model is not updated immediately upon receiving every client update. Instead, client updates are stored in a *buffer*. A server update only takes place once K client updates are in the buffer, where K is the size of the buffer and is a tunable parameter. However, we find that $K = 10$ is a good choice and does not require tuning. The buffer can be implemented by using a Trusted Execution Environment (TEE) (Karl et al., 2020; Mo et al., 2021). Note that K is independent of concurrency — the extra degree of freedom introduced by the buffer allows the server to choose the model update frequency instead of coupling concurrency with the server model update as in SyncFL. The extra degree of freedom allows FedBuff to achieve data efficiency at high concurrency while being compatible with secure aggregation and DP.

FedBuff is compatible with SecAgg, because with $K > 1$ updates in the buffer, SecAgg provides its promise by hiding individual updates in the aggregate. Since FedBuff supports SecAgg, it can be easily extended to provide global DP. In asynchronous FL settings, the server has no control over which clients participate in a particular model update and client availability is dynamic. For such settings, privacy amplification by sampling is not feasible. DP-FTRL (Kairouz et al., 2021) has emerged as a suitable solution to address this issue. FedBuff with DP-FTRL is straightforward and we show in Algorithm 1 how one can extend FedBuff to provide global DP. The three functions, *InitializeTree*, *AddToTree*, and *GetSum* in Algorithm 1 correspond to those of the DP-FTRL algorithm. We defer to Section B.1 in (Kairouz et al., 2021) for more in-depth descriptions of the functions.

Algorithm 1 `FedBuff-server`

```
Input: server learning rate $\eta_g$, client learning rate $\eta_\ell$, client SGD steps $Q$, buffer size $K$
    $n$ dataset size, noise scale $\sigma^2$, clip norm $L$ (DP)
Output: FL-trained global model
1:  $\mathcal{T} \leftarrow InitializeTree(n, \sigma^2, L)$ (DP)
2:  repeat
3:      $c \leftarrow$ sample available clients                 ▷ async
4:      run FedBuff-client$(w^t, \eta_\ell, Q)$ on $c$          ▷ async
5:      if receive client update then
6:          $\Delta_i \leftarrow$ received update from client $i$
7:          $\Delta_i \leftarrow Clip(\Delta_i, L)$ (DP)            ▷ in TEE
8:          $\overline{\Delta}^t \leftarrow \overline{\Delta}^t + \Delta_i$                ▷ in TEE
9:          $k \leftarrow k + 1$
10:     if $k == K$ then
11:         $\overline{\Delta}^t \leftarrow \frac{\overline{\Delta}^t}{K}$
12:         $\mathcal{T} \leftarrow \mathrm{AddToTree}(\mathcal{T}, t, \overline{\Delta}^t)$ (DP) ▷ in TEE
13:         $\overline{\Delta}^t \leftarrow \overline{\Delta}^t + \mathrm{GetSum}(\mathcal{T}, t)$ (DP) ▷ in TEE
14:         $w^{t+1} \leftarrow w^t - \eta_g \overline{\Delta}^t$
15:         $\overline{\Delta}^t \leftarrow 0, k \leftarrow 0, t \leftarrow t + 1$       ▷ reset buffer
16: until Convergence
```

Algorithm 2 `FedBuff-client`

```
Input: server model $w$, client learning rate $\eta_\ell$, number of client SGD steps $Q$
Output: client update $\Delta$
1: $y_0 \leftarrow w$
2: for $q = 1 : Q$ do
3:     $y_q \leftarrow y_{q-1} - \eta_\ell g_q(y_{q-1})$
4: $\Delta \leftarrow y_0 - y_q$
5: Send $\Delta$ to server
```

4 Convergence Analysis

In this section, we provide a convergence guarantee for FedBuff in the smooth, non-convex setting. Most previous works analyze synchronous federated learning methods, such as the works in (Lin et al., 2018; Li et al., 2018; Reddi et al., 2020; Li et al., 2020; Stich, 2019; Yu et al., 2019b; Li et al., 2019; Haddadpour and Mahdavi, 2019; Karimireddy et al., 2020). In contrast, in FedBuff, clients train asynchronously, and the client updates are first aggregated in a buffer before producing a server model update. Hence, it is essential to understand the relationship between client computation and server communication under asynchrony with buffered aggregation.

Notation. We use the following notation throughout: $[m]$ represents the set of all client indices, $\nabla F_i(w)$ denotes the gradient with respect to the loss on client i's data, f^* denotes the minimum of $f(w)$, $g_i(w; \zeta_i)$ denotes the stochastic gradient on client i, K is the buffer size for aggregation before producing each server update, and Q denotes the number of local steps taken by each client. We make the following assumptions throughout.

Assumption 1. *(Unbiasedness of client stochastic gradient)* $\mathbb{E}_{\zeta_i}[g_i(w; \zeta_i)] = \nabla F_i(w)$.

Assumption 2. *(Bounded local and global variance) for all clients $i \in [m]$,*

$$\mathbb{E}_{\zeta_i|i}[\|g_i(w; \zeta_i) - \nabla F_i(w)\|^2] \le \sigma_\ell^2,$$

and

$$\frac{1}{m}\sum_{i=1}^{m}\|\nabla F_i(w)-\nabla f(w)\|^2 \le \sigma_g^2.$$

Assumption 3. *(Bounded gradient)* $\|\nabla F_i\|^2 \le G$ *for all* $i \in [m]$.

Assumption 4. *(Lipschitz gradient) for all client* $i \in [m]$*, the gradient is* L*-smooth,*

$$\|\nabla F_i(w)-\nabla F_i(w')\|^2 \le L\,\|w-w'\|^2.$$

Assumptions 1–4 are commonly made in analyzing federated learning algorithms (Reddi et al. (2020); Li et al. (2020); Stich (2019); Yu et al. (2019b)). We make an additional assumption on the staleness under asynchrony.

Assumption 5. *(Bounded Staleness when* $K = 1$*) For all clients* $i \in [m]$ *and for each server step* t*, the staleness* $\tau_i(t)$ *between the model version in which* `FedBuff-client` *uses to start local training, and the model version in which* Δ^i *is used to modify the global model is not larger than* $\tau_{\max,1}$ *when* $K = 1$.

Remark. More generally, the staleness upper-bound depends on the buffer size K. When the buffer size increases, the server iterates are updated less frequently, hence reducing the number of server steps in between the initialization of client training and when the client updates are used for modifying the server model. Specifically, if Assumption 5 holds, then for any execution of FedBuff with $K > 1$, the maximum delay $\tau_{\max,K}$ is at most $\lceil \tau_{\max,1}/K \rceil$; see Appendix A.

Theorem 1. *Let* $\eta_\ell^{(q)}$ *be the local learning rate of client SGD in the q-th step, and define* $\alpha(Q) := \sum_{q=0}^{Q-1}\eta_\ell^{(q)}$, $\beta(Q) := \sum_{q=0}^{Q-1}(\eta_\ell^{(q)})^2$*. Choosing* $\eta_g\eta_\ell^{(q)}Q \le \frac{1}{L}$ *for all local steps* $q = 0, \cdots, Q-1$*, the global model iterates in Algorithm 1 achieves the following ergodic convergence rate*

$$\frac{1}{T}\sum_{t=0}^{T-1}\|\nabla f(w^t)\|^2 \le \frac{2\Big(f(w^0)-f^*\Big)}{\eta_g\alpha(Q)T} + \frac{L}{2}\frac{\eta_g\beta(Q)}{\alpha(Q)}\sigma_\ell^2 + 3L^2Q\beta(Q)\Big(\eta_g^2\tau_{\max,K}^2+1\Big)\Big(\sigma_\ell^2+\sigma_g^2+G\Big) \quad (2)$$

The proof of Theorem 1 is provided in Appendix D.

Corollary 1. *Choosing constant local learning rate* η_ℓ *and* η_g *such that* $\eta_g\eta_\ell Q \le \frac{1}{L}$*, the global model iterates in FedBuff (Algorithm 1) are bounded by*

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left[\|\nabla f(w^t)\|^2\right] \le \frac{2F^*}{\eta_g\eta_\ell QT} + \frac{L}{2}\eta_g\eta_\ell\sigma_\ell^2 + 3L^2Q^2\eta_\ell^2\Big(\eta_g^2\tau_{\max,K}^2+1\Big)\sigma^2, \quad (3)$$

where $F^* := f(w^0) - f^*$ *and* $\sigma^2 := \sigma_\ell^2 + \sigma_g^2 + G$*. Further, choosing* $\eta_\ell = \mathcal{O}\left(1/\left(K\sqrt{TQ}\right)\right)$, $\eta_g = \mathcal{O}(K)$*, for all* η_g, η_ℓ *satisfying* $\eta_g\eta_\ell Q \le \frac{1}{L}$ *and sufficiently large* T*, we have*

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left[\|\nabla f(w^t)\|^2\right] \le \mathcal{O}\left(\frac{F^*}{\sqrt{TQ}}\right) + \mathcal{O}\left(\frac{\sigma_\ell^2}{\sqrt{TQ}}\right) + \mathcal{O}\left(\frac{Q\sigma^2}{TK^2}\right) + \mathcal{O}\left(\frac{Q\sigma^2\tau_{\max,1}^2}{TK^2}\right), \quad (4)$$

where we use the relation $\tau_{\max,K} \le \lceil \tau_{\max,1}/K \rceil$.

Corollary 1 yields several insights:

Worst-case iteration complexity. Theorem 1 bounds the ergodic norm-squared of the gradient, a standard quantity studied in non-convex stochastic optimization. If this becomes small as T grows, then it must be that the norm-squared of the gradient at later iterations is vanishing, implying the algorithm is converging towards a first-order stationary point. The bound in equation (4) contains three terms. The first is standard, expressing how the initialization impacts convergence, and it decreases at a rate of $\mathcal{O}(1/T)$ as is standard for SGD. The second two terms depend on different sources of variance, due to heterogeneity of functions at different clients (σ_g^2), stochasticity of gradients ($\sigma_\ell^2 + G$), and stale gradients due to delays in asynchronous execution ($\tau_{\max,K}$). Corollary 1 is derived from Theorem 1 under a specific choice of constant learning rate, and yields interpretation of trade-offs between the convergence of loss, local and global variance, effect of client drift due to local steps, effect of staleness and effect of buffer size. We summarize the these trade-offs next.

Total communication cost. Since each server step in FedBuff involves K client trips between the server and the clients, the total communication cost is $\mathcal{O}\left(K/\epsilon^2Q\right) + \mathcal{O}\left(Q\sigma^2/K\epsilon\right) + \mathcal{O}\left(KQ\sigma^2\tau_{\max,K}^2/\epsilon\right)$ in order to achieve $\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left[\|\nabla f(w^t)\|^2\right] \le \epsilon$. This suggests a trade-off in the communication cost introduced by the effect of the buffer. We empirically investigate different values of K in Section 6 and observe this tradeoff in Table 2.

Relation between communication and local computation. Note that in equation (4), increasing the number of local steps Q improves the first term related to F^* and the second term related to the local variance σ_ℓ^2, but increases the third and fourth term. The first term with constant F^* characterizes the distance to optimal loss. Hence, increasing local computation Q reduces the loss faster, but it also leads to more drift, enlarging the effect of the local and global variance sum σ^2 and the impact of the worst-case staleness $\tau_{\max,K}$.

Effect of staleness. The effect of staleness between the initialization of `FedBuff-client` and the server update dissipates at the rate of $\mathcal{O}(1/T)$ according to the fourth term in equation (4). In addition, the maximum staleness $\tau_{\max,K}$ reduces as the buffer size K grows. (see Appendix A)

5 Practical Improvements

Staleness scaling. To control the effect of staleness $\tau_i(t)$ in client i's contribution to the t-th server update, we down-weight stale updates using the following function: $s(\tau_i(t)) := 1/(1+\tau_i(t))^{0.5}$, similar to (Xie et al. (2019)).

Learning rate normalization. In practical FL implementations, each client is typically asked to perform a fixed number of *epochs* over their local training data, rather than a fixed number Q of steps, using a server-prescribed batch size B which is the same for all clients. Because different clients have different amounts of data, some clients may only have a fraction of a batch. Previous work has suggested that increasing batch size and learning rate are complementary (Goyal et al., 2017; Smith et al., 2017; Jastrzebski et al., 2017). When a client performs a local update with a batch size smaller than B, we have it linearly scale the learning rate used for that local step; i.e., $\eta_{\text{LRN}} := \eta_\ell \cdot n_{i,q}^t / B$, where $n_{i,q}^t \leq B$ is actual batch size used for the step. We find that this small change can improve FedBuff. A theoretical justification is provided in Appendix C.6.

6 Experiments

In this section, we compare the efficiency and scalability of FedBuff with other synchronous and asynchronous FL methods from the literature via simulation. We wish to understand how FedBuff behaves under different values of K, its scalability, and data efficiency.

Evaluation metrics. The standard evaluation metric for FL is the number of communication rounds to reach a target accuracy. However, asynchronous and synchronous methods do not have the same notion of rounds. For this reason, we compare different synchronous and asynchronous methods by the *number of client trips* needed to reach a target accuracy. One client trip corresponds to one client round-trip communication. A client trip involves a client pulling the latest model from the server (download communication), performing one epoch of training on the local dataset (computation), then communicating the model update to the server (upload communication). Since the number of client trips measures both communication and computation costs, we use this as a proxy for wall-clock training time. We show wall-clock time simulation with stragglers in Appendix C.2.

Datasets, models, and tasks. We run experiments on three datasets: CelebA (Liu et al. (2015)), Sent140 (Go et al. (2009)), and CIFAR-10 (Krizhevsky et al. (2009)). Sent140 is a text classification dataset (binary sentiment analysis), whereas CelebA and CIFAR-10 are image classification datasets (multi-class classification). For Sent140 and CelebA, we use the natural non-iid client partitions , and similar models from LEAF benchmark (Caldas et al. (2018)). For Sent140, we train an LSTM classifier over 660,120 clients, where each Twitter account corresponds to a client. For CelebA, we train the same convolutional neural network classifier as LEAF over 9,343 clients, but with batch normalization layers replaced by group normalization layers (Wu and He (2018)) as suggested in (Hsieh et al. (2020)). For CIFAR-10, we generate 5000 non-iid clients using a Dirichlet distribution with parameter 0.1, the same approach as in (Hsu et al. (2019)). More details about datasets, models, and tasks are provided in Appendix B.1.

Experimental setup. We implement all algorithms in PyTorch (Paszke et al. (2017)). We repeat each experiment with three different seeds and report the average. For asynchronous FL methods, we assume that clients arrive at a constant rate. We sample the delay distribution, the time delay between a client's download and upload operation, from a half-normal distribution. We choose this distribution because it best matches the delay distribution observed in our production FL system (See Appendix C.1). We also report results with two other delay distributions (uniform and exponential) in Appendix C.1. We find that FedBuff's performance improvements are consistent across different delay distributions.

Baselines. We compare FedBuff with three SyncFL baselines, namely FedAvg (McMahan et al. (2016)), FedProx (Li et al. (2018)), FedAvgM (Hsu et al. (2019)), and one AsyncFL baseline, FedAsync (Xie et al. (2019)). For more details about the algorithms used and the experimental setup, see Appendix B.2.

Hyperparameters. For all algorithms, we run hyperparameter sweeps to tune client and server learning rates η_ℓ and η_g, server momentum β, and the proxi-

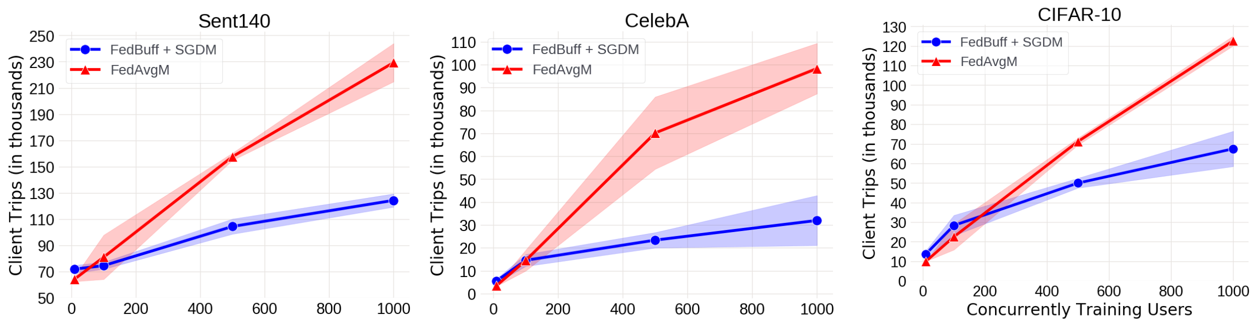


Figure 3: Number of client trips to reach target validation accuracy for FedBuff + SGD with momentum at the server and FedAvgM. At low concurrency, FedBuff and FedAvgM perform similarly. However, as concurrency increases, FedBuff outperforms FedAvgM by increasingly larger amounts. In contrast to FedAvgM, FedBuff's data-efficiency and communication-efficiency degrade less with concurrency.

Table 1: **Average (speedup)** number of client trips to reach target validation accuracy on CelebA and Sent140 (lower is better. Units = 1000 updates). We set concurrency = 1000 for all methods, $K = 10$ for FedBuff and ran all methods for 600k client trips. "> 600" indicates the target accuracy was not reached.

Dataset	Accuracy	FedBuff	FedAsync	FedAvgM	FedAvg	FedProx
CelebA	90%	31.9	37.1 (1.2×)	104 (3.3×)	231 (8.5×)	228 (8.4×)
Sent140	69%	124.7	308.9 (2.5×)	216 (1.7×)	> 600	> 600
CIFAR-10	60%	67.5	73.3 (1.1×)	122.7 (1.8×)	386.7 (5.7×)	292.7 (4.3×)

mal term μ for FedProx. We set $\beta = 0$ for FedAvg. Each client update entails running one local epoch with batch size $B = 32$, rather than a fixed number of local steps. See Appendix B.3 for additional details on hyperparameter tuning.

Concurrency and K. In at-scale cross-device FL, only a small fraction of all clients participate in training at any point in time. As discussed earlier, concurrency — the maximum number of clients that train in parallel — significantly impacts the performance of FL algorithms. For a fair comparison between synchronous and asynchronous algorithms, we keep concurrency the same across all configurations. Recall the example in Figure 2 where concurrency=100. For synchronous algorithms, this implies that 100 clients are training and contributing in each round. For asynchronous algorithms, this implies that 100 clients can train concurrently, and we can still vary the buffer size K, which will control how frequently updates occur.

6.1 Results

Comparison of Methods. Table 1 shows the number of client trips needed to converge to the target accuracy on Sent140, CelebA and CIFAR-10 for each method considered. In Table 2, we show results with other values of K and present the learning curves in Appendix C.7. Compared to the best synchronous method in the experiments (FedAvgM), FedBuff converges to target accuracy 1.7-3.3× more efficient. Compared to FedAsync, FedBuff converges to target accuracy 1.1-2.5× more efficient.

Scalablility of FedBuff. Figure 3 shows that FedBuff scales much better to larger values of concurrency than FedAvgM. FedBuff with $K = 10$ scales better because it updates the server model more frequently than FedAvgM in high concurrency. When concurrency is 10, both FedAvgM and FedBuff update the server model after every 10 client updates. However, when concurrency is 1000, FedBuff with $K = 10$ updates the server model after every 10 client updates, while FedAvgM updates the server model after 1000 client updates. One might argue that FedAvgM should run at lower concurrency, e.g. 10. However, that leads to longer wall-clock training time because less parallelism is exploited. We discuss this problem in Appendix C.2. For synchronous FL methods, larger concurrency reduces training time but is also less efficient. On the other hand, taking server model steps more frequently is not free; FedBuff has to deal with staleness as a consequence. Our empirical results show, the benefits from frequent advancing the server model outweigh the cost of staleness in client model updates.

Choice of K. Table 2 presents the number of client trips to reach validation accuracy for different values of K, with fixed concurrency. We find that K=10 is a good setting across benchmarks. We analyze FedBuff

Table 2: **Average $\pm$ standard deviation** number of client trips to reach validation accuracy on CelebA (90%), Sent140 (69%) and CIFAR-10 (60%) (lower is better, Units = 1000 updates) in FedBuff. We set concurrency = 1000 for all methods.

Dataset	K	Client trips
CelebA	1	32.4 ± 2.0
	10	31.9 ± 8.9
	100	71.6 ± 6.8
Sent140	1	190.0 ± 11.9
	10	124.7 ± 25.8
	100	178.2 ± 13.1
CIFAR-10	1	76.7 ± 9.6
	10	67.5 ± 7.4
	100	102.5 ± 2.0

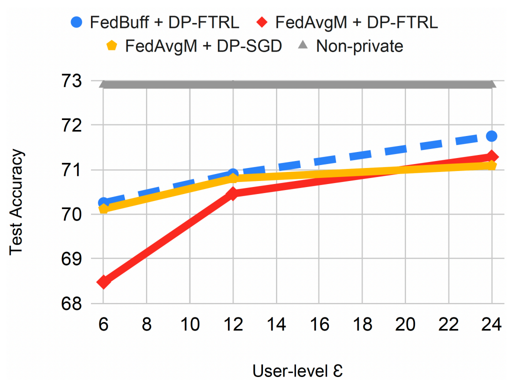


Figure 4: Accuracy on Sent140 under different levels of privacy ($\delta = 1e^{-7}$) for FedBuff with DP-FTRL versus FedAvgM with DP-FTRL and FedAvgM with amplified DP-SGD. For all methods, we use momentum at the server and fix the communication cost at 600 thousands. For FedBuff, we use $K = 10$. As for FedAvgM we use clients-per-round = 1000.

with even larger values of K in Table 6, and show the training curves of FedBuff and other algorithms in Appendix C.7.

FedBuff with Differential Privacy. To evaluate the privacy-utility trade-off of FedBuff, we present the final test accuracy of FedBuff against synchronous baselines after 600 thousands client trips, one pass over the dataset. Figure 4 illustrates that FedBuff can outperform both FedAvgM with amplified DP-SGD and FedAvgM with DP-FTRL at high values of ϵ, and be competitive for lower values of ϵ. This result illustrates the FedBuff's flexibility to be adapted for privacy. Even with DP, we find that $K = 10$ is good setting. We find that a small L can counteract the additional noise from taking more steps. For more details see Appendix C.5.

7 Related Work

In addition to the discussion in Section 2 on related works, we discuss the other efforts in related domains.

Asynchronous stochastic optimization. Asynchronous stochastic optimization in shared-memory and distributed-memory systems has been extensively studied (Bertsekas and Tsitsiklis (1989); Chaturapruek et al. (2015); Niu et al. (2011); Lian et al. (2015, 2018); Chen et al. (2016); Zheng et al. (2017); Mania et al. (2017); Leblond et al. (2017); Reddi et al. (2015); Assran et al. (2020)). Asynchronous training is resilient to stragglers in both centralized and federated settings. The idea of aggregating K asynchronous updates for convex objectives has been studied in (Dutta et al., 2018). However, Dutta et al. (2018) provide a guarantee for non-convex objectives, but their assumption on the relationship between staleness and gradient moments is difficult enforce, in contrast to bounded staleness which can be easily enforce. In this work, we consider heterogeneous objectives and show that in a federated environment with a large number of clients, the source of speed-up is not only due to avoiding stragglers but also achieving better efficiency at high concurrency.

Large-batch training. Many proposals aim to understand and characterize conditions under which linear speed-up for distributed SGD and local SGD is achievable (Lin et al. (2018); Yu et al. (2019a); Woodworth et al. (2020); Haddadpour et al. (2019)). It is well accepted that increasing concurrency eventually saturates beyond a certain batch size in synchronous methods (Yin et al. (2017); Ott et al. (2018); Goyal et al. (2017); Ott et al. (2018); You et al. (2019, 2018, 2017); Shallue et al. (2018)). However, most existing research focuses on scalability across tens of server workers, each having iid-data - very different from the FL setting.

8 Conclusions

In this paper, we propose FedBuff, an asynchronous FL training scheme with buffered aggregation. Compared to SyncFL proposals, FedBuff scales to large values of concurrency. We analyze the convergence behavior of FedBuff in the non-convex setting. Empirical evaluation shows that FedBuff is up to 3.3× more efficient than FedAvgM, and up to 2.5× more efficient than FedAsync. As for future work, we are aware that FedBuff relies on TEE for SecAgg and TEE has a limited memory. Moreover, our analyses is on standard SGD. We leave extending the analysis to include momentum or adaptive learning rates as future work.

Acknowledgements

We would like to thank Ilya Mironov, Maziar Sanjabi, Graham Cormode, Samuel Horvath and Luca Melis for the meaningful discussions and their valuable suggestions which significantly improved the quality of this paper. We would like to also thank our anonymous reviewers for their insightful feedback.

References

M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 308–318, 2016.

M. Assran, A. Aytekin, H. R. Feyzmahdavian, M. Johansson, and M. G. Rabbat. Advances in asynchronous parallel and distributed optimization. *Proceedings of the IEEE*, 108(11):2013–2031, 2020.

J. H. Bell, K. A. Bonawitz, A. Gascón, T. Lepoint, and M. Raykova. Secure single-server aggregation with (poly) logarithmic overhead. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 1253–1269, 2020.

D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Prentice-Hall, 1989.

A. Bittau, Ú. Erlingsson, P. Maniatis, I. Mironov, A. Raghunathan, D. Lie, M. Rudominer, U. Kode, J. Tinnes, and B. Seefeld. Prochlo: Strong privacy for analytics in the crowd. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 441–459, 2017.

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

K. A. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth. Practical secure aggregation for federated learning on user-held data. In *NIPS Workshop on Private Multi-Party Machine Learning*, 2016. URL `https://arxiv.org/abs/1611.04482`.

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

N. Carlini, F. Tramèr, E. Wallace, M. Jagielski, A. Herbert-Voss, K. Lee, A. Roberts, T. B. Brown, D. Song, Ú. Erlingsson, A. Oprea, and C. Raffel. Extracting training data from large language models. *CoRR*, abs/2012.07805, 2020. URL `https://arxiv.org/abs/2012.07805`.

Z. Chai, Y. Chen, L. Zhao, Y. Cheng, and H. Rangwala. Fedat: A communication-efficient federated learning method with asynchronous tiers under non-iid data. *arXiv preprint arXiv:2010.05958*, 2020.

Z. Charles, Z. Garrett, Z. Huo, S. Shmulyian, and V. Smith. On large-cohort training for federated learning. *arXiv preprint arXiv:2106.07820*, 2021.

S. Chaturapruek, J. C. Duchi, and C. Ré. Asynchronous stochastic convex optimization: the noise is in the noise and sgd don't care. *Advances in Neural Information Processing Systems*, 28:1531–1539, 2015.

J. Chen, R. Monga, S. Bengio, and R. Jozefowicz. Revisiting distributed synchronous sgd. In *International Conference on Learning Representations Workshop Track*, 2016. URL `https://arxiv.org/abs/1604.00981`.

Y. Chen, Y. Ning, M. Slawski, and H. Rangwala. Asynchronous online federated learning for edge devices with non-iid data. *arXiv preprint arXiv:1911.02134*, 2019.

S. Dutta, G. Joshi, S. Ghosh, P. Dube, and P. Nagpurkar. Slow and stale gradients can win the race: Error-runtime trade-offs in distributed sgd. In *International Conference on Artificial Intelligence and Statistics*, pages 803–812. PMLR, 2018.

C. Dwork, A. Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3-4):211–407, 2014.

Ú. Erlingsson, V. Feldman, I. Mironov, A. Raghunathan, S. Song, K. Talwar, and A. Thakurta. Encode, shuffle, analyze privacy revisited: Formalizations and empirical evaluation. *arXiv preprint arXiv:2001.03618*, 2020.

J. Geiping, H. Bauermeister, H. Dröge, and M. Moeller. Inverting gradients–how easy is it to break privacy in federated learning? *arXiv preprint arXiv:2003.14053*, 2020.

A. Go, R. Bhayani, and L. Huang. Twitter sentiment classification using distant supervision. *CS224N project report, Stanford*, 1(12):2009, 2009.

P. Goyal, P. Dollár, R. B. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He. Accurate, large minibatch SGD: training imagenet in 1 hour. *CoRR*, abs/1706.02677, 2017. URL `http://arxiv.org/abs/1706.02677`.

F. Haddadpour and M. Mahdavi. On the convergence of local descent methods in federated learning. *arXiv preprint arXiv:1910.14425*, 2019.

F. Haddadpour, M. M. Kamani, M. Mahdavi, and V. R. Cadambe. Local sgd with periodic averaging:

Tighter analysis and adaptive synchronization. *arXiv preprint arXiv:1910.13598*, 2019.

K. Hsieh, A. Phanishayee, O. Mutlu, and P. Gibbons. The non-iid data quagmire of decentralized machine learning. In *International Conference on Machine Learning*, pages 4387–4398. PMLR, 2020.

T.-M. H. Hsu, H. Qi, and M. Brown. Measuring the effects of non-identical data distribution for federated visual classification. *arXiv preprint arXiv:1909.06335*, 2019.

S. Jastrzebski, Z. Kenton, D. Arpit, N. Ballas, A. Fischer, Y. Bengio, and A. J. Storkey. Three factors influencing minima in SGD. *CoRR*, abs/1711.04623, 2017. URL `http://arxiv.org/abs/1711.04623`.

P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings, et al. Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977*, 2019.

P. Kairouz, B. McMahan, S. Song, O. Thakkar, A. Thakurta, and Z. Xu. Practical and private (deep) learning without sampling or shuffling. *arXiv preprint arXiv:2103.00039*, 2021.

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

R. Karl, J. Takeshita, and T. Jung. Cryptonite: A framework for flexible time-series secure aggregation with online fault tolerance. 2020. `https://eprint.iacr.org/2020/1561`.

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

M. Lam, G.-Y. Wei, D. Brooks, V. J. Reddi, and M. Mitzenmacher. Gradient disaggregation: Breaking privacy in federated learning by reconstructing the user participant matrix. *arXiv preprint arXiv:2106.06089*, 2021.

R. Leblond, F. Pedregosa, and S. Lacoste-Julien. ASAGA: Asynchronous Parallel SAGA. In A. Singh and J. Zhu, editors, *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pages 46–54, Fort Lauderdale, FL, USA, 20–22 Apr 2017. PMLR. URL `http://proceedings.mlr.press/v54/leblond17a.html`.

K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran. Speeding up distributed machine learning using codes. *IEEE Transactions on Information Theory*, 64(3):1514–1529, 2017.

T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith. Federated optimization in heterogeneous networks. *arXiv preprint arXiv:1812.06127*, 2018.

X. Li, W. Yang, S. Wang, and Z. Zhang. Communication efficient decentralized training with multiple local updates. 2019.

X. Li, K. Huang, W. Yang, S. Wang, and Z. Zhang. On the convergence of fedavg on non-iid data. 2020.

X. Li, Z. Qu, B. Tang, and Z. Lu. Stragglers are not disaster: A hybrid federated learning algorithm with delayed gradients. *arXiv preprint arXiv:2102.06329*, 2021.

X. Lian, Y. Huang, Y. Li, and J. Liu. Asynchronous parallel stochastic gradient for nonconvex optimization. *Advances in neural information processing systems*, 2015.

X. Lian, W. Zhang, C. Zhang, and J. Liu. Asynchronous decentralized parallel stochastic gradient descent. In *International Conference on Machine Learning*, pages 3043–3052. PMLR, 2018.

T. Lin, S. U. Stich, K. K. Patel, and M. Jaggi. Don't use large mini-batches, use local sgd. *arXiv preprint arXiv:1808.07217*, 2018.

Z. Liu, P. Luo, X. Wang, and X. Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

H. Mania, X. Pan, D. Papailiopoulos, B. Recht, K. Ramchandran, and M. I. Jordan. Perturbed iterate analysis for asynchronous stochastic optimization. *SIAM Journal on Optimization*, 27(4):2202–2229, 2017.

H. B. McMahan, E. Moore, D. Ramage, and B. A. y Arcas. Federated learning of deep networks using model averaging. *arXiv preprint arXiv:1602.05629*, 2016.

H. B. McMahan, D. Ramage, K. Talwar, and L. Zhang. Learning differentially private recurrent language models. In *International Conference on Learning Representations*, 2018.

L. Melis, C. Song, E. De Cristofaro, and V. Shmatikov. Exploiting unintended feature leakage in collaborative learning. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 691–706. IEEE, 2019.

F. Mo, H. Haddadi, K. Katevas, E. Marin, D. Perino, and N. Kourtellis. Ppfl: privacy-preserving federated learning with trusted execution environments. *arXiv preprint arXiv:2104.14380*, 2021.

F. Niu, B. Recht, C. Re, and S. J. Wright. Hogwild!: A lock-free approach to parallelizing stochastic gradient descent, 2011.

M. Ott, S. Edunov, D. Grangier, and M. Auli. Scaling neural machine translation. *arXiv preprint arXiv:1806.00187*, 2018.

A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. 2017.

J. Pennington, R. Socher, and C. D. Manning. In *EMNLP*.

S. Reddi, Z. Charles, M. Zaheer, Z. Garrett, K. Rush, J. Konečnỳ, S. Kumar, and H. B. McMahan. Adaptive federated optimization. *arXiv preprint arXiv:2003.00295*, 2020.

S. J. Reddi, A. Hefny, S. Sra, B. Poczos, and A. Smola. On variance reduction in stochastic gradient descent and its asynchronous variants. *Advances in neural information processing systems*, 2015.

A. Reisizadeh, I. Tziotis, H. Hassani, A. Mokhtari, and R. Pedarsani. Straggler-resilient federated learning: Leveraging the interplay between statistical accuracy and system heterogeneity. *arXiv preprint arXiv:2012.14453*, 2020.

C. J. Shallue, J. Lee, J. Antognini, J. Sohl-Dickstein, R. Frostig, and G. E. Dahl. Measuring the effects of data parallelism on neural network training. *arXiv preprint arXiv:1811.03600*, 2018.

S. L. Smith, P.-J. Kindermans, C. Ying, and Q. V. Le. Don't decay the learning rate, increase the batch size. *arXiv preprint arXiv:1711.00489*, 2017.

J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, 25:2951–2959, 2012.

J. So, B. Güler, and A. S. Avestimehr. Turbo-aggregate: Breaking the quadratic aggregation barrier in secure federated learning. *IEEE Journal on Selected Areas in Information Theory*, 2(1):479–489, 2021.

S. U. Stich. Local sgd converges fast and communicates little. 2019.

R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis. Gradient coding: Avoiding stragglers in distributed learning. In *International Conference on Machine Learning*, pages 3368–3376. PMLR, 2017.

M. van Dijk, N. V. Nguyen, T. N. Nguyen, L. M. Nguyen, Q. Tran-Dinh, and P. H. Nguyen. Asynchronous federated learning with reduced number of rounds and with differential privacy from less aggregated gaussian noise. *arXiv preprint arXiv:2007.09208*, 2020.

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

L. Watson, C. Guo, G. Cormode, and A. Sablayrolles. On the importance of difficulty calibration in membership inference attacks. *CoRR*, abs/2111.08440, 2021. URL `https://arxiv.org/abs/2111.08440`.

B. Woodworth, K. K. Patel, S. Stich, Z. Dai, B. Bullins, B. Mcmahan, O. Shamir, and N. Srebro. Is local sgd better than minibatch sgd? In *International Conference on Machine Learning*, pages 10334–10343. PMLR, 2020.

W. Wu, L. He, W. Lin, R. Mao, C. Maple, and S. A. Jarvis. Safa: a semi-asynchronous protocol for fast federated learning with low overhead. *IEEE Transactions on Computers*, 2020.

Y. Wu and K. He. Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19, 2018.

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

D. Yin, A. Pananjady, M. Lam, D. S. Papailiopoulos, K. Ramchandran, and P. L. Bartlett. Gradient diversity empowers distributed learning. *CoRR*, abs/1706.05699, 2017. URL `http://arxiv.org/abs/1706.05699`.

Y. You, I. Gitman, and B. Ginsburg. Large batch training of convolutional networks. *arXiv preprint arXiv:1708.03888*, 2017.

Y. You, Z. Zhang, C.-J. Hsieh, J. Demmel, and K. Keutzer. Imagenet training in minutes. In *Proceedings of the 47th International Conference on Parallel Processing*, pages 1–10, 2018.

Y. You, J. Li, S. Reddi, J. Hseu, S. Kumar, S. Bhojanapalli, X. Song, J. Demmel, K. Keutzer, and C.-J. Hsieh. Large batch optimization for deep learning: Training bert in 76 minutes. *arXiv preprint arXiv:1904.00962*, 2019.

H. Yu, R. Jin, and S. Yang. On the linear speedup analysis of communication efficient momentum sgd for distributed non-convex optimization. In *International Conference on Machine Learning*, pages 7184–7193. PMLR, 2019a.

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

Q. Yu, S. Li, N. Raviv, S. M. M. Kalan, M. Soltanolkotabi, and S. A. Avestimehr. Lagrange coded computing: Optimal design for resiliency, security, and privacy. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 1215–1225. PMLR, 2019c.

Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr. Straggler mitigation in distributed matrix multiplication: Fundamental limits and optimal coding. *IEEE Transactions on Information Theory*, 66(3):1920–1933, 2020.

S. Zheng, Q. Meng, T. Wang, W. Chen, N. Yu, Z.-M. Ma, and T.-Y. Liu. Asynchronous stochastic gradient descent with delay compensation. In *International Conference on Machine Learning*, pages 4120–4129. PMLR, 2017.

L. Zhu and S. Han. Deep leakage from gradients. In *Federated learning*, pages 17–31. Springer, 2020.

Appendix

A Relationship Between Maximum Staleness and K

Recall Assumption 5, that the staleness when executing FedBuff with $K = 1$ is always bounded as $\tau_i(t) \leq \tau_{\max,1}$. In this section we will show that this implies the staleness bound $\tau_i(t) \leq \lceil \tau_{\max,K} \rceil \leq \lceil \tau_{\max,1}/K \rceil$ when running FedBuff with $K > 1$.

Consider an execution of FedBuff. Let r_i denote the time when the i'th client update is received by the server, and let $s_i < r_i$ denote the time when the client downloaded the serve model before performing local steps that resulted in the model update received at r_i.

When $K = 1$, the staleness $\tau_i^{(1)}$ of the i'th update corresponds to the number of updates that occurs between when the client downloaded the model and when it completed local training and uploaded the model to the server,

$$\tau_i^{(K=1)} = |\{j \colon s_i < r_j < r_i\}|.$$

If Assumption 5 holds, then $\max_i \tau_i^{(1)} \leq \tau_{\max,1}$.

When $K > 1$, the server waits to aggregate K client updates before stepping the global model. Thus, if $\tau_i^{(K=1)}$ client updates are received between the times s_i and r_i, then at most $\tau_i^{(1)}/K$ server updates occur during this time. Hence $\tau_i^{(K)} \leq \lceil \tau_i^{(1)}/K \rceil$, and therefore

$$\tau_{\max,K} = \max_i \tau_i^{(K)} \leq \max_i \lceil \tau_i^{(1)}/K \rceil \leq \lceil \tau_{\max,1}/K \rceil.$$

Note that the times s_i and r_i only depend on the number of clients training concurrently, and the distribution of client execution times (the time it takes a client to complete one round of local updates; i.e., the distribution of $r_i - s_i$). These times are not impacted by the choice of K; rather K only affects how frequently the server performs an update. Thus, the arguments above hold regardless of the distribution of client execution times, and only depend on Assumption 5.

We also remark that the same relationship holds for the average delay; i.e., increasing K reduces average delay. In particular, let

$$\overline{\tau}_1 = \lim_{N \to \infty} \frac{1}{N} \sum_{i=1}^{N} \tau_i^{(1)},$$

and suppose the limit exists. Clearly, if the limit exists and Assumption 5 holds, then $\overline{\tau}_1 \leq \tau_{\max,1}$. Furthermore, then

$$\begin{aligned} \overline{\tau}_K &= \lim_{N \to \infty} \frac{1}{N} \sum_{i=1}^{N} \tau_i^{(K)} \\ &\leq \lim_{N \to \infty} \frac{1}{N} \sum_{i=1}^{N} \tau_i^{(1)}/K \\ &= \overline{\tau}_1/K. \end{aligned}$$

B Experiment Details

B.1 Datasets and Models

Sent140. We train a sentiment classifier on tweets from the Sent140 dataset (Caldas et al., 2018; Go et al., 2009) with a two-layer LSTM binary classifier. The dataset has 660,120 clients where each client is a Twitter account. The LSTM binary classifier contains 100 hidden units with a top 10,000 pretrained word embedding from 300D GloVe (Pennington et al.). The model has a max sequence length of 25 characters. The model first embeds each of the characters into a 300-dimensional space by looking up GloVe, passes through 2 LSTM layers and a 128 hidden unit linear layer to output labels 0 or 1. We set the dropout rate to 0.1. We split the data into 80%

Table 3: The best performing hyperparameters for fig. 4

	FedBuff + DP-FTRL	SyncFL + DP-SGD	SyncFL + DP-FTRL
$\epsilon = 6$	$\eta_\ell = 1.0$ $\eta_g = 4.3$ $\beta = 9.9 \cdot 10^{-1}$ $L = 1.2 \cdot 10^{-4}$	$\eta_\ell = 1.0 \cdot 10^{-1}$ $\eta_g = 5.9 \cdot 10^{2}$ $\beta = 3.0 \cdot 10^{-1}$ $L = 1.1 \cdot 10^{-2}$	$\eta_\ell = 1.0 \cdot 10^{-3}$ $\eta_g = 2.6 \cdot 10^{4}$ $\beta = 1.0 \cdot 10^{-1}$ $L = 2.7 \cdot 10^{-4}$
$\epsilon = 12$	$\eta_\ell = 1.0 \cdot 10^{-2}$ $\eta_g = 5.4 \cdot 10^{1}$ $\beta = 0$ $L = 1.1 \cdot 10^{-3}$	$\eta_\ell = 1.0$ $\eta_g = 1.0 \cdot 10^{4}$ $\beta = 5.0 \cdot 10^{-1}$ $L = 7.6 \cdot 10^{-3}$	$\eta_\ell = 1.0$ $\eta_g = 1.0 \cdot 10^{2}$ $\beta = 9.0 \cdot 10^{-1}$ $L = 2.7 \cdot 10^{-1}$
$\epsilon = 24$	$\eta_\ell = 1.0 \cdot 10^{-1}$ $\eta_g = 8.7 \cdot 10^{2}$ $\beta = 3.0 \cdot 10^{-1}$ $L = 1.0 \cdot 10^{-4}$	$\eta_\ell = 1.0 \cdot 10^{-1}$ $\eta_g = 5.1 \cdot 10^{2}$ $\beta = 3.0 \cdot 10^{-1}$ $L = 1.4 \cdot 10^{-2}$	$\eta_\ell = 1.0$ $\eta_g = 8.0 \cdot 10^{3}$ $\beta = 5.0 \cdot 10^{-1}$ $L = 1.0 \cdot 10^{-4}$

training set, 10% validation set, and 10% test set using script provided by Caldas et al. (2018). Due to memory constraint, we use 15% of the entire dataset using the script provided by Caldas et al. (2018), with split seed = 1549775860.

CelebA. We study an image classification problem on the CelebA dataset (Liu et al., 2015; Caldas et al., 2018) using a four layer CNN binary classifier with dropout rate of 0.1, stride of 1, and padding of 2. As it is standard with image datasets, we preprocess train, validation, and test images; we resize and center crop each image to 32×32 pixels, then normalize by 0.5 mean and 0.5 standard deviation. The dataset has 9,343 clients where each client is a unique celebrity.

CIFAR-10. We evaluate a multi-class image classification problem on CIFAR-10 (Krizhevsky et al., 2009) using a four layer CNN binary classifier with dropout rate of 0.1, stride of 1, and padding of 2. We normalize the images by the dataset mean and standard deviation. Following Hsu et al. (2019), we partition the dataset into 5,000 clients using a Dirichlet distribution with parameter 0.1 and split seed = 0.

B.2 Implementation Details

We implemented all algorithms in Pytorch (Paszke et al., 2017) and evaluated them on a cluster of machines, each with eight NVidia V100 GPUs. Independently, we built a simulator to simulate large-scale federated learning environments. The simulator can realistically simulate clients, server, communication channels between clients and server, model aggregation schemes, and local training of clients. We intend to open-source the simulator, making it available for the research community.

For our experiments, we assume clients arrive to the FL system at a constant rate. To simulate device heterogeneity, we sample each client training duration from a half-normal, uniform, or exponential distribution. Moreover, our implementation has two other important distinctions. First, each client does one epoch of training over its local data; this distinction stems from two observations in our production stack: that our FL production stack has plenty of users to train on, and that we train small capacity models in FL (e.g., less than 10 million parameters) because of bandwidth and client compute. Second, we use the weighted sum of the client updates instead of the weighted average. This is because each client update has different levels of staleness; taking the average cannot capture the true contribution for each client.

B.3 Hyperparameters

For all experiments, we tune hyperparameters using Bayesian optimization (Snoek et al., 2012). For optimizer on clients, we use minibatch SGD for all tasks. We select the best hyperparameters based on the number of rounds to reach target validation accuracy for each dataset.

Table 4: The best performing hyperparameters for Table 1

	FedBuff	FedAsync	FedAvgM	FedAvg	FedProx
CelebA	$\eta_\ell = 4.7 \cdot 10^{-6}$ $\eta_g = 1.0 \cdot 10^{3}$ $\beta = 3.0 \cdot 10^{-1}$	$\eta_\ell = 5.7$ $\eta_g = 2.8 \cdot 10^{-3}$	$\eta_\ell = 1.1 \cdot 10^{-1}$ $\eta_g = 2.4 \cdot 10^{-1}$ $\beta = 8.3 \cdot 10^{-1}$	$\eta_\ell = 1.0 \cdot 10^{2}$ $\eta_g = 1.6 \cdot 10^{-3}$	$\eta_\ell = 4.9 \cdot 10^{-4}$ $\eta_g = 1.0 \cdot 10^{2}$ $\mu = 1.0 \cdot 10^{-2}$
Sent140	$\eta_\ell = 1.3 \cdot 10^{1}$ $\eta_g = 4.9 \cdot 10^{-2}$ $\beta = 5.0 \cdot 10^{-1}$	$\eta_\ell = 1.7 \cdot 10^{1}$ $\eta_g = 1.5 \cdot 10^{-2}$	$\eta_\ell = 1.5$ $\eta_g = 3.4 \cdot 10^{-1}$ $\beta = 9.0 \cdot 10^{-1}$	$\eta_\ell = 2.6 \cdot 10^{-3}$ $\eta_g = 1.0 \cdot 10^{3}$	$\eta_\ell = 2.0 \cdot 10^{-3}$ $\eta_g = 1.0^{3}$ $\mu = 1.0 \cdot 10^{-3}$
CIFAR-10	$\eta_\ell = 1.95 \cdot 10^{-4}$ $\eta_g = 4.09 \cdot 10^{1}$ $\beta = 0$	$\eta_\ell = 1.0 \cdot 10^{2}$ $\eta_g = 6.4 \cdot 10^{-5}$	$\eta_\ell = 1.0 \cdot 10^{1}$ $\eta_g = 1.02 \cdot 10^{-3}$ $\beta = 9.0 \cdot 10^{-1}$	$\eta_\ell = 1.0 \cdot 10^{1}$ $\eta_g = 1.02 \cdot 10^{-3}$	$\eta_\ell = 1.0 \cdot 10^{1}$ $\eta_g = 1.02 \cdot 10^{-3}$ $\mu = 1.0 \cdot 10^{-3}$

Table 5: Speed up of FedBuff over FedAvgM and FedAsync with regards to number of client trips to reach target validation accuracy, for different delay distributions. We set concurrency = 1000 for all methods and $K = 10$ for FedBuff. FedBuff's speed up is consistent across delay distributions.

Dataset	Delay Distribution	Speedup over FedAvgM	Speedup over FedAsync
CelebA	Uniform	4.7×	1.6×
	Half-Normal	3.3×	1.2×
	Exponential	4.3×	1.1×
Sent140	Uniform	1.3×	1.2×
	Half-Normal	1.7×	2.5×
	Exponential	1.4×	2.0×

B.3.1 Hyperparameter Ranges

Below, we show the range for the client learning rate (η_ℓ), server learning rate (η_g), server momentum (β), proximal term (μ) sweep ranges.

$$\begin{aligned} \beta &\in \{0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.99\} \\ \eta_\ell &\in [1 \cdot 10^{-8}, 10000] \\ \eta_g &\in [1 \cdot 10^{-8}, 10000] \\ \mu &\in \{0.001, 0.01, 0.1, 1\} \end{aligned}$$

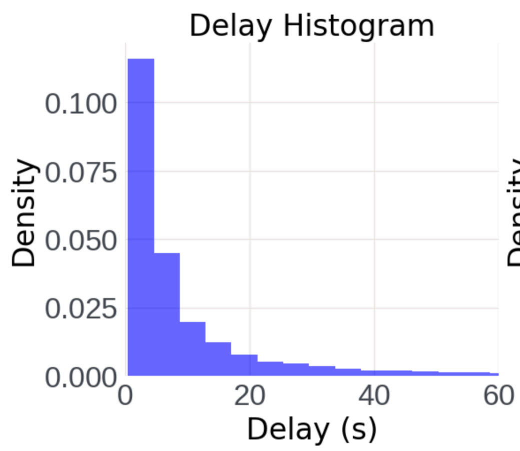
B.3.2 Best Performing Hyperparameters

Table 4 illustrates the best value for client and server learning rates (η_ℓ, η_g), server momentum (β), and proximal term (μ) for tasks in Table 1. For experiments in Table 6, we set staleness exponent $\alpha = 10$. We set $\alpha = 0.5$ for all other experiments.

C Additional Experiments

C.1 Robustness to Delay Distributions.

In this section, we analyze the sensitivity of FedBuff to different staleness distributions. We compare FedBuff against other competing algorithms with different staleness distributions. Table 5 demonstrates that FedBuff is robust and FedBuff's speed up is consistent. To have an accurate view of real-world delays, we observe the delays and their resulting staleness distribution in our production stack when training over millions of clients with concurrency = 1000 and K = 100. Figure 5 demonstrates that a half-normal is a suitable delay distribution.



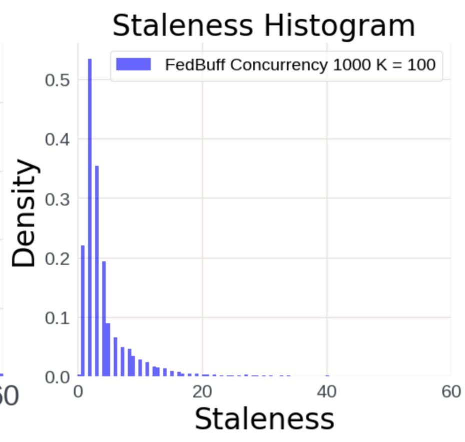


Figure 5: Delay and staleness distributions observed in production when training over millions of real clients for FedBuff.

C.2 Wall-Clock Time Simulation

In this section, we study the speed up of FedBuff over FedAvgM in terms of wall-clock time for various concurrency levels. The results for Sent140 are in Figure 6.

In cross-device FL, each client is a mobile phone with limited compute power and communication bandwidth (Kairouz et al., 2019). Moreover, clients can have vastly different number of examples. Recall, SyncFL methods wait for all the participating clients in a round to finish before updating the server model – a round proceeds at the pace of the slowest client, the straggler effect. To mitigate the straggler problem, over-selection proposed in Bonawitz et al. (2019), which selects 30% more clients than the target number of clients to participate and waits for the fastest replies.

To confirm the speedup gain by FedBuff in terms of wall-clock time, we simulate training time of FedAvgM and FedBuff using a random exponential time model from Lee et al. (2017). The random exponential time model has been widely used to simulate the straggler effect in federated learning, e.g. in Reisizadeh et al. (2020); Tandon et al. (2017); Yu et al. (2020, 2019c); Charles et al. (2021).

We assume the time a client requires to perform local training is proportional to the number of examples the client has, same as the assumption in Charles et al. (2021). Formally, let n_i be the number of examples held by client i, and let T_i be the amount of time required by client i to perform local training. Also assume that there is a constant $\lambda > 0$ such that

$$T_i \sim Exp(\frac{1}{\lambda n_i}).$$

In this context, λ is the straggler parameter. The larger the λ, the longer the expected client training time. For a given round t, let C_t be number of clients required to close a round, and let M be the total number of clients training concurrently with over-selection. If $T_1, \ldots, T_M$ denote the raw times when clients complete the round and $T_{(1)} \leq \cdots \leq T_{(M)}$ denote order statistic of the client training time, then R_t for SyncFL is

$$R_t = T_{(C_t)}.$$

If T is the number of rounds to reach a target accuracy, the expected total training time for SyncFL is

$$\mathbb{E}[T_{\text{SyncFL}}] = \sum_{t=0}^{T} R_t.$$

In the case of FedBuff, the total wall-clock time is when the last client required to reach a target accuracy finishes training. Formally, let N be the number of clients required to reach a target accuracy and $T_{(1)} \leq \cdots \leq T_{(N)}$ denotes the order statistic of the client training time. Then the expected total training time for FedBuff is

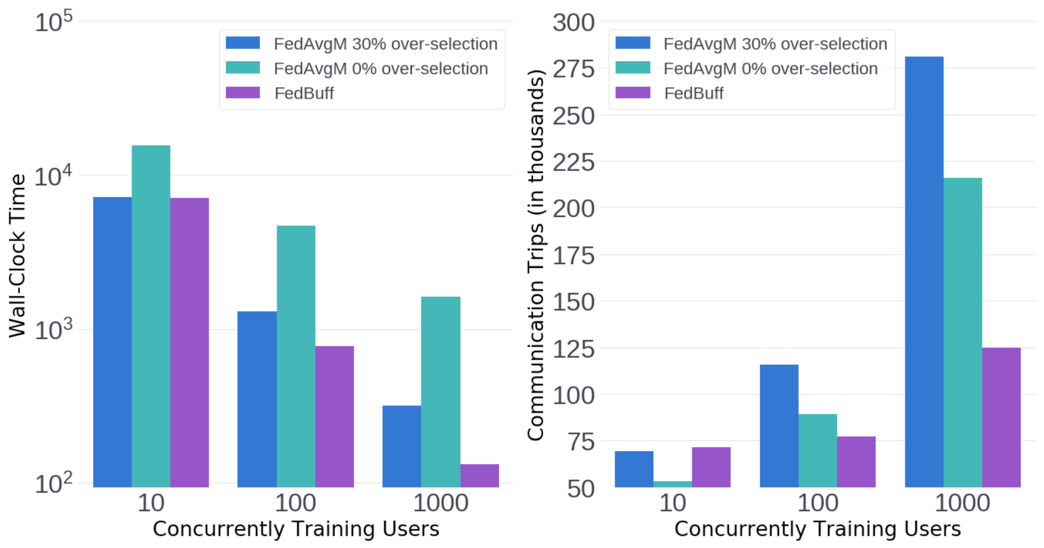


Figure 6: (left) The total run time required to reach a target accuracy on Sent140 with $\lambda = 1$ under a random exponential time model. (right) The number of client trips required to reach a target accuracy. This measures the resource efficiency of the three algorithms. In all three configurations, increasing concurrency lowers wall-clock convergence time. However, SyncFL uses much more resources compared with AsyncFL. This highlights the importance of scalability, the ability to efficiently utilize increasing number of clients training in parallel. Both figures illustrates that FedBuff is faster than SyncFL (e.g., FedAvgM) even with over-selection with increasing concurrency, while being up to 3 times more resource efficient.

$$\mathbb{E}[T_{\text{FedBuff}}] = T_{(N)}.$$

C.3 Bias

In this section, we show the diversity of participating clients in FedBuff and FedAvgM. We should that SyncFL methods can introduce bias in their selection process while FedBuff does not. We use the same random exponential time model in Appendix C.2. The result is given in Figure 7. We find that in all levels of λ, FedBuff can incorporate clients with large local datasets. On the other hand, SyncFL (e.g., FedAvgM) with over-selection drops these clients leading to bias in the selection process. FedBuff does not drop the slowest clients, and can incorporate these clients with many examples.

C.4 Large values of K

In this section, we study the performance of FedBuff with large values of K. In Table 2, we observe that FedBuff trains fast when running with small values of K, relative to the concurrency. However, large values of K are useful when providing user-level differential privacy, as essentially the noise is divided among larger number of clients (larger values of K) (Kairouz et al., 2021; McMahan et al., 2018).

We compare the training speed of FedBuff and FedAvgM in a setting where both algorithms produce a server update from the same number of aggregated client updates. We fix concurrency at 1000, and have both FedAvgM and FedBuff perform updates after aggregating responses from $K = 1000$ clients. In this setting, FedBuff's main advantage is robustness to stragglers. It cannot take advantage of frequent server updates, yet still needs to deal with staleness.

The synchronous FL system described in Bonawitz et al. (2019) uses over-selection, typically by 30%, to address stragglers. For example, if 1000 users are needed to produce a server model update, 1300 users are selected. The round will finish when the fastest 1000 users finish training. Results from the slowest 300 users will be thrown away. Over-selection makes synchronous FL more robust to stragglers, but at the cost of wasting some clients' compute and bandwidth.

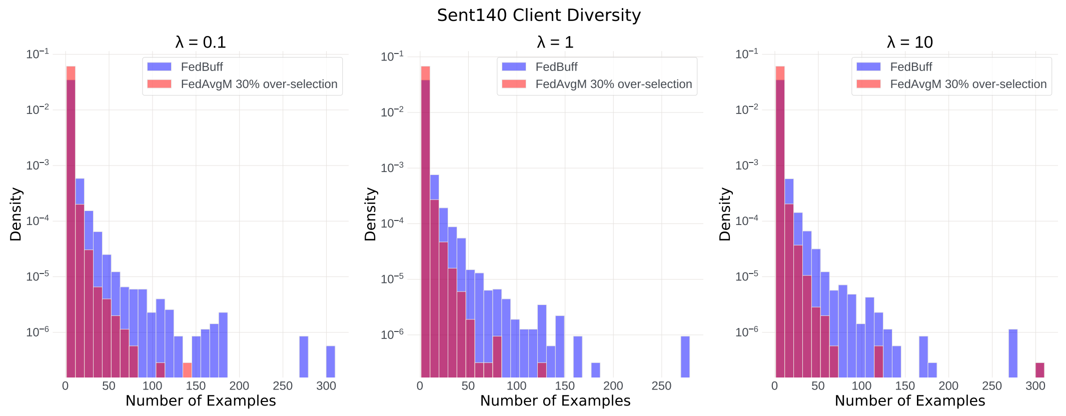


Figure 7: The distribution of Number of examples held by each participating client on Sent140 with varying λ. We fix concurrency = 1000 and buffer size $K = 10$. Since FedBuff does not drop the slowest clients, it can incorporate a more diverse set of clients.

Table 6: Wall-clock time to reach target validation accuracy on CelebA and Sent140 when K is large (Units for wall-clock time: mean training time for one client. Units for client trips updates: 1000 updates). For FedBuff, K=1000. FedAvgM with over-selection throws away results from the slowest 30% of users in each round. These users are included when calculating the number of client trips

Dataset	Algorithm	Concurrency	Wall-Clock Time	client trips
CelebA	FedBuff (K=1000)	1000	124	124
	FedAvgM	1000	446 (3.6×)	104
	FedAvgM, over-selection	1300	155 (1.25×)	135
Sent140	FedBuff (K=1000)	1000	228	228
	FedAvgM	1000	927 (4.06×)	216
	FedAvgM. over-selection	1300	322 (1.41×)	281

Table 6 reports the wall-clock training time and number of client trips to reach target accuracy for FedBuff and FedAvgM with and without over-selection. We assume a half-normal training duration distribution since that matches the behavior observed in our production system (see Figure 5). We find that over-selection reduces the impact of stragglers significantly. However, even with over-selection, FedBuff is 25%-41% faster than FedAvgM, despite using 30% lower concurrency.

C.5 FedBuff with Differential Privacy

Figure 8 shows the training curves of FedBuff with DP-FTRL, SyncFL with DP-SGD and DP-FTRL. At low values of ϵ, FedBuff can achieve the same utility as SyncFL with amplified DP-SGD at the cost of slower convergence. At the same ϵ, FedBuff with DP-FTRL achieves better utility and faster convergence compared to SyncFL with DP-FTRL. We find the source for this speed-up is from FedBuff's ability to tolerate much lower clipping norm value L. We repeat each experiment for 3 different seeds and take the average. The seeds are 0, 1, and 2.

C.6 Learning Rate Normalization (LR-Norm)

C.6.1 Theoretical Justification

Recall that LR-Norm described in Section 5 aims to address the situation where a client performing local updates may need to perform an update using a batch size b smaller than the server-prescribed batch size B. This may occur when processing a batch at the end of one epoch, including the first batch if the client has fewer than B samples in total. Since this only pertains to the local updates performed at clients, let us simply write such an

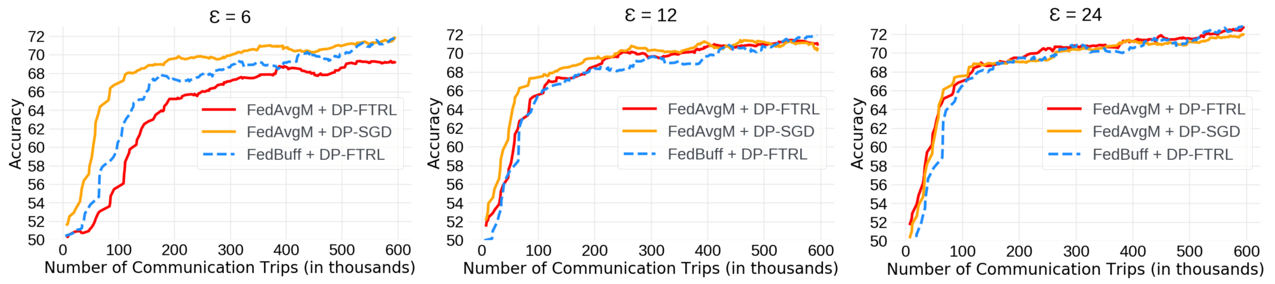


Figure 8: Training curves for different values of ϵ for three FL configurations.

update as

$$y_q = y_{q-1} - \eta_q g_q^{(b_q)}, \tag{5}$$

without referring to any specific client index i or global iteration index t. Here $g_q^{(b_q)}$ denotes a stochastic gradient of F (the client's local objective) evaluated at y_q using batch size b_q.

Assume that F is L-smooth, i.e.,

$$\|\nabla F(y) - \nabla F(y')\| \leq L \|y - y'\| .$$

Also assume that the stochastic gradients are unbiased and have variance satisfying a weak growth condition. Specifically, assume that with batch size $b_q = 1$,

$$\mathbb{E}[g_q^{(1)} | y_q] = \nabla F(y_q),$$
$$\mathbb{E}[\left\|g_q^{(1)} - \nabla F(y_q)\right\|^2] \leq \sigma_\ell^2 + M \|\nabla F(y_q)\|^2 .$$

Note that in the proof of Theorem 1, we make the stronger assumption of bounded variance, corresponding to $M = 0$.

Furthermore, suppose that a mini-batch stochastic gradient $g_q^{(n)}$ with batch size $b_q > 1$ is obtained by averaging the gradients evaluated at b_q independent and identically distributed samples. Thus,

$$\mathbb{E}[g_q^{(b_q)} | y_q] = \nabla F(y_q),$$
$$\mathbb{E}[\left\|g_q^{(b_q)} - \nabla F(y_q)\right\|^2] \leq \frac{\sigma_\ell^2}{b_q} + \frac{M}{b_q} \|\nabla F(y_q)\|^2 .$$

Uniform batch sizes. If all steps use the same batch size $b_q = B$ with constant step-size $\eta_q = \eta_\ell$ satisfying

$$0 < \eta_\ell \leq \frac{1}{L(M/B + 1)},$$

then it is well-known that the SGD iterates satisfy

$$\mathbb{E}\left[\frac{1}{Q}\sum_{q=1}^{Q} \|\nabla F(y_q)\|^2\right] \leq \frac{2(F(y_1) - F^*)}{\eta_\ell Q} + \frac{\eta_\ell L \sigma_\ell^2}{B};$$

see, for example, Theorem 4.8 in L. Bottou, F. Curtis, and J. Nocedal, "Optimization methods for large-scale machine learning," *SIAM Review*, 2019.

Non-uniform batch sizes. Now suppose that some steps will use batch size $1 < b_q \leq B$. In this case one can show the following result.

Theorem. *Consider updates as in equation 5 with per-iteration batch size*

$$\eta_q = \eta_\ell \frac{b_q}{B},$$

and let $A_Q = \sum_{q=1}^{Q} \eta_q = \frac{\eta_\ell}{B} \sum_{q=1}^{Q} b_q$. *Suppose that* η_ℓ *satisfies*

$$0 < \eta_\ell \le \frac{1}{L(M/B+1)}.$$

Then

$$\mathbb{E}\left[\frac{1}{A_Q}\sum_{q=1}^{Q}\|\nabla F(y_q)\|^2\right] \le \frac{2(F(y_1)-F^*)}{A_Q} + \frac{\eta_\ell L \sigma_\ell^2}{B}.$$

First, note that A_Q is strictly increasing in Q, since $1 \le b_q \le B$. In the special case where $b_q = B$ for all q we exactly recover the result above for uniform batch sizes. More generally, when $b_q < B$ for some steps, the asymptotic residual is identical to the case with uniform-batch size. This justifies using the LR-Norm step-size rule $\eta_q = \eta_\ell b_q / B$ when encountering batches of size $b_q < B$. The proof follows from similar arguments to those of Theorem 4.8 in L. Bottou, F. Curtis, and J. Nocedal, "Optimization methods for large-scale machine learning," *SIAM Review*, 2019.

Proof. Let $\mathbb{E}_q$ denote expectation with respect to all randomness up to step y_q. Because F is L-smooth,

$$\mathbb{E}_q[F(y_{q+1})] - F(y_q) \le -\eta_q \left\langle \nabla F(y_q), \mathbb{E}_q[g_q^{(b_q)}]\right\rangle + \frac{\eta_q^2 L}{2}\mathbb{E}_k[\left\|g_q^{(b_q)}\right\|^2].$$

From the weak growth assumption, it follows that

$$\mathbb{E}_k[\left\|g_q^{(b_q)}\right\|^2] \le \frac{\sigma_\ell^2}{b_q} + \left(\frac{M}{b_q}+1\right)\|\nabla F(y_q)\|^2,$$

and thus

$$\begin{aligned}\mathbb{E}_q[F(y_{q+1})] - F(y_q) &\le -\eta_q \|\nabla F(y_q)\|^2 + \frac{\eta_q^2 L}{2}\left(\frac{\sigma_\ell^2}{b_q} + \left(\frac{M}{b_q}+1\right)\|\nabla F(y_q)\|^2\right) \\ &= -\eta_q\left(1 - \frac{\eta_q L}{2}\left(\frac{M}{b_q}+1\right)\right)\|\nabla F(y_q)\|^2 + \frac{\eta_q^2 L \sigma_\ell^2}{2b_q}.\end{aligned}$$

Based on the relationship $\eta_q = \eta_\ell b_q / B$ and the upper-bound assumed on η_ℓ, we have

$$\frac{\eta_q L}{2}\left(\frac{M}{b_q}+1\right) \le \frac{1}{2}.$$

Consequently,

$$\mathbb{E}_q[F(y_{q+1})] - F(y_q) \le -\frac{\eta_q}{2}\|\nabla F(y_q)\|^2 + \frac{\eta_q^2 L \sigma_\ell^2}{2b_q}.$$

Rearranging, we get

$$\frac{\eta_q}{2}\|\nabla F(y_q)\|^2 \le F(y_q) - \mathbb{E}_q[F(y_{q+1})] + \frac{\eta_q^2 L \sigma_\ell^2}{2b_q}.$$

Summing both sides over $q = 1, \ldots, Q$ and taking the total expectation yields

$$\begin{aligned}\sum_{q=1}^{Q}\frac{\eta_q}{2}\mathbb{E}[\|\nabla F(y_q)\|^2] &\le F(y_1) - \mathbb{E}[F(y_Q)] + \sum_{q=1}^{Q}\frac{\eta_q^2 L \sigma_\ell^2}{2n_q} \\ &\le F(y_1) - F^* + \sum_{q=1}^{Q}\frac{\eta_q^2 L \sigma_\ell^2}{2n_q}.\end{aligned}$$

Table 7: Number of client updates (lower is better) to reach validation accuracy on CelebA (90%) and Sent140 (69%). We set $M = 1000$ for all methods. We compare LR-Norm against two other popular weighting schemes. *Example weight* is when the weight is the number of training examples for each client. *Uniform weight* is where all clients have weights of 1. (Units = 1000 updates.)

Dataset	K	LR-Norm	Example Weight	Uniform Weight
	1	20.8	23.9	20.7
CelebA	10	27.1	25.5	28.7
	100	57.6	57.6	54.4
	1	190.0	201.9	201.9
Sent140	10	124.7	207.9	136.6
	100	178.2	570.3	231.7

Now, multiplying both sides by $2/A_Q$, we obtain

$$\begin{aligned}\frac{1}{A_Q}\sum_{q=1}^{Q}\eta_q\mathbb{E}[\|\nabla F(y_q)\|^2] &\leq \frac{2(F(y_1)-F^*)}{A_Q}+\frac{1}{A_Q}\sum_{q=1}^{Q}\frac{\eta_q^2 L\sigma_\ell^2}{n_q}\\ &= \frac{2(F(y_1)-F^*)}{A_Q}+\frac{\eta_\ell L\sigma_\ell^2}{B}.\end{aligned}$$

□

C.6.2 Empirical Evaluation

In Table 7, we compare LR-Norm against two other weighting schemes: *Example Weight* where the weight is the number of training examples for each client, and *Uniform Weight* where all clients have weight of 1. We see that LR-Norm performs competitively on CelebA. For CelebA, all weighting schemes, Uniform, Example, and LR-Norm perform similarly. This is because all clients in CelebA have one batch of data and number of examples per client is fairly centered around the mean. On the other hand, LR-Norm significantly outperforms Example Weight and Uniform Weight on Sent140. LR-Norm is beneficial when there is a high degree of data imbalance across clients, as in Sent140. Sent140 is more representative of real world FL applications where there is a long tail in the number of examples and number of batches per client.

C.7 Learning Curves

In this section, we show the learning curves for each algorithm in Figures 9, 10, and 11. These figures demonstrate FedBuff's robustness to different staleness distributions. Synchronous FL algorithms, FedAvgM, FedAvg and FedProx, are unaffected by the change in staleness distribution because they simply wait for all clients in the round.

For both CelebA and Sent140, FedBuff with $K = 10$ can reach the target validation accuracy quicker than other values of K. At $K = 10$, FedBuff appears to have the optimal balance between speed and variance reduction.

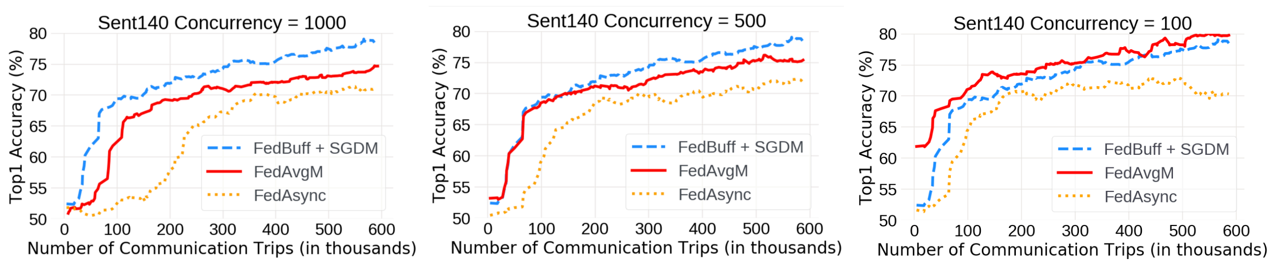


Figure 9: Training accuracy for FedAsync, FedAvgM and FedBuff on Sent140.

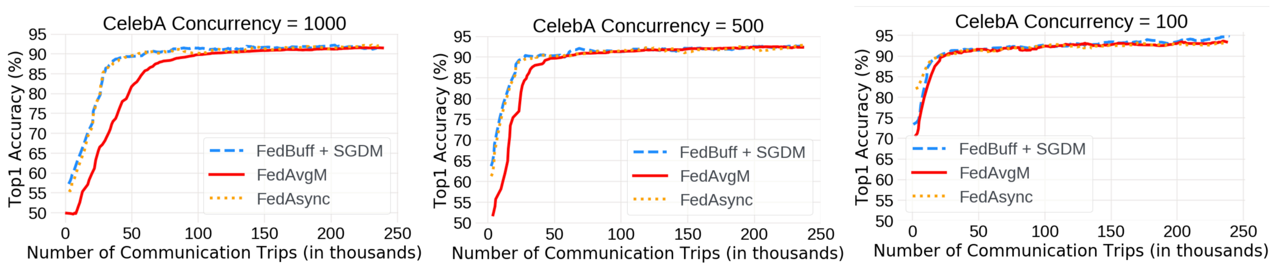


Figure 10: Training accuracy for FedAsync, FedAvgM and FedBuff on CelebA.

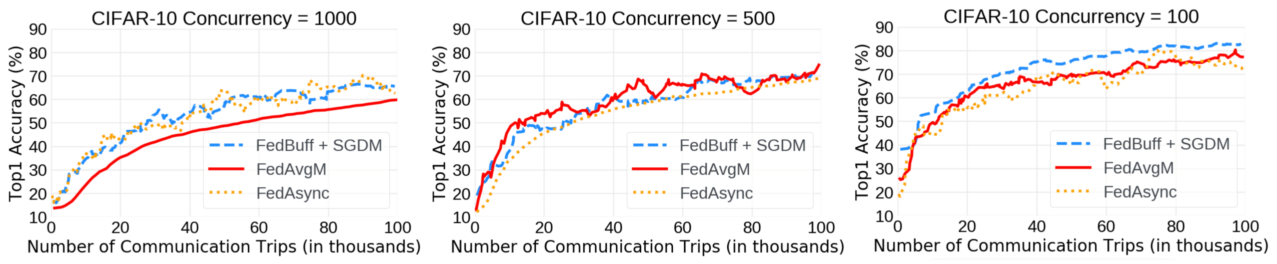


Figure 11: Training accuracy for FedAsync, FedAvgM and FedBuff on CIFAR-10.

Table 8: Summary of notation

Description	Symbol
number of server updates, server update index	T, t
set of clients updates used in server update t	$\mathcal{S}^t$
number of clients, client index	m, i or k
number of local steps per round, round index	Q, q
server model after t steps	w^t
stochastic gradient at client i	$g_i(w;\zeta_i) := g_i(w)$
local learning rate	η_l
global learning rate	η_g
number of clients in update	K
local and global gradient variance	σ_ℓ^2, σ_g^2
delay/staleness of client i's model update for the tth server update	$\tau_i(t)$
maximum staleness for buffer of size K	$\tau_{\max,K}$

D Proof of Convergence Rate

In this appendix, we prove the main convergence result for FedBuff. A summary of the notation used is provided in Table 8.

Observe that FedBuff updates can be described succinctly as

$$\begin{aligned} w^{t+1} &= w^t + \eta_g \overline{\Delta}^t \\ &= w^t + \eta_g \frac{1}{K} \sum_{k\in\mathcal{S}^t} \left(-\eta_\ell \sum_{q=1}^{Q} g_k(y_{k,q}^{t-\tau_k(t)}) \right), \end{aligned}$$

where $\mathcal{S}^t$ denotes the set of clients that contribute to the t'th server update, and $\tau_k(t) \geq 1$ is the staleness of an update contributed by client k to the t'th server update. Specifically, when $k \in \mathcal{S}^t$, the update returned by client k was computed by starting from $w^{t-\tau_k(t)}$ and performing Q local gradient steps. When $\tau_k(t) = 1$ there is no staleness in the update, and more generally $\tau_k(t) > 1$ corresponds to some staleness; i.e., $t - \tau_k(t)$ server updates have taken place between when the client last pulled a model from the server and when the client's update is being incorporated at the server.

In addition to the assumptions stated in Section 4, in the proof below we assume that $\mathcal{S}^t$ is a uniform subset $[n]$; i.e., in any given round any client is equally likely to contribute. This can be justified in practice as follows. To avoid having any client contribute more than once to any update, after the client returns an update contributing to $\overline{\Delta}^t$, the server can only sample that client after the server has performed another update.

We first state a useful lemma.

Lemma 1. $\mathbb{E}\Big[\, \|g_k\|^2 \Big] \leq 3(\sigma_\ell^2 + \sigma_g^2 + G)$, *where the total expectation* $\mathbb{E}[\cdot]$ *is evaluated over the randomness with respect to client participation and the stochastic gradient taken by a client.*

Proof. From the law of total expectation we have $\mathbb{E} = \mathbb{E}_{k\sim[m]}\mathbb{E}_{\zeta_k|k}$. Hence,

$$\begin{aligned} \mathbb{E}\Big[\, \|g_k(w)\|^2 \Big] &= \mathbb{E}_{k\sim[m]}\mathbb{E}_{g|k}\Big[\, \|g_k(w) - \nabla F_k(w) + \nabla F_k(w) - \nabla f(w) + \nabla f(w)\|^2 \Big] \\ &\leq 3\mathbb{E}_{k\sim[m]}\mathbb{E}_{g|k}\Big[\, \|g_k(w) - \nabla F_k(w)\|^2 + \|\nabla F_k(w) - \nabla f(w)\|^2 + \|\nabla f(w)\|^2 \Big] \qquad (6) \\ &= 3(\sigma_\ell^2 + \sigma_g^2 + G) \end{aligned}$$

□

D.1 Proof of Theorem 1

Theorem 2. *Let $\eta_\ell^{(q)}$ be the local learning rate of client SGD in the q-th step, and define $\alpha(Q) := \sum_{q=0}^{Q-1} \eta_\ell^{(q)}$, $\beta(Q) := \sum_{q=0}^{Q-1} (\eta_\ell^{(q)})^2$. Choosing $\eta_g \eta_\ell^{(q)} Q \leq \frac{1}{L}$ for all local steps $q = 0, \cdots, Q-1$, the global model iterates in Algorithm 1 achieves the following ergodic convergence rate*

$$\frac{1}{T}\sum_{t=0}^{T-1} \left\|\nabla f(w^t)\right\|^2 \leq \frac{2\Big(f(w^0) - f(w^*)\Big)}{\eta_g \alpha(Q) T} + 3L^2 Q\beta(Q)\Big(\eta_g^2 \tau_{\max,K}^2 + 1\Big)\Big(\sigma_\ell^2 + \sigma_g^2 + G\Big) + \frac{L}{2}\frac{\eta_g \beta(Q)}{\alpha(Q)}\sigma_\ell^2 \tag{7}$$

Proof. By L-smoothness assumption,

$$\begin{aligned} f(w^{t+1}) &\leq f(w^t) - \eta_g \langle \nabla f(w^t), \overline{\Delta}^t\rangle + \frac{L\eta_g^2}{2}\left\|\overline{\Delta}^t\right\|^2 \\ &\leq f(w^t) \underbrace{- \frac{\eta_g}{K}\sum_{k\in\mathcal{S}_t}\Big\langle \nabla f(w^t), \Delta_k^{t-\tau_k}\Big\rangle}_{T_1} + \underbrace{\frac{L\eta_g^2}{2K^2}\left\|\sum_{k\in\mathcal{S}_t}\Delta_k^{t-\tau_k}\right\|^2}_{T_2} \end{aligned} \tag{8}$$

where $\Delta_k^{t-\tau_k}$ is the client delta which is trained from using the global model after $t-\tau_k$ updates as initialization. We will next derive the upper bounds on T_1 and T_2.

$$T_1 = -\frac{\eta_g}{K}\sum_{k\in\mathcal{S}_t}\Big\langle \nabla f(w^t), \sum_{q=0}^{Q-1}\eta_\ell^{(q)} g_k(y_{k,q}^{t-\tau_k})\Big\rangle = -\frac{\eta_g}{K}\sum_{k\in\mathcal{S}_t}\sum_{q=0}^{Q-1}\eta_\ell^{(q)}\Big\langle \nabla f(w^t), g_k(y_{k,q}^{t-\tau_k})\Big\rangle \tag{9}$$

Using conditional expectation, the expectation operator can be written as

$$\mathbb{E}[\cdot] := \mathbb{E}_{\mathcal{H}}\mathbb{E}_{i\sim[m]}\mathbb{E}_{g_i|i,\mathcal{H}}[\cdot]$$

where $\mathbb{E}_{\mathcal{H}}$ is the expectation over the history of the iterates, $\mathbb{E}_{i\sim[m]}$ is evaluated over the randomness over the distribution of clients $i\sim[m]$ checking in at time-step t, and the inner expectation operates over the stochastic gradient of one step on a client. Hence, following unbiasedness,

$$\begin{aligned} \mathbb{E}[T_1] =& -\mathbb{E}\left[\frac{\eta_g}{K}\sum_{k\in\mathcal{S}_t}\sum_{q=0}^{Q-1}\eta_\ell^{(q)}\Big\langle \nabla f(w^t), g_k(y_{k,q}^{t-\tau_k})\Big\rangle\right] \\ =& -\eta_g\mathbb{E}_{\mathcal{H}}\left[\frac{1}{m}\sum_{i=1}^{m}\sum_{q=0}^{Q-1}\eta_\ell^{(q)}\mathbb{E}_{g_i|i\sim[m]}\Big\langle \nabla f(w^t), g_i(y_{i,q}^{t-\tau_i})\Big\rangle\right] \\ =& -\frac{\eta_g}{m}\mathbb{E}_{\mathcal{H}}\left[\sum_{i=1}^{m}\sum_{q=0}^{Q-1}\eta_\ell^{(q)}\Big\langle \nabla f(w^t), \nabla F_i(y_{i,q}^{t-\tau_i})\Big\rangle\right] \\ =& -\eta_g\mathbb{E}_{\mathcal{H}}\left[\sum_{q=0}^{Q-1}\eta_\ell^{(q)}\Big\langle \nabla f(w^t), \frac{1}{m}\sum_{i=1}^{m}\nabla F_i(y_{i,q}^{t-\tau_i})\Big\rangle\right] \end{aligned}$$

From the identity

$$\langle a, b\rangle = \frac{1}{2}(\|a\|^2 + \|b\|^2 - \|a-b\|^2)$$

we have

$$\begin{aligned} \mathbb{E}[T_1] = -\frac{\eta_g}{2}\left(\sum_{q=0}^{Q-1}\eta_\ell^{(q)}\right)\left\|\nabla f(w^t)\right\|^2 + \sum_{q=0}^{Q-1}\frac{\eta_g\eta_\ell^{(q)}}{2}\Big(-\mathbb{E}_{\mathcal{H}}\left\|\frac{1}{m}\sum_{i=1}^{m}\nabla F_i(y_{i,q}^{t-\tau_i})\right\|^2 \\ + \mathbb{E}_{\mathcal{H}}\underbrace{\left\|\nabla f(w^t) - \frac{1}{m}\sum_{i=1}^{m}\nabla F_i(y_{i,q}^{t-\tau_i})\right\|^2}_{T_3}\Big) \end{aligned} \tag{10}$$

Now for T_3, from the definition $f(w^t)$,

$$\begin{aligned}\mathbb{E}_{\mathcal{H}}[T_3] &= \mathbb{E}_{\mathcal{H}}\left\|\frac{1}{m}\sum_{i=1}^{m}\nabla F_i(w^t) - \frac{1}{m}\sum_{i=1}^{m}\nabla F_i(y_{i,q}^{t-\tau_i})\right\|^2 \\ &\leq \frac{1}{m}\sum_{i=1}^{m}\mathbb{E}_{\mathcal{H}}\left\|\nabla F_i(w^t) - \nabla F_i(y_{i,q}^{t-\tau_i})\right\|^2\end{aligned} \tag{11}$$

Further, by telescoping, T_3 can be decomposed as

$$\begin{aligned}\mathbb{E}[T_3] &= \frac{1}{m}\sum_{i=1}^{m}\mathbb{E}_{\mathcal{H}}\left\|\nabla F_i(w^t) - \nabla F_i(w^{t-\tau_i}) + \nabla F_i(w^{t-\tau_i}) - \nabla F_i(y_{i,q}^{t-\tau_i})\right\|^2 \\ &\leq \frac{2}{m}\sum_{i=1}^{m}\mathbb{E}_{\mathcal{H}}\Big(\underbrace{\left\|\nabla F_i(w^t) - \nabla F_i(w^{t-\tau_i})\right\|^2}_{\text{staleness}} + \underbrace{\left\|\nabla F_i(w^{t-\tau_i}) - \nabla F_i(y_{i,q}^{t-\tau_i})\right\|^2}_{\text{local drift}}\Big) \\ &\leq \frac{2}{m}\sum_{i=1}^{m}\Big(L^2\mathbb{E}_{\mathcal{H}}\left\|w^t - w^{t-\tau_i}\right\|^2 + L^2\mathbb{E}_{\mathcal{H}}\left\|w^{t-\tau_i} - y_{i,q}^{t-\tau_i}\right\|^2\Big)\end{aligned} \tag{12}$$

The upper bound on T_3 can be understood as sums of bounds on the effect of staleness and local drift during client training, and local variance induced by client-side SGD. Further, we need to produce upper bound on the staleness of initial model from which the client models are trained.

$$\begin{aligned}\left\|w^t - w^{t-\tau_i}\right\|^2 = \left\|\sum_{\rho=t-\tau_i}^{t-1}(w^{\rho+1} - w_\rho)\right\|^2 &= \left\|\sum_{\rho=t-\tau_i}^{t-1}\frac{\eta_g}{K}\sum_{j_\rho\in\mathcal{S}_\rho}\Delta_{j_\rho}^{\rho}\right\|^2 \\ &= \frac{\eta_g^2}{K^2}\left\|\sum_{\rho=t-\tau_i}^{t-1}\sum_{j_\rho\in\mathcal{S}_\rho}\sum_{l=0}^{Q-1}\eta_\ell^{(l)}g_{j_\rho}(y_{j_\rho,l}^{\rho})\right\|^2\end{aligned} \tag{13}$$

Taking expectation in terms of $\mathcal{H}$,

$$\begin{aligned}\mathbb{E}_{\mathcal{H}}\left\|w^t - w^{t-\tau_i}\right\|^2 &\leq \frac{\eta_g^2 Q\tau_i}{K}\sum_{\rho=t-\tau_i}^{t-1}\sum_{j_\rho\in\mathcal{S}_\rho}\sum_{l=0}^{Q-1}(\eta_\ell^{(l)})^2\mathbb{E}\left\|g_{j_\rho}(y_{j_\rho,l}^{\rho})\right\|^2 \\ &\leq 3\eta_g^2 Q\max_{\tau_i}\tau_i^2\Big(\sum_{l=0}^{Q-1}(\eta_\ell^{(l)})^2\Big)\Big(\sigma_\ell^2 + \sigma_g^2 + G\Big) \\ &\leq 3\eta_g^2 Q\tau_{\max,K}^2\Big(\sum_{l=0}^{Q-1}(\eta_\ell^{(l)})^2\Big)\Big(\sigma_\ell^2 + \sigma_g^2 + G\Big)\end{aligned} \tag{14}$$

where the last inequality follows from the assumption on maximal delay and apply Lemma 1. Similarly, the local drift term can be upper-bounded by

$$\mathbb{E}\left\|w^{t-\tau_i} - y_{i,q}^{t-\tau_i}\right\|^2 = \mathbb{E}\left\|y_{i,0}^{t-\tau_i} - y_{i,q}^{t-\tau_i}\right\|^2 \leq \mathbb{E}\left\|\sum_{l=0}^{q-1}\eta_\ell^{(l)}g_i(y_{i,l}^{t-\tau_i})\right\|^2 \leq 3q\left(\sum_{l=0}^{q-1}(\eta_\ell^{(l)})^2\right)\left(\sigma_\ell^2 + \sigma_g^2 + G\right) \tag{15}$$

Thus, the upper bound on T_3 becomes:

$$\begin{aligned}\mathbb{E}[T_3] &\leq 6\Bigg(L^2\eta_g^2 Q\tau_{\max,K}^2\Big(\sum_{i=0}^{Q-1}(\eta_\ell^{(i)})^2\Big)\Big(\sigma_\ell^2+\sigma_g^2+G\Big)+L^2q\left(\sum_{i=0}^{q-1}(\eta_\ell^{(i)})^2\right)\Big(\sigma_\ell^2+\sigma_g^2+G\Big)\Bigg)\\ &\leq 6L^2\Big(\sum_{i=0}^{Q-1}(\eta_\ell^{(i)})^2\Big)(\eta_g^2 Q\tau_{\max,K}^2+q)\Big(\sigma_\ell^2+\sigma_g^2+G\Big)\\ &\leq 6L^2Q\Big(\sum_{i=0}^{Q-1}(\eta_\ell^{(i)})^2\Big)(\eta_g^2\tau_{\max,K}^2+1)\Big(\sigma_\ell^2+\sigma_g^2+G\Big)\end{aligned} \tag{16}$$

Inserting the upper bound on T_3 into (10), we have,

$$\mathbb{E}[T_1] \leq -\frac{\eta_g}{2}\left(\sum_{q=0}^{Q-1}\eta_\ell^{(q)}\right)\left\|\nabla f(w^t)\right\|^2+\sum_{q=0}^{Q-1}\frac{\eta_g\eta_\ell^{(q)}}{2}\mathbb{E}[T_3]-\sum_{q=0}^{Q-1}\frac{\eta_g\eta_\ell^{(q)}}{2}\mathbb{E}_{\mathcal{H}}\left\|\frac{1}{m}\sum_{i=1}^{m}\nabla F_i(y_{i,q}^{t-\tau_i})\right\|^2 \tag{17}$$

Let $\alpha(Q) := \sum_{q=0}^{Q-1}\eta_\ell^{(q)}$, $\beta(Q) := \sum_{q=0}^{Q-1}(\eta_\ell^{(q)})^2$,

$$\mathbb{E}[T_1] \leq -\frac{\eta_g\alpha(Q)}{2}\left\|\nabla f(w^t)\right\|^2+3\eta_g L^2Q\alpha(Q)\beta(Q)\Big(\eta_g^2\tau_{\max,K}^2+1\Big)\Big(\sigma_\ell^2+\sigma_g^2+G\Big)\underbrace{-\sum_{q=0}^{Q-1}\frac{\eta_g\eta_\ell^{(q)}}{2}\mathbb{E}_{\mathcal{H}}\left\|\frac{1}{m}\sum_{i=1}^{m}\nabla F_i(y_{i,q}^{t-\tau_i})\right\|^2}_{T_4} \tag{18}$$

To derive the upperbound on the R.H.S. of (8), we now need to upper bound $\mathbb{E}[T_2]$.

$$\begin{aligned}\mathbb{E}[T_2] &= \mathbb{E}\left[\frac{L\eta_g^2}{2K^2}\left\|\sum_{k\in\mathcal{S}_t}\sum_{q=0}^{Q-1}\eta_\ell^{(q)}g_k(y_{k,q}^{t-\tau_k})\right\|^2\right]\\ &= \mathbb{E}\left[\frac{L\eta_g^2}{2K^2}\left\|\sum_{k\in\mathcal{S}_t}\sum_{q=0}^{Q-1}\eta_\ell^{(q)}\Big(g_k(y_{k,q}^{t-\tau_k})-\nabla F_k(y_{k,q}^{t-\tau_k})\Big)+\sum_{k\in\mathcal{S}_t}\sum_{q=0}^{Q-1}\eta_\ell^{(q)}\nabla F_k(y_{k,q}^{t-\tau_k})\right\|^2\right]\\ &\overset{\text{(A.)}}{=} \frac{L\eta_g^2}{2K^2}\mathbb{E}\left\|\sum_{k\in\mathcal{S}_t}\sum_{q=0}^{Q-1}\eta_\ell^{(q)}\Big(g_k(y_{k,q}^{t-\tau_k})-\nabla F_k(y_{k,q}^{t-\tau_k})\Big)\right\|^2+\frac{L\eta_g^2}{2K^2}\mathbb{E}\left\|\sum_{k\in\mathcal{S}_t}\sum_{q=0}^{Q-1}\eta_\ell^{(q)}\nabla F_k(y_{k,q}^{t-\tau_k})\right\|^2\\ &\overset{\text{(B.)}}{=} \frac{L\eta_g^2}{2}\sum_{k\in\mathcal{S}_t}\sum_{q=0}^{Q-1}(\eta_\ell^{(q)})^2\mathbb{E}\left\|\Big(g_k(y_{k,q}^{t-\tau_k})-\nabla F_k(y_{k,q}^{t-\tau_k})\Big)\right\|^2+\frac{L\eta_g^2}{2K^2}\mathbb{E}\left\|\sum_{k\in\mathcal{S}_t}\sum_{q=0}^{Q-1}\eta_\ell^{(q)}\nabla F_k(y_{k,q}^{t-\tau_k})\right\|^2\\ &\leq \frac{L\eta_g^2\beta(Q)\sigma_\ell^2}{2}+\frac{LQ\eta_g^2}{2K}\sum_{k\in\mathcal{S}_t}\sum_{q=0}^{Q-1}(\eta_\ell^{(q)})^2\mathbb{E}_{\mathcal{H}}\mathbb{E}_{k\sim[m]|\mathcal{H}}\left\|\nabla F_k(y_{k,q}^{t-\tau_k})\right\|^2\\ &= \frac{L\eta_g^2\beta(Q)\sigma_\ell^2}{2}+\frac{LQ\eta_g^2}{2K}\sum_{k\in\mathcal{S}_t}\sum_{q=0}^{Q-1}(\eta_\ell^{(q)})^2\mathbb{E}_{\mathcal{H}}\left[\frac{1}{m}\sum_{i=1}^{m}\left\|\nabla F_i(y_{i,q}^{t-\tau_i})\right\|^2\right]\\ &= \frac{L\eta_g^2\beta(Q)\sigma_\ell^2}{2}+\underbrace{\frac{LQ\eta_g^2}{2m}\sum_{q=0}^{Q-1}\sum_{i=1}^{m}(\eta_\ell^{(q)})^2\mathbb{E}_{\mathcal{H}}\left[\left\|\nabla F_i(y_{i,q}^{t-\tau_i})\right\|^2\right]}_{T_5}\end{aligned} \tag{19}$$

where (A.) follows the unbiasedness of g_k, and (B.) follows from the fact that $g_k - \nabla F_k$ are independent and unbiased for $k \sim [m]$. To produce an upperbound on $\mathbb{E}[T_1 + T2]$, we need to make sure $T_4 + T_5 \leq 0$.

$$\begin{aligned}
&\big(T_4 + T_5\big) \\
&= -\sum_{q=0}^{Q-1} \frac{\eta_g \eta_\ell^{(q)}}{2} \mathbb{E}_{\mathcal{H}} \left\| \frac{1}{m} \sum_{i=1}^{m} \nabla F_i(y_{i,q}^{t-\tau_i}) \right\|^2 + \frac{LQ\eta_g^2}{2m} \sum_{q=0}^{Q-1} \sum_{i=1}^{m} (\eta_\ell^{(q)})^2 \mathbb{E}_{\mathcal{H}} \left\| \nabla F_i(y_{i,q}^{t-\tau_i}) \right\|^2 \\
&= -\sum_{q=0}^{Q-1} \sum_{i=1}^{m} \frac{\eta_g \eta_\ell^{(q)}}{2m} \mathbb{E}_{\mathcal{H}} \left\| \nabla F_i(y_{i,q}^{t-\tau_i}) \right\|^2 + \frac{LQ\eta_g^2}{2m} \sum_{q=0}^{Q-1} \sum_{i=1}^{m} (\eta_\ell^{(q)})^2 \mathbb{E}_{\mathcal{H}} \left\| \nabla F_i(y_{i,q}^{t-\tau_i}) \right\|^2 \\
&= \sum_{q=0}^{Q-1} \sum_{i=1}^{m} \Big(-\frac{\eta_g \eta_\ell^{(q)}}{2m} + \frac{LQ\eta_g^2 (\eta_\ell^{(q)})^2}{2m} \Big) \mathbb{E}_{\mathcal{H}} \left\| \nabla F_i(y_{i,q}^{t-\tau_i}) \right\|^2
\end{aligned} \tag{20}$$

To ensure $T_4 + T_5 \leq 0$, it is sufficient to choose $\eta_g \eta_\ell^{(q)} Q \leq \frac{1}{L}$ for all local steps $q = 0, \cdots, Q-1$.

Now, plugging (18), (19) and (20) into (8),

$$\mathbb{E}[f(w^{t+1})] \leq \mathbb{E}[f(w^t)] - \frac{\eta_g \alpha(Q)}{2} \left\| \nabla f(w^t) \right\|^2 + 3\eta_g L^2 Q \alpha(Q) \beta(Q) \Big(\eta_g^2 \tau_{\max,K}^2 + 1 \Big) \Big(\sigma_\ell^2 + \sigma_g^2 + G \Big) + \frac{L}{2} \eta_g^2 \beta(Q) \sigma_\ell^2 \tag{21}$$

Summing up t from 1 to T and rearrange, yields

$$\begin{aligned}
\sum_{t=0}^{T-1} \eta_g \alpha(Q) \left\| \nabla f(w^t) \right\|^2 &\leq \sum_{t=0}^{T-1} 2\Big(\mathbb{E}[f(w^t)] - \mathbb{E}[f(w^{t+1})] \Big) + 3 \sum_{t=0}^{T-1} \eta_g L^2 Q \alpha(Q) \beta(Q) \Big(\eta_g^2 \tau_{\max,K}^2 + 1 \Big) \Big(\sigma_\ell^2 + \sigma_g^2 + G \Big) \\
&\quad + \frac{L}{2} \eta_g^2 \beta(Q) \sigma_\ell^2 \\
&\leq 2\Big(f(w^0) - f(w^*) \Big) + 3 \sum_{t=0}^{T-1} \eta_g L^2 \alpha(Q) \beta(Q) \Big(\eta_g^2 \tau_{\max,K}^2 + Q \Big) \Big(\sigma_\ell^2 + \sigma_g^2 + G \Big) + \frac{L}{2} \eta_g^2 \beta(Q) \sigma_\ell^2
\end{aligned} \tag{22}$$

Thus we have

$$\frac{1}{T} \sum_{t=0}^{T-1} \left\| \nabla f(w^t) \right\|^2 \leq \frac{2\Big(f(w^0) - f(w^*) \Big)}{\eta_g \alpha(Q) T} + 3L^2 Q \beta(Q) \Big(\eta_g^2 \tau_{\max,K}^2 + 1 \Big) \Big(\sigma_\ell^2 + \sigma_g^2 + G \Big) + \frac{L}{2} \frac{\eta_g \beta(Q)}{\alpha(Q)} \sigma_\ell^2 \tag{23}$$

□