BLADE: Pushing the Performance Envelope of Asynchronous Federated Learning

Chen Ying, Baochun Li
Department of Electrical and Computer Engineering
University of Toronto

Bo Li
Department of Computer Science and Engineering
Hong Kong University of Science and Technology

Abstract*—Asynchronous federated learning (FL) has been proposed to decrease the training time in conventional FL where the communication paradigm is synchronous. Instead of aggregating after receiving updates from all the selected clients, an asynchronous FL server conducts aggregation without waiting for slow clients. Though superior to synchronous FL, the performance of existing works in asynchronous FL — measured by the wall-clock time of global training — leaves much to be desired, as the staleness of client updates may degrade the performance substantially. In this paper, we propose BLADE, a new staleness-aware framework that seeks to push the performance envelope of asynchronous FL by designing new mechanisms in all important design aspects of FL training, including client selection, adaptive pruning, quantization, and update aggregation. BLADE selects clients based on their staleness and the quality of their previous updates. Before reporting to the server, every client prunes its update with a pruning amount related to its staleness and quantizes the pruned update. When aggregating updates, BLADE tunes the aggregation weight of each update according to its staleness and divergence from the previous global model. In an extensive array of performance evaluations with six benchmark datasets, BLADE consistently showed its substantial performance superiority over its state-of-the-art competitors. It decreased the wall-clock training time by up to 64.6%.

I. INTRODUCTION

Federated learning (FL) [1], [2] has emerged as a popular distributed machine learning paradigm for a large number of clients to collaboratively train a high-performance model without revealing their private data. In each communication round, a shared global model is trained collaboratively by a set of selected clients using their local data. After receiving updates from all the selected clients, the server aggregates these updates to generate an improved global model. As data privacy has become a major concern, FL has received an extensive amount of research attention in the recent literature.

Similar to conventional distributed machine learning, FL aims to minimize the *elapsed wall-clock time* for the global model to converge to a target accuracy, which is the most important performance metric in FL. Unfortunately, it is common that computing capabilities vary widely among clients. For the same amount of computation, training times of different clients may follow a heavy-tailed distribution. Therefore, some clients need much more time to complete local training than others. These slow clients — known as *stragglers* [3] in conventional distributed machine learning — lead to an excessively long wall-clock time to finish a training session, since the server needs to wait for their updates.

To alleviate the performance degradation due to stragglers, it has been addressed that the FL server can proceed *asynchronously* without waiting for all the selected clients [4]–[8]. In asynchronous FL, the server aggregates after receiving updates from only a subset of the selected clients and starts a new round. It has been shown that asynchronous FL can far outperform synchronous FL when the clients are heterogeneous, which is the norm in reality. In fact, the performance advantages of asynchronous FL are so convincing that, even with only a mild degree of client heterogeneity, asynchronous FL should be used as the default FL mechanism [9].

Despite the performance benefits of asynchronous FL, existing works have not pushed its performance to the limits. There are three common directions to improve FL performance regarding the elapsed wall-clock time to complete a training session. *First*, as the data distribution across clients is non-i.i.d. (not independent and identically distributed) [10], [11], a *client selection* mechanism can be designed to select clients with a higher potential [12]. *Second*, as the sizes of modern models are excessively large, applying two common techniques *model pruning* [13], [14] and *quantization* [15] strategically to remove unnecessary parameters in updates can reduce the communication overhead without decreasing the global model accuracy. *Finally*, as the global model is produced by weighted averaging with updates, designing an *update aggregation* mechanism to assign a proper aggregation weight to each client's update can improve the validation accuracy of the converged global model [16], [17].

We argue that, however, directly applying existing mechanisms designed for synchronous FL may not be effective in asynchronous FL, and may prevent the global model from converging to the highest possible validation accuracy in the shortest amount of time. This is attributed to a key difference between synchronous and asynchronous FL: the training performance may be degraded due to *client staleness*. In asynchronous FL, the global model that a slow client used for local training may be *stale* as compared to the latest global model, since the server may have already advanced for quite a number of additional rounds by aggregating updates from

The research was supported in part by a RGC RIF grant under the contract R6021-20, a RGC TRS grant under the contract T43-513/23N-2, RGC CRF grants under the contracts C7004-22G and C1029-22G, and RGC GRF grants under the contracts 16200221, 16207922 and 16207423.

faster clients. Intuitively, an update from a client that is less stale should be of higher quality. Thus, it should be pruned with a smaller ratio and have a higher aggregation weight, and the client should have a higher probability of being selected in future rounds. As existing mechanisms in synchronous FL did not consider such staleness, it is challenging for them to perform well under asynchronous mode.

In this paper, we focus on pushing the performance envelope of asynchronous FL by minimizing the elapsed wall-clock training time used to reach a target accuracy. By designing and consolidating new client selection, pruning, quantization, and update aggregation mechanisms that are specifically custom-tailored for asynchronous FL, we propose BLADE, a staleness-aware asynchronous FL framework with a provable convergence guarantee. BLADE selects clients according to their selection scores, which are calculated based on their staleness and the quality of their latest reported updates. Pruning and quantization are leveraged to compress client updates with minimal amount of reduction or even an increase in global model accuracy. During update aggregation, the server assigns aggregation weights to client updates based on their staleness and degrees of divergence from the global model.

Highlights of our original contributions in this paper are three-fold. *First*, with BLADE, we seek to push the performance envelope of asynchronous FL by proposing new client selection, pruning, quantization, and update aggregation mechanisms that offer a theoretical convergence guarantee. *Second*, our mechanisms in BLADE are *staleness-aware*, in that updates with higher degrees of staleness are pruned more aggressively and have smaller aggregation weights. Also, these clients are less likely to be selected in future rounds. *Finally*, we have implemented BLADE in our open-source real-world FL research framework, which has been designed from scratch for reproducible and scalable FL research experiments. With an extensive array of experiments over a variety of datasets and models in both image classification and language modeling tasks, BLADE shows its capability to outperform its state-of-the-art competitors in the literature by a considerable margin, reducing the wall-clock training time by up to 64.6%.

II. BLADE: OVERVIEW, MOTIVATION, AND DESIGN

To minimize the wall-clock training time in asynchronous FL, we propose a new staleness-aware framework, BLADE, to enhance every important design aspect of the training process. Fig. 1 shows the overview of BLADE. Its client selector, compressor, and update aggregator are all designed based on client staleness, whose definition is as follows.

Definition 1 (Client staleness)**.** *In communication round t, client staleness s_n^t is the number of rounds that have elapsed since the last time client n received the global model from the server. If client n received the global model $\mathbf{w}^{\tau_n}$ and uses it to conduct local training, the client staleness $s_n^t := t - \tau_n$.*

A substitute for client staleness. However, the client staleness may not be readily available when needed. For example, a client cannot know its staleness when pruning its update.

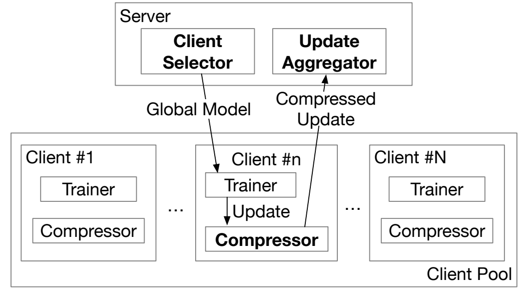


Fig. 1: BLADE: a design overview.

Thus, a substitute is required. Intuitively, if a client requires a longer local training time, its update will be staler because the server should finish more aggregations during its local training. Hence, a client's local training time can be a substitute. The server in BLADE records γ_i, the average training time of i reported clients. After receiving the $(i+1)$-th update from client n, the server updates γ_{i+1} as $\gamma_{i+1} = \frac{\gamma_i \times i + l_n/|\mathcal{D}_n|}{i+1}$, where l_n is the local training time of client n, $|\mathcal{D}_n|$ is the number of its data samples, and $\gamma_0 = 0$. As local dataset sizes usually vary among clients in FL, a client with a larger local dataset should need a longer local training time than one with the same computing capability but a smaller local dataset. For fairness, we use $l_n/|\mathcal{D}_n|$ instead of l_n to update γ.

The workflow. In the first communication round, the server randomly selects C clients from N clients. After receiving K ($K < C$) updates and aggregating them, the server updates the average training time γ, and its client selector selects K new clients. The server then sends the newly aggregated global model and the latest γ to the selected clients.

In communication round t, the selected client n conducts local training to compute its update Δ_n^t with local stochastic gradient descent (SGD) on its local dataset $\mathcal{D}_n$ and the current global model $\mathbf{w}^t$. It then reduces the size of Δ_n^t based on its staleness and received γ with a compressor, and sends its compressed update and $l_n/|\mathcal{D}_n|$ to the server. After the server receives K updates, the update aggregator generates a new global model. The next communication round then starts.

In the remainder of this section, we will show several motivational experiments and introduce our design of the client selector, compressor, and update aggregator in detail.

Experimental settings. We conducted our experiments on NVIDIA A100 GPUs with 40 GB of CUDA memory. Motivational experiments trained `LeNet-5` models with the Federated Extended MNIST (`FEMNIST`) dataset, which provides 3597 local datasets with a highly skewed non-i.i.d. distribution. For local training, we set the batch size to 32, epoch number to 5, learning rate to 0.01, and momentum to 0.9. The baseline was *FedBuff* [8], the leading asynchronous FL mechanism where the server aggregates every K updates. K was set to 50.

A. Client Selector

We introduce a client selector for the server to select clients that are more likely to generate high-quality updates in a short amount of time. The client selector records a *selection score* for every client. Clients with higher selection scores have

higher probabilities of being selected. All scores are initialized as K, the number of updates required for aggregation. Denote the set of clients whose updates are aggregated in communication round t as $\mathcal{A}^t$. After the t-th aggregation, the client selector updates the selection score of client n, $\forall n \in \mathcal{A}^t$ as:

$$\text{Score}_n^t = \underbrace{\frac{|\mathcal{D}^t|}{|\mathcal{D}_n|}\frac{\Theta(\Delta_n^t, \mathbf{w}^{t+1} - \mathbf{w}^t) + 1}{2}}_{\text{Update quality}} \times \underbrace{\text{sigmoid}\left(\frac{\gamma t K}{\frac{l_n}{|\mathcal{D}_n|}}\right)^{\alpha}}_{\text{Local training time}}, \tag{1}$$

where $|\mathcal{D}^t| = \sum_{n \in \mathcal{A}^t} |\mathcal{D}_n|$. Θ is cosine similarity.

The first component of Eq. (1) represents the update quality. Ideally, client update Δ_n^t should have the same angle as $\mathbf{w}^{t+1} - \mathbf{w}^t$. Therefore, we use the cosine similarity Θ to quantify the quality of Δ_n^t, so that a client with a smaller angle would have a higher selection score. The second component assigns higher selection scores to clients with shorter local training times by using a monotonically increasing function, sigmoid. The hyperparameter α controls how much the local training time should contribute to the selection score.

In the next communication round $t+1$, client n will be selected with the probability of $\psi_n^{t+1} = \frac{\text{Score}_n^t}{\sum_{i \in \mathcal{S}^{t+1}} \text{Score}_i^t}$, where $\mathcal{S}^{t+1}$ are the set of clients who are available to participate in this round, i.e., they are not currently training. For any client $i \in \mathcal{S}^{t+1}$ and $i \notin \mathcal{A}^t$, $\text{Score}_i^t = \text{Score}_i^{t-1}$.

To empirically validate our design of the client selector, we replaced the random client selector on *FedBuff* [8] with it, and evaluated its performance with different values of α. We also added *Oort* [12], a state-of-the-art client selection mechanism designed for synchronous FL, to *FedBuff* for comparison. Fig. 2a shows that with *Oort*, *FedBuff* performed better than without it, indicating that a carefully designed client selection mechanism is better than random selection. However, our new client selector works even better with whichever value of α.

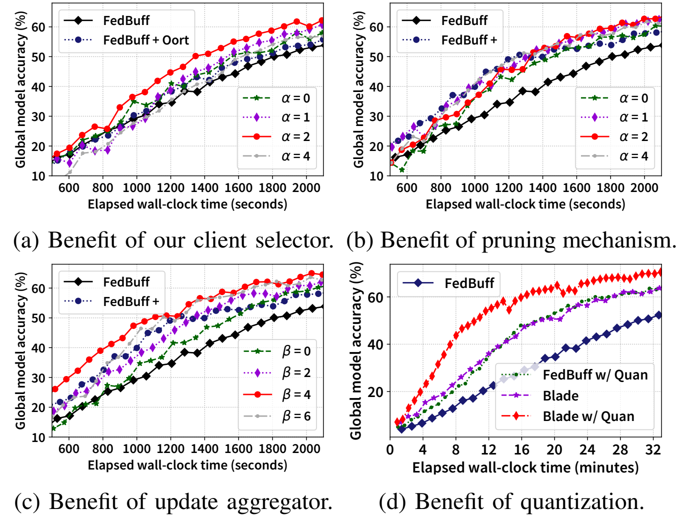
Oort compares l_n with Γ, a predefined desired training time, and decreases their probabilities of being selected by the same amount for any client n with $l_n > \Gamma$. However, it is not reasonable to penalize all slow clients by the same amount as their local training times and staleness may vary greatly. Furthermore, it is nontrivial to choose a proper value of Γ. In contrast, our client selector uses the average training time γ to estimate the staleness of updates and penalize slow clients.

The varying performance from different values of α also shows the importance of considering staleness in asynchronous FL. Among the four different values, the worst performance occurred when $\alpha = 0$, which turns the second component in Eq. (1) to 0.5 for all clients regardless of their staleness.

B. Compressor: Pruning

Pruning [18] can significantly reduce the size of a neural network model by reducing its number of parameters with a negligible reduction in validation accuracy. In FL, pruning can reduce the sizes of updates and thus shorten the transmission time, which takes up a major portion of the training time.

The core of a pruning mechanism is its policy to determine the least important parameters that can be removed with



(a) Benefit of our client selector. (b) Benefit of pruning mechanism.

(c) Benefit of update aggregator. (d) Benefit of quantization.

Fig. 2: Designing BLADE: motivational examples.

minimal impact on accuracy. One common policy is based on magnitude, such as its L1-norm, since parameters of lower magnitudes usually have less significant effects on the output, and hence are less likely to affect the model performance.

BLADE uses L1-norm pruning with an adaptive pruning amount. Client n zeros out ρ_n of its update parameters with the smallest L1-norm. We set the pruning amount ρ_n as:

$$\rho_n = 1 - \text{sigmoid}\left(\frac{\gamma}{\frac{l_n}{|\mathcal{D}_n|}}\right). \tag{2}$$

As client staleness s_n^t is unknown at this point, we use $l_n/|\mathcal{D}_n|$ as an alternative to tune ρ_n. The design rule of Eq. (2) is to assign higher ρ_n to client n with a longer local training time.

In addition to our client selector, we add our pruning mechanism to *FedBuff* and compare it with *FedBuff + Oort + FedSCR*. Designed for synchronous FL, *FedSCR* [14] prunes each update's entire filters and channels if their summed parameter values are below a particular threshold. We have implemented *Zstandard* [19], a fast real-time compression algorithm, to compress data before transmitting it over the network, so that zeros in pruned updates will take up no space. Fig. 2b indicates the advantages of using pruning in addition to client selection and considering staleness for pruning.

C. Update Aggregator

In communication round t, the server generates an improved global model after receiving K updates:

$$\mathbf{w}^{t+1} = \mathbf{w}^t + \sum_{n \in \mathcal{A}^t} p_n^t \Delta_n^t. \tag{3}$$

Conventionally, $p_n^t = |\mathcal{D}_n| / \sum_{n \in \mathcal{A}^t} |\mathcal{D}_n|$ [1]. To improve performance, updates with higher quality and lower staleness should have larger aggregation weights. Thus, our update aggregator assigns the aggregation weight p_n^t to client n as:

$$p_n^t = \underbrace{\frac{|\mathcal{D}_n|}{|\mathcal{D}^t|}\frac{\Theta(\Delta_n^t, \mathbf{w}^t - \mathbf{w}^{t-1}) + 1}{2}}_{\text{Estimation of update quality}} \times \underbrace{\left(1 - \frac{s_n^t}{s_{\max}^t + 1}\right)^{\beta}}_{\text{Staleness}}, \tag{4}$$

where $s^t_{\max}$ is the highest staleness of updates to be aggregated. After normalizing p^t_n, $\forall n \in \mathcal{A}^t$, such that their sum becomes 1, the update aggregator aggregates a new global model with Eq. (3). The hyperparameter in Eq. (4) β controls how significantly staleness should affect the aggregation weights.

We further add our update aggregator to *FedBuff* with our client selector and pruning mechanism. As $\alpha = 2$ led to the best performance in the former two examples, we set $\alpha = 2$ and evaluated it with different values of β. Fig. 2c shows the advantage of our update aggregator. It converged to higher accuracy and further reduced the wall-clock training time.

D. An Additional Compressor: Quantization

The three aforementioned components constitute BLADE's foundation. In addition, we find that quantizing the global model and client updates from the commonly used format of 32-bit floating-point to 16-bit floating-point can reduce wall-clock training time without sacrificing global model accuracy. Fig. 2d demonstrates the advantage of using quantization in asynchronous FL. It largely reduced the wall-clock time for both *FedBuff* and BLADE. Thus, we propose to employ quantization as an additional compressor of BLADE.

III. BLADE: Convergence Analysis

This section presents BLADE's convergence guarantee based on the following assumptions that are commonly used in analyzing FL mechanisms [8], [10], [20], [21].

1) *Lipschitz gradient.* For all clients $n \in [N]$, their gradients are L-smooth: $\|\nabla F_n(\mathbf{w}) - \nabla F_n(\mathbf{w}')\| \le L\|\mathbf{w} - \mathbf{w}'\|^2$, where $\nabla F_n(\mathbf{w})$ denotes the stochastic gradient of model $\mathbf{w}$ with respect to the loss on data of client n.
2) *Unbiased local gradient.* $\mathbb{E}_{\zeta_n}[g_n(\mathbf{w};\zeta_n)] = \nabla F_n(\mathbf{w})$, where $g_n(\mathbf{w};\zeta_n)$ denotes the stochastic gradient on client n with randomness ζ_n.
3) *Bounded local and global variance.* $\mathbb{E}_{\zeta_n|n}\big[\|g_n(\mathbf{w};\zeta_n) - \nabla F_n(\mathbf{w})\|^2\big] \le \sigma_l^2, \forall n \in [N]$, and $\frac{1}{N}\sum_{n=1}^N \|\nabla F_n(\mathbf{w}) - \nabla f(\mathbf{w})\|^2 \le \sigma_g^2$, $\forall n \in [N]$, where $f(\mathbf{w}) := \sum_{n=1}^N p_n F_n(\mathbf{w})$ denotes the global learning objective.
4) *Uniformly bounded local gradient.* $\|\nabla F_n(\mathbf{w})\|^2 \le G$, $\forall n \in [N]$.

Theorem 1 (Convergence rate)**.** *The global model trained by* BLADE *has the following convergence rate:*

$$\frac{1}{T}\sum_{t=0}^{T-1}\|\nabla f(\mathbf{w}^t)\|^2 \le \frac{2\big(f(\mathbf{w}^0) - f(\mathbf{w}^*)\big)}{\eta KET} + \eta\sigma_l^2 NL + 6L^2\eta^2(KB+1)(\sigma_l^2 + \sigma_g^2 + G^2). \tag{5}$$

Proof. Due to L-smoothness and Eq. (3),

$$f(\mathbf{w}^{t+1}) \le f(\mathbf{w}^t) + \underbrace{\sum_{n\in\mathcal{A}^t} p^t_n\langle\nabla f(\mathbf{w}^t), \Delta^t_n\rangle}_{T_1} + \underbrace{\frac{L}{2}\|\sum_{n\in\mathcal{A}^t} p^t_n\Delta^t_n\|^2}_{T_2}. \tag{6}$$

Note that Δ^t_n is the update received by the server, which was pruned by client n. Denote the original update as $\bar{\Delta}^t_n$ and since $\bar{\Delta}^t_n = -\eta\sum_{e=1}^E g_n(\mathbf{w}^{\tau^t_n}_{n,e})$ due to SGD, we have $\Delta^t_n = \bar{\Delta}^t_n \odot M^t_n = (-\eta\sum_{e=1}^E g_n(\mathbf{w}^{\tau^t_n}_{n,e})) \odot M^t_n$, where $M^t_n \in \{0,1\}^{|\mathbf{w}|}$ is the pruning mask, η is the local learning rate, and E is the local epoch number. As M^t_n turns some parameters in $\bar{\Delta}^t_n$ to 0, $\|\Delta^t_n\| \le \|\bar{\Delta}^t_n\|$. The local update $\bar{\Delta}^t_n$ has the staleness of s^t_n. Hence, it was trained based on global model $\mathbf{w}^{\tau^t_n}$, $\tau^t_n := t - s^t_n$.

We first derive the upper bound of T_1. Using conditional expectation $\mathbb{E}[\cdot] := \mathbb{E}_{\mathcal{H}}\mathbb{E}_{n\sim[N]}\mathbb{E}_{g_n|n,\mathcal{H}}[\cdot]$, where $\mathbb{E}_{\mathcal{H}}$ is the expectation over the history of communication rounds,

$$\begin{aligned}
\mathbb{E}[T_1] &\overset{(A.)}{\le} -\frac{\eta K}{N}\mathbb{E}_{\mathcal{H}}\Big[\sum_{n=1}^N p^t_n\sum_{e=1}^E\Big\langle\nabla f(\mathbf{w}^t), \nabla F_n(\mathbf{w}^{\tau^t_n}_{n,e})\Big\rangle\Big] \\
&\overset{(B.)}{=} -\frac{\eta KE}{2N}\|\nabla f(\mathbf{w}^t)\|^2 \\
&\quad + \frac{\eta K}{2N}\sum_{e=1}^E\Big(- \mathbb{E}_{\mathcal{H}}\Big[\Big\|\sum_{n=1}^N p^t_n\nabla F_n(\mathbf{w}^{\tau^t_n}_{n,e})\Big\|^2\Big] \\
&\quad + \mathbb{E}_{\mathcal{H}}\Big[\underbrace{\Big\|\nabla f(\mathbf{w}^t) - \sum_{n=1}^N p^t_n\nabla F_n(\mathbf{w}^{\tau^t_n}_{n,e})\Big\|^2}_{T_3}\Big]\Big) \\
&\overset{(C.)}{\le} \frac{3L^2\eta^3EK}{N}(KB+1)(\sigma_l^2 + \sigma_g^2 + G^2) \\
&\quad - \frac{\eta KE}{2N}\|\nabla f(\mathbf{w}^t)\|^2 \underbrace{- \frac{\eta K}{2N}\sum_{e=1}^E\mathbb{E}_{\mathcal{H}}\Big[\Big\|\sum_{n=1}^N p^t_n\nabla F_n(\mathbf{w}^{\tau^t_n}_{n,e})\Big\|^2\Big]}_{T_4},
\end{aligned} \tag{7}$$

where $(A.)$ follows the unbiasedness of g_n and $(B.)$ is because $\langle a,b\rangle = \frac{1}{2}(\|a\|^2 + \|b\|^2 - \|a-b\|^2)$. Due to $\nabla f(\mathbf{w}^t) = \sum_{n=1}^N p^t_n\nabla F_n(\mathbf{w}^t)$ in FL, $T_3 \le N\sum_{n=1}^N(p^t_n)^2\|\nabla F_n(\mathbf{w}^t) - \nabla F_n(\mathbf{w}^{\tau^t_n}_{n,e})\|^2$. $(C.)$ is due to $\mathbb{E}[T_3] \le 6L^2\eta^2E(KB+1)(\sigma_l^2 + \sigma_g^2 + G^2)$, which is based on the analysis of *FedBuff* [8].

Now we move to derive the upper bound of $\mathbb{E}[T_2]$:

$$\begin{aligned}
\mathbb{E}[T_2] &\overset{(A.)}{\le} \frac{LK}{2}\mathbb{E}\Big[\sum_{n\in\mathcal{A}^t}\|p^t_n\Delta^t_n\|^2\Big] \le \frac{LK}{2}\mathbb{E}\Big[\sum_{n\in\mathcal{A}^t}\|p^t_n\bar{\Delta}^t_n\|^2\Big] \\
&\overset{(B.)}{=} \frac{LK\eta^2}{2}\mathbb{E}\Big[\sum_{n\in\mathcal{A}^t}(p^t_n)^2\sum_{e=1}^E\|g_n(\mathbf{w}^{\tau^t_n}_{n,e}) - \nabla F_n(\mathbf{w}^{\tau^t_n}_{n,e})\|^2\Big] \\
&\quad + \frac{LK\eta^2}{2}\mathbb{E}\Big[\sum_{n\in\mathcal{A}^t}\sum_{e=1}^E\|p^t_n\nabla F_n(\mathbf{w}^{\tau^t_n}_{n,e})\|^2\Big] \\
&\le \frac{LKE\eta^2\sigma_l^2}{2} + \underbrace{\frac{LK^2\eta^2}{2N}\sum_{e=1}^E\sum_{n=1}^N\mathbb{E}_{\mathcal{H}}\Big[\|\nabla F_n(\mathbf{w}^{\tau^t_n}_{n,e})\|^2\Big]}_{T_5},
\end{aligned} \tag{8}$$

where $(A.)$ is due to $\|\sum_{k=1}^K a_k\|^2 \le K\sum_{k=1}^K\|a_k\|^2$, and $(B.)$ holds because local gradient g_n is unbiased.

Combining Eq. (6), Eq. (7) and Eq. (8) results in $\mathbb{E}\big[f(\mathbf{w}^{t+1})\big] \le \mathbb{E}\big[f(\mathbf{w}^t)\big] - \frac{\eta KE}{2N}\|\nabla f(\mathbf{w}^t)\|^2 + \frac{LKE\eta^2\sigma_l^2}{2} +$

$\frac{3L^2\eta^3 EK}{N}(KB+1)(\sigma_l^2+\sigma_g^2+G^2)+T_4+T_5$. Summing up this inequality from $t=0$ to $T-1$ with $\eta \leq \frac{1}{KL}$ to ensure that $T_4+T_5 \leq 0$ yields Eq. (5). $\square$

IV. BLADE: IMPLEMENTATION AND EVALUATION

A. Implementation and Preparation

We have implemented BLADE in PLATO (https://github.com/TL-System/plato), a new open-source FL research framework developed from scratch. Designed for *scalable*, *reproducible*, and *extensible* FL research, PLATO can fairly and accurately compare different FL mechanisms in the same real-world or emulated environment.

Improving scalability. To scale up the number of clients with limited CPU and GPU (CUDA) memory, our implementation runs a client's training loop *in its own process*, so that memory is guaranteed to be released after the process completes. The number of launched processes depends only on resource availability and PLATO can automatically use all the available GPUs. With a sufficient amount of time, PLATO is scalable to an *unlimited* number of clients.

Improving reproducibility. For fair comparisons across different FL mechanisms, it is critically important to improve the reproducibility by seeding, saving, and restoring random number generators, which are used for sampling participating clients and local datasets. PLATO can not only specify random seeds, but also eliminate effects of third-party frameworks with `random.getstate()` and `random.setstate()`.

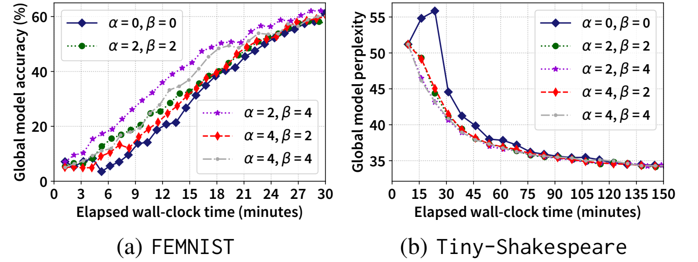
Datasets and models. PLATO provides ready access to a wide variety of existing models beyond image classification. For instance, it supports *HuggingFace Transformers*, which provides thousands of pretrained models to perform tasks on different modalities such as text, vision, and audio. With such ready availability of datasets and models in PLATO, we evaluated BLADE with six FL training tasks. Five are image classification tasks, including the `LeNet-5` model with the `MNIST`, `EMNIST`, and `FEMNIST` datasets, the `ResNet-18` model with the `CIFAR-10` dataset, and the `VGG-16` model with the `CINIC-10` dataset. We also conducted a more complex language modeling task to train a distilled variant of the `GPT-2` model with the `Tiny-Shakespeare` dataset. Table I lists important parameters used across tasks, and the parameters of local training are the same as motivational experiments in II.

TABLE I: Six training tasks we tested: parameter settings.

Parameter	`MNIST` `EMNIST`	`FEMNIST`	`CIFAR-10` `CINIC-10`	`Tiny-Shakespeare`
Total clients	1000	3597	1000	200
C	100	100	30	20
K	50	50	20	10

BLADE was compared with *FedBuff* and *FedBuff + Oort + FedSCR* in image classification tasks. Since *FedSCR* was designed solely for image classification models, we compared BLADE with *FedBuff* and *FedBuff + Oort* for the language modeling task. *FedAvg* is added as a synchronous FL baseline.

Hyperparameter sweep. We performed a hyperparameter sweep for α in Eq. (1) and β in Eq. (4). Fig. 3 illustrates a comparison of several representative pairs of α and β in two tasks as examples. Given their clear performance advantages over other pairs, $\alpha=2$ and $\beta=4$ were set as the default.

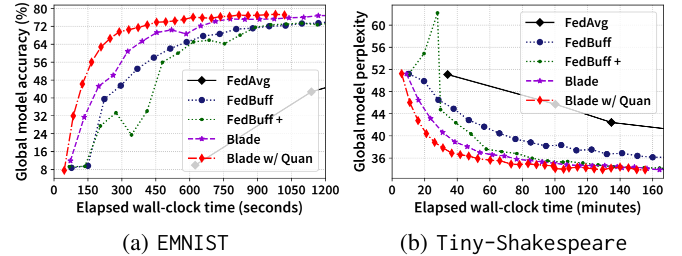


(a) `FEMNIST` (b) `Tiny-Shakespeare`

Fig. 3: Evaluating different pairs of hyperparameters α and β.

B. Performance Evaluation

Evaluating BLADE without quantization. We first evaluated BLADE with its three basic components. Fig. 4 shows the results of the `EMNIST` and `Tiny-Shakespeare` datasets as examples. The label *FedBuff* + represents *FedBuff + Oort + FedSCR* in the image classification tasks, and *FedBuff + Oort* for the task with the `Tiny-Shakespeare` dataset.



(a) `EMNIST` (b) `Tiny-Shakespeare`

Fig. 4: Performance comparison: BLADE vs. its competitors.

Overall, BLADE clearly outperformed all its competitors across all six tasks. It consistently converged to the highest validation accuracy, or the lowest perplexity. As BLADE used its client selector to replace conventional random selection, it is not surprising that its accuracy was higher than *FedBuff*. Moreover, with adaptive pruning, BLADE incurred a shorter transmission time for each communication round, which amplified its advantage. However, coupled with client selection from *Oort* and pruning from *FedSCR*, *FedBuff* performed only slightly better or even worse than vanilla *FedBuff* when training with the `EMNIST` dataset, which substantiates our design philosophy that client staleness should be considered.

Evaluating BLADE with quantization. Fig. 4 illustrates the sample results of applying quantization to BLADE. Numerical results of elapsed wall-clock training times to reach a target accuracy (or perplexity) are listed in Table II. The numbers in brackets are the percentages of reduced training time compared with *FedBuff + Oort (+ FedSCR)*. With quantization, BLADE reduced even more training time.

Which component contributes the most in BLADE? In our experiments, BLADE showed its superiority in all six tasks. Since it has three basic components and one optional component (quantization), we are intrigued by the question: which component contributes the most to its superior performance?

TABLE II: Elapsed wall-clock training time to reach a target accuracy or target perplexity.

Settings			Wall-clock training time (minutes)				
Dataset	Model	Target accuracy or target perplexity	*FedAvg*	*FedBuff*	*FedBuff* + *Oort* (+ *FedSCR*)	BLADE	BLADE w/ quantization
MNIST	LeNet-5	94%	36.50	9.44	6.88	5.11 (25.73%)	2.71 (60.61%)
EMNIST	LeNet-5	75%	106.60	26.75	21.67	12.99 (40.06%)	9.58 (55.79%)
FEMNIST	LeNet-5	60%	235.93	45.16	38.95	27.29 (29.94%)	16.63 (57.30%)
CIFAR-10	ResNet-18	80%	412.63	87.07	55.92	49.12 (12.16%)	34.66 (38.02%)
CINIC-10	VGG-16	55%	352.16	98.23	65.53	33.92 (48.24%)	23.17 (64.64%)
Tiny-Shakespeare	DistilGPT2	35	844.42	235.7	126.49	107.27 (15.19%)	73.08 (42.22%)

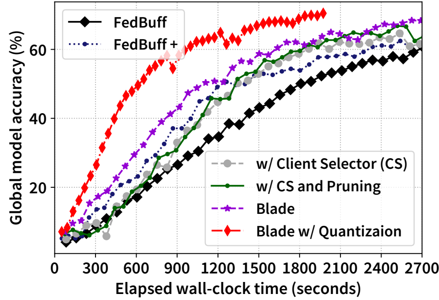


Fig. 5: Effects of different components in BLADE.

To answer this question, we used FEMNIST as an example and plotted Fig. 5. The client selector itself improved the performance by a substantial margin over *FedBuff*, and beat *FedBuff* + *Oort* + *FedSCR*. However, pruning only provided a minor overall improvement. The update aggregator (the purple line with star markers) significantly improved performance and converged to higher accuracy. Yet, quantization contributed the most. To reach 60% accuracy, quantizing 32-bit floating point numbers to 16-bit ones reduced training time by 29.4%.

V. CONCLUDING REMARKS

To minimize the wall-clock training time of asynchronous FL, we propose a staleness-aware framework, BLADE, which consolidates new client selection, adaptive pruning, quantization, and update aggregation mechanisms to improve the performance of every important aspect of a training process as much as possible. In BLADE, the server selects clients based on their latest updates' quality and staleness. After finishing local training, a selected client prunes and quantizes its update based on its estimated staleness. When the server aggregates updates by weighted averaging, the aggregation weight of each update is determined by its estimated quality and staleness. With a scalable implementation and reproducible experiments, we have demonstrated convincing evidence that BLADE consistently outperformed its state-of-the-art alternatives with respect to the wall-clock training time used for converging to a target accuracy over a variety of tasks.

REFERENCES

[1] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," in *Proc. 20th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2017.

[2] K. A. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, and et al, "Towards Federated Learning at Scale: System Design," in *Proc. Machine Learning and Systems 2019 (MLSys)*, 2019.

[3] C. Yang, Q. Wang, M. Xu, Z. Chen, K. Bian, Y. Liu, and X. Liu, "Characterizing Impacts of Heterogeneity in Federated Learning upon Large-Scale Smartphone Data," in *Proc. The Web Conference*, 2021.

[4] C. Xie, S. Koyejo, and I. Gupta, "Asynchronous Federated Optimization," in *Proc. NeurIPS Workshop on Optimization for Machine Learning (OPT)*, 2020.

[5] Y. Chen, Y. Ning, M. Slawski, and H. Rangwala, "Asynchronous Online Federated Learning for Edge Devices with Non-IID Data," in *Proc. 2020 IEEE International Conference on Big Data*. IEEE, 2020.

[6] Y. Chen, X. Sun, and Y. Jin, "Communication-Efficient Federated Deep Learning With Layerwise Asynchronous Model Update and Temporally Weighted Aggregation," *IEEE Trans. Neural Networks and Learning Systems*, vol. 31, no. 10, pp. 4229–4238, 2020.

[7] 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 Trans. Computers*, vol. 70, no. 5, pp. 655–668, 2021.

[8] J. Nguyen, K. Malik, H. Zhan, A. Yousefpour, M. Rabbat, M. Malek, and D. Huba, "Federated Learning with Buffered Asynchronous Aggregation," in *Proc. 25th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2022.

[9] N. Su and B. Li, "How Asynchronous can Federated Learning Be?" in *Proc. 29th IEEE/ACM International Symposium on Quality of Service (IWQoS)*, 2022.

[10] X. Li, K. Huang, W. Yang, S. Wang, and Z. Zhang, "On the Convergence of FedAvg on Non-IID Data," in *Proc. 8th International Conference on Learning Representations (ICLR)*, 2020.

[11] K. Hsieh, A. Phanishayee, O. Mutlu, and P. B. Gibbons, "The Non-IID Data Quagmire of Decentralized Machine Learning," in *Proc. 37th International Conference on Machine Learning (ICML)*, 2020.

[12] F. Lai, X. Zhu, H. V. Madhyastha, and M. Chowdhury, "Oort: Efficient Federated Learning via Guided Participant Selection," in *15th USENIX Symposium on Operating Systems Design and Implementation*, 2021.

[13] A. Li, J. Sun, P. Li, Y. Pu, H. Li, and Y. Chen, "Hermes: An Efficient Federated Learning Framework for Heterogeneous Mobile Clients," in *Proc. 27th Annual International Conference on Mobile Computing and Networking (MobiCom)*, 2021.

[14] X. Wu, X. Yao, and C. Wang, "FedSCR: Structure-Based Communication Reduction for Federated Learning," *IEEE Trans. Parallel Distributed Syst.*, vol. 32, no. 7, pp. 1565–1577, 2021.

[15] A. Reisizadeh, A. Mokhtari, H. Hassani, A. Jadbabaie, and R. Pedarsani, "FedPAQ: A Communication-Efficient Federated Learning Method with Periodic Averaging and Quantization," in *Proc. 23rd International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2020.

[16] H. Wu and P. Wang, "Fast-Convergent Federated Learning with Adaptive Weighting," *IEEE Trans. Cognitive Communications and Networking*, 2021.

[17] S. Ji, S. Pan, G. Long, X. Li, J. Jiang, and Z. Huang, "Learning Private Neural Language Modeling with Attentive Aggregation," in *International Joint Conference on Neural Networks*. IEEE, 2019.

[18] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both Weights and Connections for Efficient Neural Network," in *Proc. 29th International Conference on Neural Information Processing Systems (NeurIPS)*, 2015.

[19] Y. Collet, N. Terrell, and P. Skibiński, "Zstandard," 2021. [Online]. Available: https://github.com/facebook/zstd

[20] S. J. Reddi, Z. Charles, M. Zaheer, Z. Garrett, K. Rush, J. Konečný, S. Kumar, and H. B. McMahan, "Adaptive Federated Optimization," in *Proc. 9th International Conference on Learning Representations*, 2021.

[21] 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 *Proc. 33rd AAAI Conference on Artificial Intelligence (AAAI)*, 2019.