Federated Learning with Buffered Asynchronous Aggregation

John Nguyen [1] **Kshitiz Malik** [1] **Hongyuan Zhan** [1] **Ashkan Yousefpour** [1] **Michael Rabbat** [1] **Mani Malek** [1] **Dzmitry Huba** [1]

Abstract

Federated Learning (FL) trains a shared model across distributed devices while keeping the training data on the devices. Most FL schemes are synchronous: they perform a synchronized aggregation of model updates from individual devices. Synchronous training can be slow because of late-arriving devices (*stragglers*). On the other hand, completely asynchronous training makes FL less private because of incompatibility with secure aggregation. In this work, we propose a model aggregation scheme, FedBuff, that combines the best properties of synchronous and asynchronous FL. Similar to synchronous FL, FedBuff is compatible with secure aggregation. Similar to asynchronous FL, FedBuff is robust to stragglers. In FedBuff, clients trains asynchronously and send updates to the server. The server aggregates client updates in a private buffer until K updates have been received, at which point a server model update is immediately performed. We provide theoretical convergence guarantees for FedBuff in a non-convex setting. Empirically, FedBuff converges up to $3.8\times$ faster than previous proposals for synchronous FL (e.g., FedAvgM), and up to $2.5\times$ faster than previous proposals for asynchronous FL (e.g., FedAsync). We show that FedBuff is robust to different staleness distributions and is more scalable than synchronous FL.

1. Introduction

Federated Learning (FL) trains a shared model across distributed clients while training data stays on the client devices. The most common FL scenario is cross-device FL, where typically a large number of client devices participate in the training with a single server. Since the number of client devices in cross-device FL is very large (Kairouz et al., 2019), designing a scheme for FL comes with several challenges.

Challenge 1: Scalability. Practical FL systems often train over hundreds or thousands of clients in parallel (Hard et al., 2019). Hence, FL training algorithms must be scalable and data-efficient — they should be able to exploit parallelism across clients to speed up training.

Challenge 2: Device heterogeneity and data imbalance. Client devices have heterogeneous compute power, differing by more than an order of magnitude (Wu et al., 2019). Moreover, client devices can have vastly different amounts of training data (Caldas et al., 2018). Therefore, training time across clients has a large variance, driven by differences in device capabilities and the amount of training data. Since FL algorithms typically make one or more complete passes over a client's data per round, variance in training data per client further increases variance in training time.

Challenge 3: Privacy. Outside of model convergence, privacy is a big consideration when designing a protocol for FL. Secure Aggregation and differential privacy make FL more private, and more robust to attacks such as model inversion, model poisoning, and training data poisoning. Secure Aggregation (Bonawitz et al., 2016; Karl et al., 2020; Emanuel) ensures that individual client updates are aggregated together before they are visible to the server. For differential privacy (DP), the DP-SGD algorithm (Abadi et al., 2016) can be used to provide user-level DP guarantees in FL (McMahan et al., 2018). When combined with Secure Aggregation, user-level DP guarantees can achieve superior privacy-utility trade-off.

While there are many other challenges in FL (Kairouz et al., 2019), this paper focuses on the three mentioned above.

This paper proposes and analyzes FedBuff, a novel asynchronous federated optimization framework using buffered asynchronous aggregation. By using asynchronous updates, we demonstrate that FedBuff can scale efficiently to 1000's of concurrent users, in addition to alleviating the straggler issue. Moreover, by aggregating client updates in a secure buffer before applying them at the server, FedBuff is directly

[1]Facebook Inc., Menlo Park, CA, USA. Correspondence to: John Nguyen <ngjhn@fb.com>, Kshitiz Malik <kmalik2@fb.com>.

This work was presented at the International Workshop on Federated Learning for User Privacy and Data Confidentiality in Conjunction with ICML 2021 (FL-ICML'21). This workshop does not have official proceedings and this paper is non-archival.

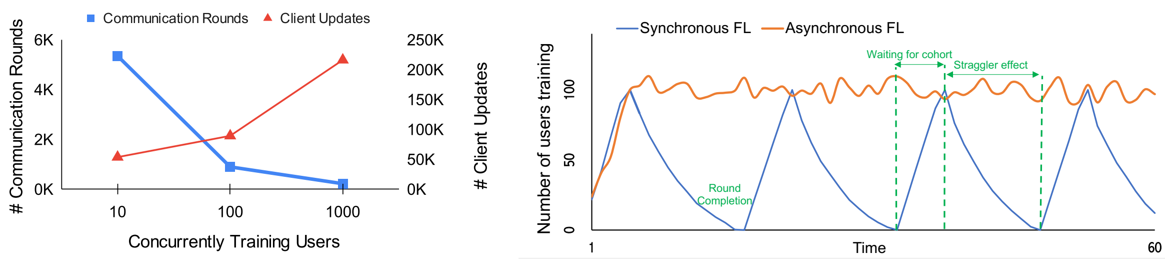

Figure 1. (left) Number of communication rounds and number of client updates required by FedAvgM (Hsu et al., 2019), a synchronous FL algorithm, to reach a target accuracy on the Sent140 (Caldas et al., 2018). This figure demonstrates the diminishing returns from increasing concurrency for synchronous FL in terms of training time. Increasing concurrency from 100 to 1000 decreases the number of communication rounds by less than a factor of 2 and more than doubles the total number of client updates required. This is similar to observations in conventional SGD training, where increasing the batch size eventually gives diminishing returns (Goyal et al., 2017; Shallue et al., 2018; Ott et al., 2018; You et al., 2017; 2018; 2019). As concurrency increases, synchronous FL becomes less data-efficient. The server learning rate and hyperparameters are tuned separately for each number of concurrently training users, and the number of client updates does not include any overhead related to over-selection. (right) Training progress for asynchronous and synchronous FL, and the associated delays. The y-axis in the figure shows the number of clients actively computing updates at any given point in time. 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.

compatible with existing secure aggregation and privacy techniques, unlike previous asynchronous FL proposals.

Synchronous FL. Most works in FL have focused on synchronous methods, as they are easier to analyze and debug. Synchronous FL methods are also better suited for privacy — the amount of noise added for user-level DP decreases as the number of client updates increases(Kairouz et al., 2021; McMahan et al., 2018). However, synchronous FL methods are prone to stragglers. They proceed at the pace of the slowest client; a round has to wait for all the participating clients to finish. Additionally, the more heterogeneous the clients, the worse the straggler problem.

Although the straggler problem in Synchronous FL has been well-studied (Xie et al., 2019; van Dijk et al., 2020; Chai et al., 2020; Chen et al., 2019; Wu et al., 2020), the scalability of Synchronous FL is perhaps an even bigger concern that has not received as much attention. In practical FL systems at internet-scale (Bonawitz et al., 2019), only a small fraction of clients train in parallel at any given time. An important parameter in FL is the *concurrency*: the number of clients training concurrently (sometimes also called *users-per-round* in synchronous FL). In synchronous FL, the optimal server learning rate generally increases with concurrency; aggregating over more users has a variance-reducing effect, enabling the server to take longer steps. Consequently, higher concurrency reduces the number of rounds needed to reach a target accuracy, similar to in large-batch training (Goyal et al., 2017; Ott et al., 2018; You et al., 2019; 2018; 2017; Shallue et al., 2018). However, to have stable, convergent training dynamics, the server learning rate cannot be increased indefinitely. Eventually it saturates, resulting in a sub-linear speed-up as illustrated in Figure 1(left). As a result, practical synchronous FL systems cannot accelerate training through parallelism beyond a few hundred clients, and synchronous FL training is often an order of magnitude slower than conventional server training (Bonawitz et al., 2019).

Asynchronous FL. Asynchronous methods are a good match for the FL setting, where device heterogeneity and clients dropping out mid-round amplify the straggler problem. Asynchronous FL schemes solve the straggler problem by incorporating a client update as soon as it is available rather than over-selecting clients and then discarding their updates. Asynchronous methods have their challenges, including staleness of client updates which complicates analysis, and non-determinism, which can complicate debugging.

In pure asynchronous FL methods (Xie et al., 2019), every client update results in a server model update. This has implications for privacy in FL. When every client update forces a server update, Secure Aggregation cannot be used to make FL more private; Secure Aggregation's benefit is in hiding updates in an aggregate. Additionally, providing user-level DP in asynchronous FL is only feasible with local differential privacy (LDP), where the client clips and adds noise before sending the update to the server. LDP for high dimensional data has been criticized for poor privacy-utility trade-off (Erlingsson et al., 2020; Bittau et al., 2017).

Our proposal: FedBuff. In FedBuff, clients train and communicate asynchronously with the server. Unlike other asynchronous methods, though, the client updates are aggregated in a secure buffer until K updates have been selected, at which point a server model update is performed. The number of client updates required to trigger a server model update, K, is a tunable parameter. We show in Section 3.1 that small values of K (e.g., 10) result in fast and data-efficient training, up to 3.8× faster than synchronous FL methods. Larger values of K (e.g. 1000) require less noise with DP, at the cost of slower training convergence. Unlike other asynchronous FL proposals (Xie et al., 2019), FedBuff is compatible with Secure Aggregation, especially techniques that rely on a TEE (Karl et al., 2020; Emanuel).

Contributions. We highlight the main contributions here.

• We propose FedBuff, a novel asynchronous FL training scheme, to simultaneously achieve scalability and compatibility with Secure Aggregation.

• We provide convergence analysis for general non-convex settings. When FedBuff is configured to produce a server update for every K client updates in the buffer and client training is triggered asynchronously to take Q SGD steps, FedBuff requires $\mathcal{O}\left(1/(\epsilon^2 KQ)\right)$ server iterations to reach ϵ accuracy (Appendix D).

• We show empirically that FedBuff is up to 3.8 times faster than competing synchronous FL algorithms, even without penalizing synchronous FL algorithms for stragglers. We also demonstrate that FedBuff is up to 2.5 times faster than its closest competing asynchronous FL algorithm, FedAsync (Xie et al., 2019).

• We demonstrate that FedBuff's speed up over competing synchronous and asynchronous FL methods are robust across various staleness distributions and datasets. We show that staleness distribution in a real-world setup; running FL across millions of client devices closely approximates a half-normal distribution. To the best of our knowledge, we are the first to analyze empirical staleness distribution from a production asynchronous FL system.

2. 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 client' data well on (weighted) average. In FL, F_i is only accessible by device i. For simplicity, in this paper we focus on the unweighted setting, $p_i = 1$ for all i, although our analysis can be easily extended to the more general case with non-uniform weights.

Algorithm 1 `FedBuff-server`

Input: global learning rate η_g, local learning rate η_ℓ, num. client SGD steps, buffer size K, model w^0
Output: FL-trained global model

repeat
  $c \leftarrow$ sample available clients ▷ async
  run `FedBuff-client`(w^t, η_ℓ, Q) on c ▷ async
  if receive client update **then**
    $\Delta_i \leftarrow$ received update from client
    $\overline{\Delta}^t \leftarrow \overline{\Delta}^t + \Delta_i$ ▷ inside secure aggregator
    $k \leftarrow k + 1$
  end if
  if $k == K$ **then**
    $w^{t+1} \leftarrow w^t - \eta_g \overline{\Delta}^t$
    $\overline{\Delta}^t \leftarrow 0, k \leftarrow 0, t \leftarrow t+1$ ▷ reset buffer
  end if
until Convergence

Algorithm 2 `FedBuff-client`

Input: server model w, local learning rate η_ℓ, number of client SGD steps Q
Output: client update Δ

$y_0 \leftarrow w$
for $q = 1 : Q$ **do**
  $y_q \leftarrow y_{q-1} - \eta_\ell g_q(y_{q-1})$
end for
$\Delta \leftarrow y_0 - y_q$
Send Δ to server

Synchronous FL methods need to aggregate and synchronize clients after each round. Hence, concurrency in synchronized FL is equal to the number of clients who participate in a given round. In asynchronous methods, concurrency is the number of clients in training at a given point in time. 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, a *buffer* is responsible for aggregating client updates, and a server update only takes place once K client updates have been aggregated, where K is a tunable parameter. It is important to note that, K is independent of concurrency — the extra degree of freedom introduced by the buffer allows the server to update more frequently than concurrency (as in synchronous FL). This allows FedBuff to achieve data efficiency at high concurrency while being compatible with Secure Aggregation. Theoretical convergence guarantees for FedBuff are provided in Appendix D.

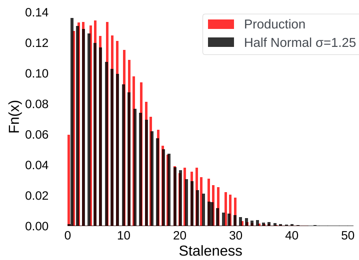

Figure 2. Staleness distribution observed in production when training over millions of real clients for FedBuff. Overlaid on this figure is the staleness distribution from our simulation when using a half-normal with $\sigma = 1.25$. Our observation that execution time on mobile clients exhibits a folded normal distribution is consistent with previous findings (Wu et al., 2019).

3. Experiments

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

Datasets, models, and tasks. We run experiments on two problems from the LEAF benchmark (Caldas et al., 2018), one text classification (binary sentiment analysis on Sent140 (Go et al., 2009)), and one image classification (multi-class classification with CelebA (Liu et al., 2015)). We use the standard non-iid client partitions from LEAF, and similar models. For Sent140 (Go et al., 2009), we train an LSTM classifier, where each Twitter account corresponds to a client. For CelebA (Liu et al., 2015), we train the same convolutional neural network classifier as LEAF, but with batch normalization layers replaced by group normalization layers (Wu & He, 2018) as suggested in (Hsieh et al., 2020). More details about datasets, models, and tasks are provided in Appendix C.1.

Experimental setup. We implement all algorithms in PyTorch (Paszke et al., 2017). For asynchronous training, we assume that clients arrive at a constant rate. Training duration of a client is sampled from a half-normal probability distribution. We chose this distribution because it best matches the staleness distribution observed in our production FL system, as illustrated in Figure 2. We also report results with two other staleness distributions (uniform, as used in (Xie et al., 2019), and exponential) below, demonstrating that FedBuff performance improvements are consistent across different staleness distributions. The best fit to the production data is a half-normal distribution with $\sigma = 1.25$.

Baselines. We compare FedBuff with three synchronous baselines, namely FedAvg (McMahan et al., 2016), FedProx (Li et al., 2018), FedAvgM (Hsu et al., 2019), and one asynchronous FL baseline, FedAsync (Xie et al., 2019). We focus on non-adaptive methods here since FedBuff currently uses non-adaptive updates; FedBuff could also be modified to use adaptive updates at the server, and we leave this to future work. For more details see Appendix C.2. Our implementation of FedBuff incorporates two practical improvements, staleness scaling and learning rate normalization, described in Appendix A.

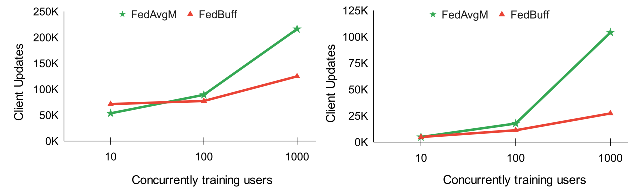

Figure 3. Number of client updates to reach target validation accuracy. At low concurrency, FedBuff and FedAvgM perform similarly. However, as concurrency increases, FedBuff outperforms FedAvgM by increasingly larger amounts. In contrast to FedAvgM, FedBuff's data-efficiency and communication-efficiency degrade less with concurrency.

Hyperparameters. For all algorithms, we run hyperparameter sweeps to tune learning rates η_ℓ and η_g, staleness exponent α, server momentum β, and the proximal term μ for FedProx. We fix $\beta = 0$ for FedAvg. Each client update entails running one local epoch with batch size $B = 32$. See Appendix C.3 for more on hyperparameter tuning.

Concurrency and K. In a practical FL deployment, 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 that are being compared. Recall the example in Figure 1 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.

Evaluation metrics. Synchronous FL algorithms are often evaluated by the number of communication rounds taken to converge to a target model accuracy. However, asynchronous methods do not have the same notion of rounds; clients join and leave asynchronously. For this reason, we compare synchronous and asynchronous methods by the number of client updates needed to reach a target accuracy. For CelebA the target is 90% top-1 validation accuracy, and for Sent140 the target is 69% classification accuracy. The number of client updates measures both the computation

Table 1. Number of client updates to reach target validation accuracy on CelebA and Sent140 (lower is better. Units = 1000 updates).Concurrency, (M), is the maximum number of training clients at any point in time. We set $M = 1000$ for all methods and $K = 10$ for FedBuff. We ran CelebA for 240k and Sent140 for 600k updates. > 600 indicates the target accuracy was not reached for Sent140.

Dataset	Target Accuracy	FedBuff	FedAsync	FedAvgM	FedAvg	FedProx
CelebA	90%	27.1	28.7 (1.1×)	104 (3.8×)	231 (8.5×)	228 (8.4×)
Sent140	69%	124.7	308.9 (2.5×)	216 (1.7×)	> 600	> 600

Table 2. Speed up of FedBuff over FedAvgM and FedAsync w.r.t number of client updates to reach target validation accuracy, for different staleness distributions. We set $M = 1000$ for all methods and $K = 10$ for FedBuff. FedBuff's speed up is consistent across staleness distributions.

Dataset	Staleness Distribution	Speedup over FedAvgM	Speedup over FedAsync
CelebA	Uniform	4.65×	1.64×
	Half-Normal	3.83×	1.06×
	Exponential	4.34×	1.07×
Sent140	Uniform	1.25×	1.17×
	Half-Normal	1.73×	2.48×
	Exponential	1.40×	1.96×

and communication required to reach the target accuracy.

Unless otherwise mentioned, the reported number of updates does not account for over-selection, commonly used in practical implementations of synchronous FL methods. Since we use the number of client updates to compare algorithms, synchronous methods are not penalized for stragglers — their behavior is independent of training time assumptions.

3.1. Results

Comparison of Methods. Table 1 shows the number of client updates needed to converge to the target accuracy on Sent140 and CelebA. For brevity, we only show results from FedBuff with K=10. See Appendices B.1 and C.4 for FedBuff results with other values of K and learning curves. Compared to the best synchronous method in the experiments (FedAvgM), FedBuff converges to target accuracy 1.7-3.8× faster. Compared to FedAsync, FedBuff converges to target accuracy 1.1-2.5× faster.

Robustness to Staleness Distributions. To analyze the sensitivity of FedBuff to different training time distributions, we compare FedBuff against other competing algorithms with different staleness distributions. Table 2 demonstrates that FedBuff is robust and FedBuff's speed up is consistent across staleness distributions.

Scalablility of FedBuff. Figure 3 shows that FedBuff scales much better to larger values of concurrency than FedAvgM, the best-performing synchronous algorithm in our experiments. FedBuff with K=10 scales better because it updates the server model more frequently than FedAvgM when concurrency is high. 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 simply be run at lower concurrency (i.e. 10). However, that leads to longer wall-clock training time because of less parallelism being exploited, as shown in Figure 1 (left). For synchronous 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. Results show that empirically, the benefits from frequently advancing the server model outweigh the cost of staleness in client updates.

To summarize, synchronous FL algorithms have only one degree of freedom: concurrency. Synchronous methods need higher concurrency in practice to speed up training time by increasing parallelism, but higher concurrency brings data inefficiency. FedBuff has two degrees of freedom, concurrency and K, the frequency of server model updates. High concurrency coupled with frequent server model updates (small values of K) result in extremely fast training.

Large values of K. We saw that FedBuff trains fast when run with small values of K relative to the concurrency. However, large values of K are useful when providing user-level DP guarantees because the noise added for user-level DP decreases with K (Kairouz et al., 2021; McMahan et al., 2018).

Next, we compare the training speed of FedBuff and synchronous training 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.

Some synchronous FL systems (Bonawitz et al., 2019) use over-selection, typically by 30%, to address stragglers. For example, if 1000 users are needed to produce a server model

Table 3. 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 Number of client updates: 1000 updates). Concurrency is the maximum number of training clients at any point in time. 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 updates

Dataset	Algorithm	Concurrency	Wall-Clock Time	Num Client Updates
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

update, 1300 users may be 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, at the cost of wasting some user compute and bandwidth.

Table 3 reports the wall-clock training time and number of client updates 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 2). 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.

4. Related Work

We compared FedBuff with three synchronous algorithms — FedProx, FedAvgM and FedAvg — in Section 3. FedProx (Li et al., 2018) improved upon the FedAvg (McMahan et al., 2016) algorithm by adding a proximal term μ to the local SGD solver. FedAvgM (Reddi et al., 2020) improves synchronous FL convergence by adding server momentum. Adaptive methods such as FedAdam have comparable performance to FedAvgM (Reddi et al., 2020).

Previous works (Xie et al., 2019; van Dijk et al., 2020; Chai et al., 2020; Chen et al., 2019; Wu et al., 2020) also proposed asynchronous FL methods. However, the methods proposed in those papers generally include aspects that make them impractical for internet-scale deployment, e.g., involving profiling client speed (Chai et al., 2020; Li et al., 2021), assuming clients have same speed (van Dijk et al., 2020), requiring information about participating clients (Wu et al., 2020), frequently broadcasting the server model updates to all participating clients (Chen et al., 2019), or being incompatible with secure aggregation (Xie et al., 2019).

Asynchronous stochastic optimization in shared-memory and distributed-memory systems has also been widely-studied (Bertsekas & 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). In this work, we show that in FL systems with a large number of clients, the source of speed-up is not only due to avoiding stragglers, but also achieving better data efficiency at high concurrency.

Many proposals aim 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 certain batch size in synchronized 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.

5. Conclusions

In this paper, we propose FedBuff, an asynchronous training scheme for FL that incorporates a private buffer. Compared to synchronous FL proposals, FedBuff is more scalable to large values of concurrency because it can update the server model more frequently. Additionally, FedBuff is robust to stragglers. Compared to asynchronous FL proposals, FedBuff is more private as it is compatible with Secure Aggregation. We analyze the convergence behavior of FedBuff in the non-convex setting. Empirical evaluation shows that FedBuff is up to 3.8× faster than synchronous FL (FedAvgM (Hsu et al., 2019)), and up to 2.5× faster than asynchronous FL (FedAsync (Xie et al., 2019)). Additionally, FedBuff is robust to different staleness distributions.

FedBuff, being an asynchronous method, may be harder to debug than synchronous methods. In the setting where K=concurrency, FedBuff does update the server model more frequently than synchronous methods, and then its only advantage is robustness to stragglers. Our empirical evaluation makes assumptions about staleness distributions. We have validated these assumptions in a production system and also simulate with different distributions. However, under other staleness assumptions, FedBuff may behave differently. FedBuff adds an extra hyperparameter (staleness exponent) that needs to be tuned. We leave a rigorous analysis of FedBuff with differential privacy as future work.

References

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

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

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

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

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

Bonawitz, K. A., Ivanov, V., Kreuter, B., Marcedone, A., McMahan, H. B., Patel, S., Ramage, D., Segal, A., and Seth, K. 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.

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

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

Chaturapruek, S., Duchi, J. C., and Ré, C. 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.

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

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

Emanuel, H. Pysyft, pytorch and Intel SGX: Secure aggregation on trusted execution environments. Available online at https://blog.openmined.org/author/hericles/, version dated April 15, 2020.

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

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

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

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

Haddadpour, F., Kamani, M. M., Mahdavi, M., and Cadambe, V. R. Local sgd with periodic averaging: Tighter analysis and adaptive synchronization. *arXiv preprint arXiv:1910.13598*, 2019.

Hard, A., Rao, K., Mathews, R., Ramaswamy, S., Beaufays, F., Augenstein, S., Eichner, H., Kiddon, C., and Ramage, D. Federated learning for mobile keyboard prediction, 2019.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Wu, C.-J., Brooks, D., Chen, K., Chen, D., Choudhury, S., Dukhan, M., Hazelwood, K., Isaac, E., Jia, Y., Jia, B., Leyvand, T., Lu, H., Lu, Y., Qiao, L., Reagen, B., Spisak, J., Sun, F., Tulloch, A., Vajda, P., Wang, X., Wang, Y., Wasti, B., Wu, Y., Xian, R., Yoo, S., and Zhang, P. Machine learning at facebook: Understanding inference at the edge. pp. 331–344, 2019. doi: 10.1109/HPCA.2019.00048.

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

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

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

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

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

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

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

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

Yu, H., Yang, S., and Zhu, S. 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, pp. 5693–5700, 2019b.

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

Supplementary Material for
Federated Learning with Buffered Asynchronous Aggregation

A. Practical Improvements

Staleness scaling. Corollary 1 is derived based on constant learning rate. Equation (5) suggests a server learning rate adaptive to the staleness can potentially be beneficial in practice. To control the effect of staleness $\tau_i(t)$ in client i's contribution to the t-th server update, we adopt a polynomial staleness function, $s(\tau_i(t)) := 1/(1+t)^{\alpha}$, as suggested in (Xie et al., 2019) to adaptively discount stale client terms when they are aggregated at the server.

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., $\eta_{\mathrm{LRN}} := \eta_\ell \cdot n_{i,q}^t / B$, where $n_{i,q}^t \le B$ is actual batch size used for the step. We find that this small change can improve the behavior of asynchronous FL. A theoretical justification and detailed comparison is provided in Appendix B.2.

B. Additional Experiments

B.1. FedBuff with Different Values of K

In Table B.1 we present the number of client updates to reach validation accuracy on CelebA and Sent140 for different values of K. As with all experiments, we tune for the best learning rates and server momentum. We find that FedBuff with lower values of K reaches high accuracy quicker on CelebA, though increasing K from 1 to 10 speeds up training on Sent140. Note that there is a point of diminishing returns as K increases. We show the training curves of FedBuff along with other algorithms in Appendix C.4.

Table B.1. Number of client updates (lower is better) to reach validation accuracy on CelebA (90%) and Sent140 (69%). We set $M = 1000$ for all methods (Units = 1000 updates).

Dataset	K	Number of Client Updates
CelebA	1	20.8
	10	27.1
	100	57.6
Sent140	1	190.0
	10	124.7
	100	178.2

B.2. Learning Rate Normalization (LR-Norm)

B.2.1. Theoretical Justification

Recall that LR-Norm 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 update as

$$y_q = y_{q-1} - \eta_q g_q^{(b_q)}, \tag{2}$$

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 (2) 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*

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

Then

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

First, note that A_Q is strictly increasing in Q, since $1 \leq b_q \leq B$. 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

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

and thus

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

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,

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

Rearranging, we get

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

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

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

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

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

□

B.2.2. Empirical Evaluation

In Table B.2, 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, as it is illustrated in Figure B.1. 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, as it is illustrated in Figure B.2.

Table B.2. 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
CelebA	1	20.8	23.9	20.7
	10	27.1	25.5	28.7
	100	57.6	57.6	54.4
Sent140	1	190.0	201.9	201.9
	10	124.7	207.9	136.6
	100	178.2	570.3	231.7

C. Experiment Details

C.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 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, pass through 2 LSTM layers and a 128 hidden unit linear layer to output labels 0 or 1. We set our dropout rate to 0.1. We split the data into 80% 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.

C.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

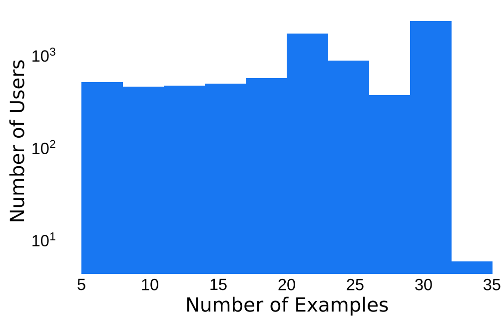

Figure B.1. Statistics for CelebA dataset. For this dataset, we use batch size of 32, hence each user has a single batch.

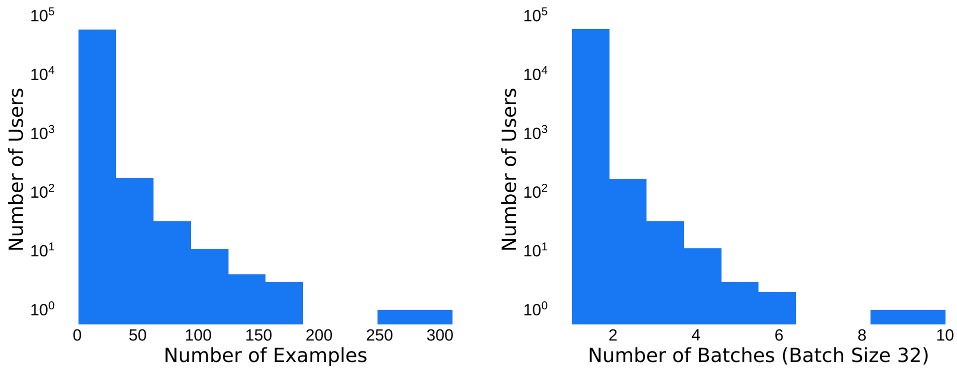

Figure B.2. Statistics for Sent140 dataset.

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.

C.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.

C.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.

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

C.3.2. Best Performing Hyperparameters

Table C.1 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 3, we set staleness exponent $\alpha = 10$. We set $\alpha = 0.5$ for all other experiments.

Table C.1. The best performing hyperparameters for Table 1

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

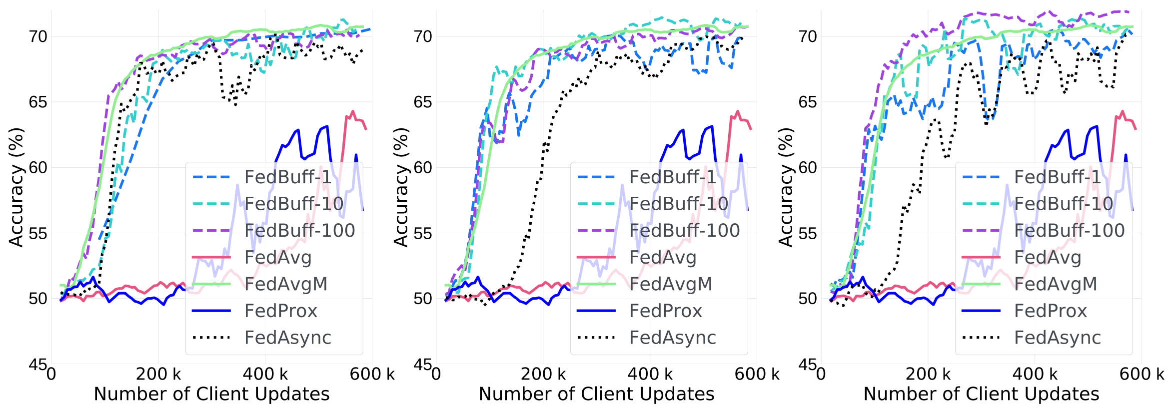

Figure C.1. Validation accuracy over the course of training for all algorithms on Sent140 for different distributions (left: uniform, center: half-normal, right: exponential) of staleness.

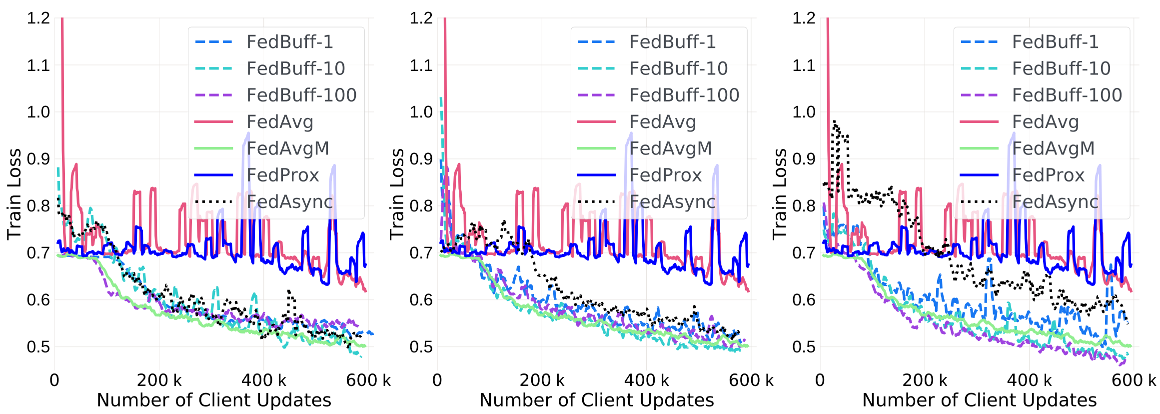

Figure C.2. Train loss over the course of training for all algorithms on Sent140 for different distributions (left: uniform, center: half-normal, right: exponential) of staleness.

C.4. Learning Curves

In this appendix we show the learning curves for each algorithm in Figures C.1, C.3, C.2 and C.4. 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.

We find that algorithms without momentum, FedProx and FedAvg, have erratic train loss curves. This highlights the importance of momentum tuning in FL. The erratic train loss curves for FedProx and FedAvg is consistent with findings in (Li et al., 2018).

D. Convergence Analysis

This section provides a convergence guarantee for FedBuff in the smooth, non-convex setting. A summary of the notation used is provided in Table C.2. Most previous work analyzes synchronous federated learning methods (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 & Mahdavi, 2019; Karimireddy et al., 2020). In contrast, in FedBuff, clients are trained asynchronously and the client updates are first

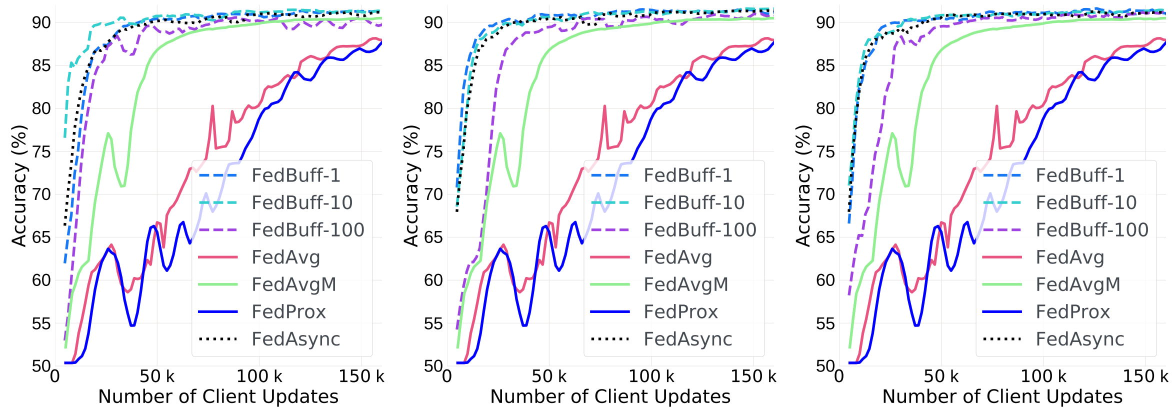

Figure C.3. Validation accuracy over the course of training for all algorithms on CelebA for different distributions (left: uniform, center: half-normal, right: exponential) of staleness.

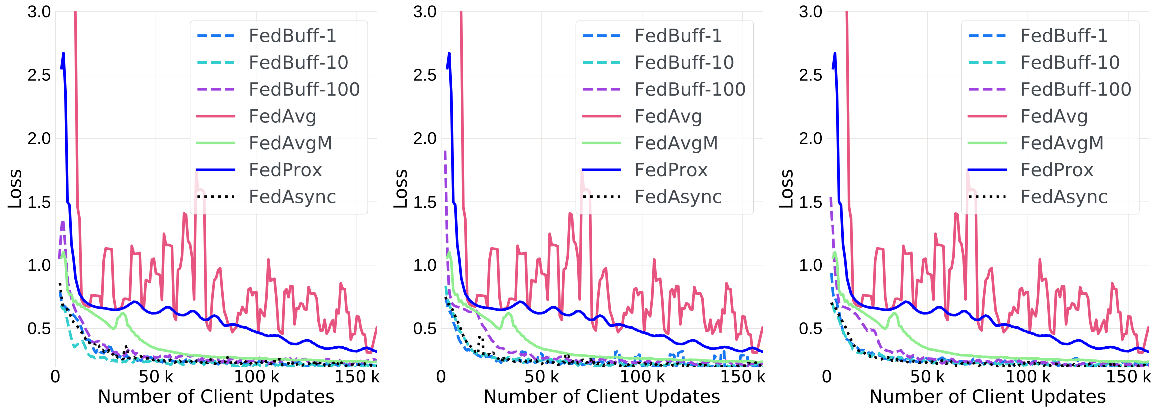

Figure C.4. Train loss over the course of training for all algorithms on CelebA for different distributions (left: uniform, center: half-normal, right: exponential) of staleness..

aggregated in a buffer before producing a global model update. Hence, it is important to understand the relationship between client computation and global communication under asynchrony and buffered aggregation. 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, $g_i(w;\zeta_i)$ 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 in the analysis.

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,*

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

Table C.2. 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 delay	$\tau_{\max}$

Assumption (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 delay under asynchrony.

Assumption 5. *(Bounded delay) For all clients $i \in [m]$ and for each server step t, the delay $\tau_i(t)$ between the checkpoint in which `FedBuff-client` is triggered, and the checkpoint in which Δ^i is used to modify the global model is not larger than $\tau_{\max}$.*

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 $\eta_g \eta_\ell^{(q)} KQ \leq \frac{1}{L}$ for all local steps $q = 0, \cdots, Q-1$, the global model iterates in FedBuff (Algorithm 1) achieve the following ergodic convergence rate*

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

The proof of Theorem 1 is provided in Appendix E.

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

$$\frac{1}{T}\sum_{t=0}^{T-1} \mathbb{E}\left[\left\|\nabla f(w^t)\right\|^2\right] \leq \frac{2F^*}{\eta_g\eta_\ell QKT} + \frac{L}{2}\eta_g\eta_\ell\sigma_\ell^2 + 3L^2Q^2\eta_\ell^2\Big(\eta_g^2K^2\tau_{\max}^2 + 1\Big)\sigma^2, \quad (4)$$

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

$$\frac{1}{T}\sum_{t=0}^{T-1} \mathbb{E}\left[\left\|\nabla f(w^t)\right\|^2\right] \leq \mathcal{O}\left(\frac{F^*}{\eta_g\sqrt{TKQ}}\right) + \mathcal{O}\left(\frac{\eta_g\sigma_\ell^2}{\sqrt{TKQ}}\right) + \mathcal{O}\left(\frac{Q\sigma^2}{TK}\right) + \mathcal{O}\left(\frac{\eta_g^2 QK\sigma^2\tau_{\max}^2}{T}\right). \quad (5)$$

Corollary 1 yields several insights:

Worst-case iteration complexity. Taking $\eta_g = \mathcal{O}(1)$ and satisfying the step-size constraint in Corollary 1, the convergence rate of FedBuff is dominated by $\mathcal{O}\left(\sigma_\ell^2/\sqrt{TKQ}\right)$ for large T. FedBuff requires $T = \mathcal{O}\left(1/(\epsilon^2 KQ)\right)$ server updates to guarantee $(1/T)\sum_{t=1}^{T}\mathbb{E}[\|\nabla f(w^t)\|^2] \leq \epsilon$. This is of the same order as the dominant term in synchronous methods such as FedAvg and SCAFFOLD (Karimireddy et al., 2020).

Relation between communication and local computation. Note that in equation (5), 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 global variance σ^2 and the impact of the worst-case delay $\tau_{\max}$.

Effect of server learning rate η_g. In (5), η_g is reciprocal in the first term compared to the second and the fourth term. When taking a large server learning rate, the loss F^* decreases faster, but the effect of variance σ_ℓ^2, σ^2, and staleness $\tau_{\max}$ are amplified. On the other hand, a smaller server learning rate η_g controls the variance and the effect of delay at the cost of amplifying the dominant term involving F^*. This suggests that in practice it may be better to initially have larger η_g, when F^* dominates the error, and to reduce η_g later in training then the local noise σ_ℓ^2 dominates the error.

Effect of staleness. The effect of delay between the initialization of `ClientOpt` and the server update dissipates at the rate of $1/T$ according to the fourth term in (5). The effect of staleness can be controlled by taking the server learning rate as $\eta_g = \mathcal{O}\left(1/\tau_{\max}\right)$, at the cost of slower convergence of the loss term F^*.

E. Proof of Convergence Rate

In this appendix, we prove the main convergence result for FedBuff.

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 Appendix D, 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,

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

□

E.1. Proof of Theorem 1

Theorem. *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* $\eta_g \eta_\ell^{(q)} KQ \leq \frac{1}{L}$ *for all local steps* $q = 0, \cdots, Q-1$, *the global model iterates in FedBuff*

(Algorithm 1) achieve the following ergodic convergence rate

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

Proof. By L-smoothness assumption,

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

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

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

Using conditional expectation, the expectation operator can be written as $\mathbb{E}[\cdot] := \mathbb{E}_{\mathcal{H}}\mathbb{E}_k\mathbb{E}_{g_k|k,\mathcal{H}}[\cdot]$, where $\mathbb{E}_{\mathcal{H}}$ is the expectation over the history of the iterates, $\mathbb{E}_k$ is evaluated over the randomness over the distribution of clients $k\sim[m]$ checking in at time-step t, and the inner expectation operates over the stochastic gradient of one step on a client. Hence, following unbiasedness,

$$\begin{aligned} \mathbb{E}[T_1] &= -\eta_g\mathbb{E}_{\mathcal{H}}\sum_{k\in\mathcal{S}_t}\sum_{q=0}^{Q-1}\eta_\ell^{(q)}\mathbb{E}_{k\sim[m]}\mathbb{E}_{g_k|k}\left\langle\nabla f(w^t), g_k(y_{k,q}^{t-\tau_k(t)})\right\rangle \\ &= -\eta_g\mathbb{E}_{\mathcal{H}}\sum_{k\in\mathcal{S}_t}\sum_{q=0}^{Q-1}\eta_\ell^{(q)}\mathbb{E}_{k\sim[m]}\left\langle\nabla f(w^t), \nabla F_k(y_{k,q}^{t-\tau_k(t)})\right\rangle \\ &= -\eta_g\mathbb{E}_{\mathcal{H}}\sum_{k\in\mathcal{S}_t}\sum_{q=0}^{Q-1}\eta_\ell^{(q)}\left\langle\nabla f(w^t), \frac{1}{m}\sum_{i=1}^{m}\nabla F_i(y_{i,q}^{t-\tau_i(t)})\right\rangle. \end{aligned} \quad (10)$$

From the identity

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

we have

$$\begin{aligned} \mathbb{E}[T_1] = &-\frac{K\eta_g}{2}\left(\sum_{q=0}^{Q-1}\eta_\ell^{(q)}\right)\left\|\nabla f(w^t)\right\|^2 + \sum_{q=0}^{Q-1}\frac{K\eta_g\eta_\ell^{(q)}}{2}\Big(-\mathbb{E}_{\mathcal{H}}\left\|\frac{1}{m}\sum_{i=1}^{m}\nabla F_i(y_{i,q}^{t-\tau_i(t)})\right\|^2 \\ &+\mathbb{E}_{\mathcal{H}}\underbrace{\left\|\nabla f(w^t)-\frac{1}{m}\sum_{i=1}^{m}\nabla F_i(y_{i,q}^{t-\tau_i(t)})\right\|^2}_{T_3}\Big) \end{aligned} \quad (11)$$

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

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

Further, by telescoping, T_3 can be decomposed as

$$\begin{aligned}\mathbb{E}[T_3] &= \frac{1}{m}\sum_{i=1}^{m}\mathbb{E}_{\mathcal{H}}\left\|\nabla F_i(w^t) - \nabla F_i(w^{t-\tau_i(t)}) + \nabla F_i(w^{t-\tau_i(t)}) - \nabla F_i(y_{i,q}^{t-\tau_i(t)})\right\|^2 \\ &\leq \frac{2}{m}\sum_{i=1}^{m}\mathbb{E}_{\mathcal{H}}\Big(\underbrace{\left\|\nabla F_i(w^t) - \nabla F_i(w^{t-\tau_i(t)})\right\|^2}_{\text{staleness}} + \underbrace{\left\|\nabla F_i(w^{t-\tau_i(t)}) - \nabla F_i(y_{i,q}^{t-\tau_i(t)})\right\|^2}_{\text{local drift}}\Big) \\ &\leq \frac{2}{m}\sum_{i=1}^{m}\Big(L^2\mathbb{E}_{\mathcal{H}}\left\|w^t - w^{t-\tau_i(t)}\right\|^2 + L^2\mathbb{E}_{\mathcal{H}}\left\|w^{t-\tau_i(t)} - y_{i,q}^{t-\tau_i(t)}\right\|^2\Big).\end{aligned} \tag{13}$$

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 obtain an upper bound on the staleness of initial model from which the client models are trained.

$$\begin{aligned}\left\|w^t - w^{t-\tau_i(t)}\right\|^2 &= \left\|\sum_{\rho=t-\tau_i(t)}^{t-1}(w^{\rho+1} - w_\rho)\right\|^2 = \left\|\sum_{\rho=t-\tau_i(t)}^{t-1}\eta_g\sum_{j_\rho\in\mathcal{S}_\rho}\Delta_{j_\rho}^{\rho}\right\|^2 \\ &= \eta_g^2\left\|\sum_{\rho=t-\tau_i(t)}^{t-1}\sum_{j_\rho\in\mathcal{S}_\rho}\sum_{l=0}^{Q-1}\eta_\ell^{(l)}g_{j_\rho}(y_{j_\rho,l}^{\rho})\right\|^2\end{aligned} \tag{14}$$

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

$$\begin{aligned}\mathbb{E}_{\mathcal{H}}\left\|w^t - w^{t-\tau_i(t)}\right\|^2 &\leq \eta_g^2 QK\tau_i(t)\sum_{\rho=t-\tau_i(t)}^{t-1}\sum_{j_\rho\in\mathcal{S}_\rho}\sum_{l=0}^{Q-1}(\eta_\ell^{(l)})^2\mathbb{E}\left\|g_{j_\rho}(y_{j_\rho,l}^{\rho})\right\|^2 \\ &\leq 3\eta_g^2 QK^2\max_{\tau_i(t)}\tau_i(t)^2\Big(\sum_{l=0}^{Q-1}(\eta_\ell^{(l)})^2\Big)\Big(\sigma_\ell^2+\sigma_g^2+G\Big) \\ &\leq 3\eta_g^2 QK^2\tau_{\max}^2\Big(\sum_{l=0}^{Q-1}(\eta_\ell^{(l)})^2\Big)\Big(\sigma_\ell^2+\sigma_g^2+G\Big)\end{aligned} \tag{15}$$

where the last inequality follows from the assumption on maximal delay and applying Lemma 1. Similarly, we can find an upper bound for the local drift term as

$$\begin{aligned}\mathbb{E}\left\|w^{t-\tau_i(t)} - y_{i,q}^{t-\tau_i(t)}\right\|^2 &= \mathbb{E}\left\|y_{i,0}^{t-\tau_i(t)} - y_{i,q}^{t-\tau_i(t)}\right\|^2 \\ &\leq \mathbb{E}\left\|\sum_{l=0}^{q-1}\eta_\ell^{(l)}g_i(y_{i,l}^{t-\tau_i(t)})\right\|^2 \\ &\leq 3q\left(\sum_{l=0}^{q-1}(\eta_\ell^{(l)})^2\right)\Big(\sigma_\ell^2+\sigma_g^2+G\Big)\end{aligned} \tag{16}$$

Thus, the upper bound on T_3 becomes:

$$\begin{aligned}\mathbb{E}[T_3] &\leq 6\left(L^2\eta_g^2 QK^2\tau_{\max}^2\Big(\sum_{i=0}^{Q-1}(\eta_\ell^{(i)})^2\Big)\Big(\sigma_\ell^2+\sigma_g^2+G\Big) + L^2 q\left(\sum_{i=0}^{q-1}(\eta_\ell^{(i)})^2\right)\Big(\sigma_\ell^2+\sigma_g^2+G\Big)\right) \\ &\leq 6L^2\Big(\sum_{i=0}^{Q-1}(\eta_\ell^{(i)})^2\Big)(\eta_g^2 QK^2\tau_{\max}^2 + q)\Big(\sigma_\ell^2+\sigma_g^2+G\Big) \\ &\leq 6L^2 Q\Big(\sum_{i=0}^{Q-1}(\eta_\ell^{(i)})^2\Big)(\eta_g^2 K^2\tau_{\max}^2 + 1)\Big(\sigma_\ell^2+\sigma_g^2+G\Big)\end{aligned} \tag{17}$$

Plugging the upper bound on T_3 into (11), we have,

$$\begin{aligned}\mathbb{E}[T_1] \leq & -\frac{K\eta_g}{2}\left(\sum_{q=0}^{Q-1}\eta_\ell^{(q)}\right)\left\|\nabla f(w^t)\right\|^2 + \sum_{q=0}^{Q-1}\frac{K\eta_g\eta_\ell^{(q)}}{2}\mathbb{E}[T_3] \\ & -\sum_{q=0}^{Q-1}\frac{K\eta_g\eta_\ell^{(q)}}{2}\mathbb{E}_{\mathcal{H}}\left\|\frac{1}{m}\sum_{i=1}^{m}\nabla F_i(y_{i,q}^{t-\tau_i(t)})\right\|^2\end{aligned} \tag{18}$$

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

$$\begin{aligned}\mathbb{E}[T_1] \leq & -\frac{K\eta_g\alpha(Q)}{2}\left\|\nabla f(w^t)\right\|^2 + 3K\eta_g L^2 Q\alpha(Q)\beta(Q)\left(\eta_g^2K^2\tau_{\max}^2+1\right)\left(\sigma_\ell^2+\sigma_g^2+G\right) \\ & \underbrace{-\sum_{q=0}^{Q-1}\frac{K\eta_g\eta_\ell^{(q)}}{2}\mathbb{E}_{\mathcal{H}}\left\|\frac{1}{m}\sum_{i=1}^{m}\nabla F_i(y_{i,q}^{t-\tau_i(t)})\right\|^2}_{T_4}\end{aligned} \tag{19}$$

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

$$\begin{aligned}\mathbb{E}[T_2] &= \mathbb{E}\left[\frac{L\eta_g^2}{2}\left\|\sum_{k\in\mathcal{S}_t}\sum_{q=0}^{Q-1}\eta_\ell^{(q)}g_k(y_{k,q}^{t-\tau_k(t)})\right\|^2\right] \\ &= \mathbb{E}\left[\frac{L\eta_g^2}{2}\left\|\sum_{k\in\mathcal{S}_t}\sum_{q=0}^{Q-1}\eta_\ell^{(q)}\left(g_k(y_{k,q}^{t-\tau_k(t)})-\nabla F_k(y_{k,q}^{t-\tau_k(t)})\right)+\sum_{k\in\mathcal{S}_t}\sum_{q=0}^{Q-1}\eta_\ell^{(q)}\nabla F_k(y_{k,q}^{t-\tau_k(t)})\right\|^2\right] \\ &\overset{\text{(A.)}}{=} \frac{L\eta_g^2}{2}\mathbb{E}\left\|\sum_{k\in\mathcal{S}_t}\sum_{q=0}^{Q-1}\eta_\ell^{(q)}\left(g_k(y_{k,q}^{t-\tau_k(t)})-\nabla F_k(y_{k,q}^{t-\tau_k(t)})\right)\right\|^2 + \frac{L\eta_g^2}{2}\mathbb{E}\left\|\sum_{k\in\mathcal{S}_t}\sum_{q=0}^{Q-1}\eta_\ell^{(q)}\nabla F_k(y_{k,q}^{t-\tau_k(t)})\right\|^2 \\ &\overset{\text{(B.)}}{=} \frac{L\eta_g^2}{2}\sum_{k\in\mathcal{S}_t}\sum_{q=0}^{Q-1}(\eta_\ell^{(q)})^2\mathbb{E}\left\|\left(g_k(y_{k,q}^{t-\tau_k(t)})-\nabla F_k(y_{k,q}^{t-\tau_k(t)})\right)\right\|^2 + \frac{L\eta_g^2}{2}\mathbb{E}\left\|\sum_{k\in\mathcal{S}_t}\sum_{q=0}^{Q-1}\eta_\ell^{(q)}\nabla F_k(y_{k,q}^{t-\tau_k(t)})\right\|^2 \\ &\leq \frac{LK\eta_g^2\beta(Q)\sigma_\ell^2}{2} + \frac{LKQ\eta_g^2}{2}\sum_{k\in\mathcal{S}_t}\sum_{q=0}^{Q-1}(\eta_\ell^{(q)})^2\mathbb{E}_{\mathcal{H}}\mathbb{E}_{k\sim[m]|\mathcal{H}}\left\|\nabla F_k(y_{k,q}^{t-\tau_k(t)})\right\|^2 \\ &= \frac{LK\eta_g^2\beta(Q)\sigma_\ell^2}{2} + \frac{LKQ\eta_g^2}{2}\sum_{k\in\mathcal{S}_t}\sum_{q=0}^{Q-1}(\eta_\ell^{(q)})^2\mathbb{E}_{\mathcal{H}}\left[\frac{1}{m}\sum_{i=1}^{m}\left\|\nabla F_i(y_{i,q}^{t-\tau_i(t)})\right\|^2\right] \\ &= \frac{LK\eta_g^2\beta(Q)\sigma_\ell^2}{2} + \underbrace{\frac{LK^2Q\eta_g^2}{2m}\sum_{q=0}^{Q-1}\sum_{i=1}^{m}(\eta_\ell^{(q)})^2\mathbb{E}_{\mathcal{H}}\left[\left\|\nabla F_i(y_{i,q}^{t-\tau_i(t)})\right\|^2\right]}_{T_5}\end{aligned} \tag{20}$$

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

$$\begin{aligned}
&\big(T_4 + T_5\big) \\
=& -\sum_{q=0}^{Q-1} \frac{K\eta_g\eta_\ell^{(q)}}{2}\mathbb{E}_{\mathcal{H}}\left\|\frac{1}{m}\sum_{i=1}^{m}\nabla F_i(y_{i,q}^{t-\tau_i(t)})\right\|^2 + \frac{LK^2Q\eta_g^2}{2m}\sum_{q=0}^{Q-1}\sum_{i=1}^{m}(\eta_\ell^{(q)})^2\mathbb{E}_{\mathcal{H}}\left\|\nabla F_i(y_{i,q}^{t-\tau_i(t)})\right\|^2 \\
=& -\sum_{q=0}^{Q-1}\sum_{i=1}^{m} \frac{K\eta_g\eta_\ell^{(q)}}{2m}\mathbb{E}_{\mathcal{H}}\left\|\nabla F_i(y_{i,q}^{t-\tau_i(t)})\right\|^2 + \frac{LK^2Q\eta_g^2}{2m}\sum_{q=0}^{Q-1}\sum_{i=1}^{m}(\eta_\ell^{(q)})^2\mathbb{E}_{\mathcal{H}}\left\|\nabla F_i(y_{i,q}^{t-\tau_i(t)})\right\|^2 \\
=& \sum_{q=0}^{Q-1}\sum_{i=1}^{m}\Big(-\frac{K\eta_g\eta_\ell^{(q)}}{2m} + \frac{LK^2Q\eta_g^2(\eta_\ell^{(q)})^2}{2m}\Big)\mathbb{E}_{\mathcal{H}}\left\|\nabla F_i(y_{i,q}^{t-\tau_i(t)})\right\|^2
\end{aligned} \tag{21}$$

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

Now, plugging (19), (20) and (21) into (8),

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

Summing up t from 1 to T and rearrange, yields

$$\begin{aligned}
&\sum_{t=0}^{T-1}\eta_g K\alpha(Q)\mathbb{E}\left[\left\|\nabla f(w^t)\right\|^2\right] \\
\leq& \sum_{t=0}^{T-1} 2\Big(\mathbb{E}[f(w^t)] - \mathbb{E}[f(w^{t+1})]\Big) + 3\sum_{t=0}^{T-1}\eta_g L^2KQ\alpha(Q)\beta(Q)\Big(\eta_g^2K^2\tau_{\max}^2 + 1\Big)\Big(\sigma_\ell^2 + \sigma_g^2 + G\Big) \\
&+ \frac{L}{2}\eta_g^2\beta(Q)K\sigma_\ell^2 \\
\leq& 2\Big(f(w^0) - f(w^*)\Big) + 3\sum_{t=0}^{T-1}\eta_g L^2K\alpha(Q)\beta(Q)\Big(\eta_g^2K^2\tau_{\max}^2 + Q\Big)\Big(\sigma_\ell^2 + \sigma_g^2 + G\Big) \\
&+ \frac{L}{2}\eta_g^2\beta(Q)K\sigma_\ell^2.
\end{aligned} \tag{23}$$

Thus we have

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left[\left\|\nabla f(w^t)\right\|^2\right] \leq \frac{2\Big(f(w^0) - f(w^*)\Big)}{\eta_g\alpha(Q)TK} + 3L^2Q\beta(Q)\Big(\eta_g^2K^2\tau_{\max}^2 + 1\Big)\Big(\sigma_\ell^2 + \sigma_g^2 + G\Big) + \frac{L}{2}\frac{\eta_g\beta(Q)}{\alpha(Q)}\sigma_\ell^2 \tag{24}$$

□