PAS: Towards Accurate and Efficient Federated Learning with Parameter-Adaptive Synchronization

Zuo Gan[1], Chen Chen[1], Jiayi Zhang[1], Gaoxiong Zeng[2*], Yifei Zhu[1], Jieru Zhao[1], Quan Chen[1], Minyi Guo[1]
[1]Shanghai Jiao Tong University [2]Huawei

***Abstract*—Federated Learning (FL) is a distributed paradigm that supports collaborated model training while preserving data privacy, where clients periodically synchronize their local gradients once after multiple local iterations. Due to non-uniform data distribution and poor network condition, FL processes often suffer degraded training accuracy and efficiency. In this work, we analyze the microscopic parameter variation behaviors in FL, and find that an effective method to improve FL accuracy is to switch to more frequent synchronization at proper moments. Moreover, such moments can be detected from gradient characteristics, and are *heterogeneous* across different parameters. Motivated by such observations, we propose Parameter-Adaptive Synchronization (PAS), a FL scheme that adaptively tunes the synchronization period for each scalar parameter. The benefits of PAS are two-fold: By switching to more frequent synchronization when necessary, we can improve the FL training accuracy; by synchronizing different parameters independently, we can enable communication-computation overlapping and enhance the network utilization. We implemented PAS atop PyTorch, and extensive experiments show that it can substantially improve FL performance in both accuracy and communication efficiency.**

I. Introduction

In recent years, Federated Learning (FL) [1], [2] is booming as an effective technique that allows multiple clients (e.g., mobile devices) to jointly train a machine learning model without disclosing their local private data. In typical FL scenarios, the network connection between the edge devices and the FL server is often of low quality; to improve training efficiency, FedAvg [2] has become the *de facto* client coordination mechanism for FL. Under FedAvg, each client pulls the latest model and trains it for *multiple* local iterations before reporting the updates, which procedure is called a *round*.

While FL has proven effective for a series of real-world scenarios [3], [4], [5], it has long been plagued with two kinds of challenges: *statistical* challenge (non-IID data) and *system* challenge (incapable hardware resources). Due to the privacy constraints, the local datasets on different FL clients are often of heterogeneous distributions, which may severely hurt the training accuracy [6], [7]. Both existing works [2], [6], [8] and our empirical study show that model accuracy would be lower with less frequent synchronization. Meanwhile, model synchronization over low-bandwidth connections is a severe bottleneck for FL [9], [10], and more frequent synchronization would increase the communication delay. Therefore, there is a trade-off in setting up the *synchronization period* (i.e., the number of local iterations in each round).

*This is non-Huawei achievement.

Our objective here is to jointly improve the accuracy and efficiency of FL. While existing works [11], [12], [13], [14], [15], [9] have explored how to tune the accuracy-efficiency trade-off by properly setting up the synchronization period, they implicitly assume that all the model parameters be synchronized *at the same pace*—parameter synchronization can only be kicked off after the completion of all the local iterations in each round. However, it remains largely uncharted whether different parameters of a model do need heterogeneous synchronization periods, and a fixed setup may be too rigid, yielding compromised accuracy and efficiency.

In this work, we study the impact of FL synchronization period on the parameter updating process, seeking to quantitatively understand the preference of an individual parameter on the synchronization period setup. Note that to behave well in both accuracy and efficiency, a practical strategy is to dynamically tuning the synchronization frequency at runtime. By making fine-grained analysis on the variation of each parameter during the FL process, we find a phenomena called *gradient bifurcation* that can signal the appropriate frequency tuning moment of each parameter. Most importantly, such frequency-tuning moments are heterogeneous among different parameters (even for those in the same layer), suggesting that we need to independently tune the synchronization period for each parameter.

Motivated by the above study, in this paper we propose a FL scheme called *Parameter-Adaptive Synchronization* (PAS), which works by adaptively tuning the *synchronization period* at the *per-parameter* (scalar) granularity. Compared with exiting frequency setting up strategies, PAS has two distinct advantages. First, fine-grained frequency control can attain better adaptivity and improve the training accuracy. Second, by starting synchronizing different parameters at different times, PAS can overlap the communication of some parameters with the computation of the others, which is a novel *inter-iteration* communication-computation overlapping technique for FL that can effectively mitigate the network bottleneck.

To realize PAS, an immediate question is how to tune the synchronization period of each parameter. Based on gradient characteristics, we propose a metric called *Gradient Consistent Rate*, which can quantify the level of gradient bifurcation in a practical manner. To be specific, we adopt the exponential moving average method to handle mini-batch randomness, and we also adopt the pooling method to tackle the issue of large-scale yet dynamic client participation. Based on that metric, we devise a simple yet effective heuristic: Once the gradient

consistent rate of a parameter stabilizes around zero (indicating maximized gradient bifurcation level), we scale down that parameter's synchronization period by a predefined constant.

Meanwhile, in the engineering aspect, we note that special PAS designs are needed to guarantee communication speedup. If we synchronize each parameter independently, the parameters with shorter synchronization periods would be synchronized for more times until all the other parameters converge. That is, although communication can be overlapped with computation, the total transmission amount is nonetheless increased, canceling out the potential communication benefit. To tackle that, we propose *Eager Synchronization with Uniform Round Pace*: Eagerly aggregate the updates for parameters with shorter synchronization periods, and after aggregation, let those early-bird parameters wait for the others instead of directly proceed to the next round. In this way, we can avoid inflating the total communication amount and guarantee the improvement on training efficiency.

We have implemented PAS atop PyTorch, and have conducted experiments on a 100-node EC2 cluster emulating realistic FL scenarios. The results confirm that PAS can effectively improve FL in both accuracy and efficiency. For example, after training LSTM for 6 hours, the model accuracy with PAS is 12.25% higher than FedAvg and 6.50% higher than the second best strategy. Meanwhile, when training ResNet-20, PAS can speed up the completion of each round by over 50%. We also confirm that the performance of PAS is robust to hyper-parameter variations, and the overhead it incurs is negligible compared to the performance benefits.

II. Research Background

A. The Basics of Federated Learning

Federated Learning (FL) [1], [2] is a special form of distributed model training that places data privacy at the first priority. It allows clients to refine a model copy locally and aggregate the updates (instead of the data samples) to the central server. The network capacity of typical FL clients (e.g., IoT devices or cellphones) is quite limited; to reduce the communication overhead, *FedAvg* [2] has become a norm for FL practices. It enforces each client to perform *multiple* local iterations before global aggregation.

To put it formally, we let τ represent the *synchronization period* (i.e., the number of local iterations each client trains within a round), let ω_k^i be the model parameters at iteration k for client-i ($i = 1, 2, ..., N$), and let $F^i(\omega)$ be the local loss function. Then ω_k^i is updated as:

$$\omega_k^i = \begin{cases} \omega_{k-1}^i - \eta \nabla F^i(\omega_{k-1}^i), & \text{if } k \bmod \tau \neq 0, \\ \frac{1}{N}\sum_{i=1}^{N}[\omega_{k-1}^i - \eta \nabla F^i(\omega_{k-1}^i)], & \text{otherwise.} \end{cases} \quad (1)$$

We summarize the main notations of this paper in Table I. While FL has attained remarkable success (as reported by Google [4] and Meta [5]), it still suffers compromised accuracy and efficiency.

In the accuracy aspect, in typical FL scenarios, the local datasets on different clients are *not* independently and identically distributed (i.e., being non-IID). Existing works have shown that, FL with non-IID data may suffer substantial accuracy degradation [6], [7], [16]. Our measurements also confirm that there is a strong correlation between the model convergence accuracy and the synchronization period τ: The smaller τ, the higher model accuracy.

Meanwhile, in the efficiency aspect, given the bandwidth limitation of typical FL devices, model synchronization is often a severe performance bottleneck. Our measurements show that, when training ResNet-20 with a network bandwidth of 1Mbps, model synchronization takes nearly a half of the per-round time even with $\tau = 200$.

To summarize, there is a clear trade-off when setting up the synchronization period: A smaller τ can yield a higher model accuracy—yet at the cost of larger communication cost. It is indeed a challenging task to attain both high accuracy and high communication efficiency at the same time.

TABLE I: Summary of Main Notations

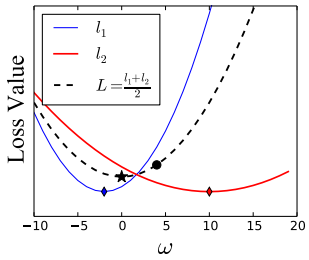
N	Number of clients	γ	Divisor to scale down τ
ω	Parameters of global model	$\mathcal{R}$	Gradient Consistent Rate
$\omega^\star$	The optimal value of ω	X_Δ	Indices of τ to be tuned
ω_k^i	Local model on client-i at iteration k	$\omega[\{x\}]$	A parameter subset indexed by $\{x\}$
$F(\omega)$	Global loss function	η	Learning rate
$F^i(\omega)$	Loss function on client-i	θ	EMA smoothing factor
g^i	Local gradient from client-i	P	Positive gradient component
τ	Synchronization period	N	Negative gradient component

B. Prior Arts on Synchronization Period Setup

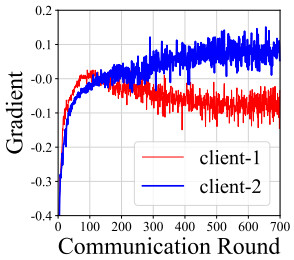
In the literature, a series of methods have already been proposed to set the synchronization frequency for better accuracy and efficiency performance.

Some research works [11], [12], [13] focus on setting up a *static* synchronization frequency with the objective of maximizing the overall accuracy under a given resource budget. For example, Wang et al. [12] proposed Adaptive Federated Learning, which directly estimates the best τ with a formula related to the loss function characteristics (e.g., Lipschitz constant, smoothness constant) and gradient divergence bound. However, in reality it is hard to obtain those ground-truth knowledge a priori, and in many FL practices [4], [5], there is no explicit resource budget as a hard constraint.

In the meantime, some other works [14], [15], [9] propose to dynamically adjust the synchronization frequency at runtime: Setting a large τ in the beginning for high communication efficiency, and switching to a small τ later for high accuracy. For example, AdaComm [14] divides the training process into short intervals and estimates the best frequency in each interval based on the instantaneous training loss: $\tau_k = \left\lceil \sqrt{\frac{F(\omega_k)}{F(\omega_0)}} \tau_0 \right\rceil$, where τ_0 is the initial synchronization period. Compared to directly setting up a static τ assuming various ground-truth knowledge, dynamically changing τ at runtime based on the instantaneous training status is more practical, which is more of our focus in this paper.



(a) If clients' local loss functions (l_1 and l_2) differ in curvature, the average of local minimums (●) is inconsistent with the global minimum (⋆ for L).



(b) When training LSTM model with non-IID data, the local gradients from different clients for a randomly chosen parameter gradually bifurcate during training.

Fig. 1: With insufficient synchronization, the global model may stagnate with suboptimal parameters, where the local gradients from different clients counteract with each other.

However, the above works treat all the model parameters as an indivisible synchronization unit[1], and it remains unclear whether different parameters prefer different synchronization periods during the FL process. Were that true, we can have two remarkable benefits. First, by respectively setting up the synchronization frequency for each parameter, we can provision better adaptivity and attain better accuracy. Second, since parameters with smaller τ can be synchronized while the others are still in local computations, we can enable *inter-iteration computation-communication overlapping* and mitigate the communication bottleneck.

In the next section, we will analyze the microscopic parameter variations during the FL process, seeking to quantify the frequency-tuning preference of different parameters.

III. Motivating Explorations

In this section, we explore whether all the model parameters share the same preference on synchronization period (τ). To acquire the per-parameter preference, we need to find out a fine-grained metric to sense the instantaneous training status of each parameter. We find that *gradients* provides an appropriate observation angle to monitor the training status. During the FL process, clients continuously calculate the gradients for each parameter and report the updates to the server; compared with loss or accuracy information [12], [14], gradient-based information is more privacy-friendly and can naturally reflect fine-grained training status for each corresponded parameter.

Gradient information can help instruct frequency tuning. When training with non-IID data, the local loss functions of different clients would be heterogeneous, i.e., having inconsistent local minimums; during the local iterations, each client is essentially refining the model parameters towards its local optimum [6], [8], which is inaccurate for global optimization—especially in the later iterations of a round. In particular, when synchronization is conducted late, clients may already reach their local minimums; worse, since FL clients often have different loss function curvatures, the average of their local minimums (obtained after FedAvg aggregation) is not the global minimum, as described in Fig. 1a. Such a process would repeat in each subsequent round, making the global model stagnate with sub-optimal accuracy performance. At such moment, it would be a waste of resources to continue training under the current setup. To attain higher accuracy, we need to increase the synchronization frequency (i.e., reduce τ) to mitigate the divergence error of clients' local updates.

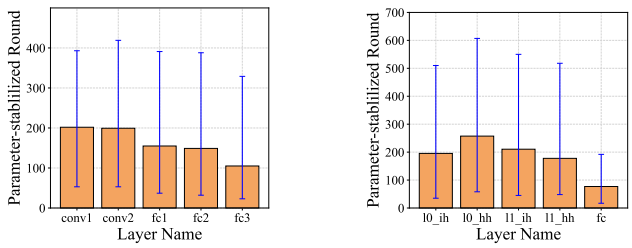
We further note that the above frequency-tuning moment can be probed by observing gradient characteristics. When sub-optimal stagnation happens, the accumulated gradients from different clients would exhibit a conflicting pattern. To confirm that, we train the LSTM model above two clients with non-IID data (each hosting one half of the KWS dataset, which is described in Sec. VI-A). Fig. 1b depicts—for a parameter chosen randomly—the *accumulated* local gradients of each client during training: In the initial stage, the two clients' local gradients are consistent, because the parameter is far away from their local optimums and both clients make similar refinement; as training proceeds, the gradients gradually bifurcate across the zero-value axis, because the parameter may now be between the two clients' local optimums and the clients make conflicting refinement. We call this phenomena *gradient bifurcation*, which can work as a signal to tune the synchronization period. Next, we make quantitative analysis on whether all the model parameters share similar gradient bifurcation patterns.

Different parameters prefer frequency tuning at different moments. To quantify the level of gradient bifurcation, we propose an intuitive metric called *Gradient Consistent Rate* ($\mathcal{R}$), which is defined at per-parameter granularity. Let $\{g^1[x], g^2[x], ..., g^N[x]\}$ be the gradients[2] for the parameter with index x from N clients, then its gradient consistent rate can be calculated as: $\mathcal{R}[x] = \frac{|\sum_{i=1}^{N} g^i[x]|}{\sum_{i=1}^{N} |g^i[x]|}$. Obviously, this metric would be almost 0 if there exist strong conflicts among clients' local gradients. Thus, we judge that a parameter has stagnated due to gradient counteraction when its gradient consistent rate decreases below a given threshold close to 0, and we call that time *gradient-counteracting moment*.

Further, we train a CNN model and a LSTM model (described in Sec. VI-A) with the same non-IID setup as in Fig. 1b, and measure the gradient-counteracting moments of all the parameters, the results shown in Fig. 2. In particular, we calculate the average gradient-counteracting moment (with a threshold of $\mathcal{R}[x] < 0.1$) respectively for parameters within each layer, and the error bars show their 5% and

[1]A recent work, FedLAMA [17], allows parameters to be synchronized under two different frequencies—those layers that are insignificant to model accuracy (i.e., with a smaller discrepancy between the global model and the local model) but incur a large communication cost would be synchronized less frequently. Yet, each layer synchronized under FedLAMA are still communicated in one batch, suffering essentially the same problem as discussed above. We will compare our solution also with FedLAMA in Sec. VI.

[2]For simplicity, hereafter we use "gradient" to represent the accumulated local updates over all the iterations of a round. Moreover, while the model parameters or gradients have multiple tensors, for easy manipulation we treat them as a single vector, which can be efficiently realized by vectorizing (with `Tensor.view(-1)`) and concatenating the original tensors.



(a) CNN (b) LSTM

Fig. 2: Average *gradient-counteracting moment* (i.e., round id when gradient consistent rate decreases below a threshold) for each layer. The error bars show the $5^{th}/95^{th}$ percentiles.

95% percentiles. From Fig. 2, we can learn that gradient bifurcation is quite common when conducting FL with non-IID data. More importantly, *gradients for different parameters do not bifurcate at the same pace*: In Fig. 2, the average gradient-counteracting moments for different layers are not the same. Moreover, as implied by the error bars, there exist huge gaps among the gradient-counteracting moments even for the parameters *within the same layer*. Such *intra-layer heterogeneity* is indeed reasonable as echoed by existing works [18], [19]: Some features may be easier to learn, and the corresponded parameters would converge faster.

In a nutshell, as indicated by gradient bifurcation characteristics, different parameters do prefer heterogeneous frequency tuning moments. In the remaining part of this paper, we will show how to leverage such property to jointly enhance the accuracy and efficiency performance of FL.

IV. Parameter Adaptive Synchronization

Given the above explorations, in this paper we propose *Parameter-Adaptive Synchronization* (PAS), which adaptively tunes the synchronization period for each parameter based on the runtime gradient behavior.

A. Tuning Synchronization Period

Recall that in Sec. III, to quantify the level of gradient bifurcation for a given parameter, we have already proposed a metric called *Gradient Conflicting Rate*. However, it is not practical given the distinct challenges of real-world FL. First, due to mini-batch randomness, the local gradients on each client would fluctuate drastically, as can be seen in Fig. 1b. Meanwhile, a client may join or leave the FL process at random time, and only a portion of gradients that are reported the earliest are collected by the FL server [3], [4], [20]. Moreover, quantifying gradient bifurcation level requires tracking the historical gradients of every client; yet, real-world FL may involve hundreds or thousands of clients [3], [4], [5], and our metric should scale well in computing or storage overhead when facing massive clients.

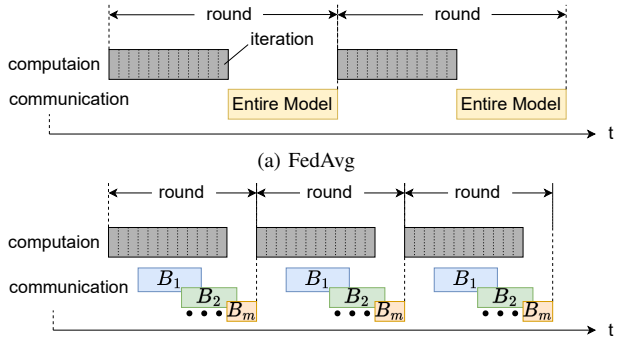
To tackle the above challenges, we extend the metric definition of gradient consistent rate with two additional techniques: *smoothing* and *pooling*.

First, to address statistical instability, we smooth the raw gradients with their historical values. Nonetheless, a window-based smoothing method requires maintaining multiple model snapshots from previous rounds, which would consume a large storage space. To maintain low storage overhead, we instead calculate the gradients' *exponential moving average* (EMA). Let $\langle\cdot\rangle_\theta$ denote the operation to calculate exponential moving average with a decay factor θ, and g_r^i be the local gradient of client-i in the r-th round, then we maintain $\tilde{g}_r^i = \langle g_r^i \rangle_\theta = \theta * \tilde{g}_{r-1}^i + (1-\theta) * g_r^i$ as a smoothed version of g_r^i.

Moreover, we note that only the smoothing method is not enough: Due to system (participant) instability, it is infeasible to persistently track the gradients of each client; meanwhile, given the large client quantity, it is not memory-efficient to record $\tilde{g}_r^i$ for each client. To tackle such problems, we further propose *bilateral gradient pooling*. That is, the FL server only maintains two values for each parameter: $\tilde{P}_r[x] = \langle \sum_i \mathbf{Relu}(g_r^i[x]) \rangle_\theta$—to add up the *positive* gradients, and $\tilde{N}_r[x] = \langle \sum_i \text{-}\mathbf{Relu}(\text{-}g_r^i[x]) \rangle_\theta$—to add up the *negative* ones. Here $\tilde{P}_r[x]$ collects the EMA values of all the positive gradients across all the clients, and $\tilde{N}_r[x]$ collects the EMA values of all the negative ones (we use the **Relu** operation to filter out the positive/negative components). When the local gradients for a parameter bifurcate, the two gradients also bifurcate. This way, we can get a stable gradient pattern despite with unstable and large-scale client participation. We further define *gradient consistent rate* for parameter index x at round r as: $\mathcal{R}_r[x] = \frac{|\tilde{P}_r[x]+\tilde{N}_r[x]|}{|\tilde{P}_r[x]|+|\tilde{N}_r[x]|}$. With such a metric definition, we can effectively quantify the gradient bifurcation level of each parameter with low cost and high robustness.

Algorithm to Tune Synchronization Period. Ideally, $\mathcal{R}_r[x]$ shall decrease to 0 when that parameter's training stagnates, yet it may not be so in practice: First, the EMA smoothing method cannot fundamentally eliminate the impact of mini-batch randomness; second, deep neural networks may exhibit irregular landscapes like *flat minima* [21], where some parameters conduct random walk even after model convergence. To avoid the disturbance of such mini-batch randomness or irregular landscape, we diagnose training stagnation not by the absolute value of $\mathcal{R}_r[x]$ —but by its stabilizing behavior. That is, if $\mathcal{R}_r[x]$ stops decreasing, we can judge that it reaches gradient-counteracting moment, and the synchronization period of that parameter shall then be adjusted.

Further, how to adjust the synchronization period of a parameter when its gradient-counteracting moment arrives? For clarity, we use $\vec{\tau}$ to represent the synchronization periods of all the parameters. To make our solution generally applicable, we employ a simple-yet-effective heuristic to update $\vec{\tau}$: Once a parameter reaches the gradient-counteracting moment, we scale down its synchronization period $\vec{\tau}[x]$ by a constant γ (typically set to 2). If training a parameter for less iterations in a round, its aggregated gradient would be more accurate for global optimum. Besides, it can be expected that the parameter would stagnate again under the new synchronization period—a moment that will also be detected with the $\mathcal{R}_r[x]$ metric,



(a) FedAvg

(b) PAS

Fig. 3: In FedAvg, model synchronization is conducted after all the local iterations; yet with PAS, parameters with shorter synchronization periods can be immediately transferred, mitigating the communication bottleneck in the critical path.

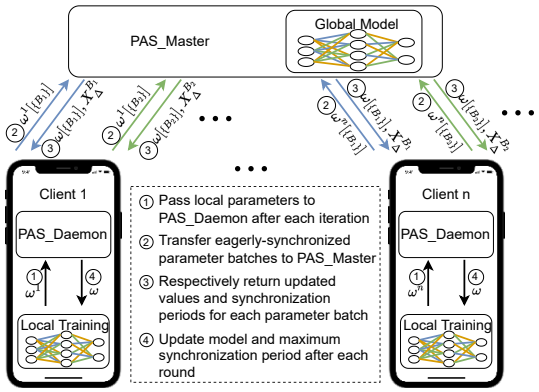


Fig. 4: Architecture and workflow of PAS.

and another τ tuning action would then be triggered.

B. Eager Synchronization with Uniform Round Pace

When tuning $\vec{\tau}[x]$ based on $\mathcal{R}_r[x]$, as implicated by Sec. III, different parameters may be configured with different synchronization periods. Then, how to conduct aggregation for such parameters with heterogeneous synchronization periods? Intuitively, when any parameter finishes its local training iterations (specified by $\vec{\tau}[x]$), we can immediately aggregate its gradient and start its next training round. However, while this intuitive method can enable communication-computation overlap, it has a fatal deficiency. In fact, a model is deemed converged only when all of its parameters converge—this is essentially a straggler effect, meaning that parameters converging the most slowly would dominate the model training time. In that case, if we synchronize the parameters with shorter synchronization periods independently, they would be synchronized for more times before the ultimate model convergence, and totally more network resources would be consumed. That is, although the intuitive method can increase link utilization, it—on the other hand—would inflate the total communication demand, canceling out the potential efficiency benefit.

Therefore, to aggregate model parameters with heterogeneous synchronization periods, we introduce two principles:

1) Uniform Round Pace. In our solution, all the parameters—irrespective of their synchronization periods—are always at the same round. That is, if some parameters with smaller synchronization periods finish their local iterations, instead of immediately starting their next round, we stop training them (by not recording their updates) and have them wait for the other parameters until all the local computations of that round complete. This way, while attaining higher accuracy by shortening the synchronization periods of the *early-bird* parameters (i.e., those parameters with shorter synchronization periods), we do not increase the total communication amount.

2) Eager Synchronization. When the early-bird parameters are waiting for others, we can synchronize them in the background. As shown in Fig. 3, in original FedAvg, all the model parameters synchronize at the end of each round, incurring long delay under the limited network bandwidth. Yet, for those early-bird parameters, there is no need to delay their communication to the last minute. In contrast, with our eager-synchronization strategy, we can communicate their updates to the FL server immediately after finishing their local training iterations. This way, their communication delay can be removed out of the critical path—hidden by the other parameters' local computations, and it can be expected that the per-round time can be remarkably reduced.

Note that the above *inter-iteration* communication-computation overlapping method opens up a new optimization domain for FL. In the literature, computation-communication overlapping can effectively accelerate distributed model training in cluster scenarios [22], [23], yet it occurs at *intra-iteration* level, i.e., overlapping the communication and computation across different *layers* in forward/backward propagation within an iteration. For FL scenarios, intra-iteration overlapping occurs only in the last iteration of each round, and due to the poor network condition, communication can hardly be overlapped by the computation of an iteration. In that sense, *inter-iteration* overlapping is much more effective than *intra-iteration* overlapping for FL scenarios.

Remarks on Convergence Analysis. Some works [24], [8] have proved that, for smooth and strongly-convex models, FedAvg can attain a linear convergence rate $\mathcal{O}(\frac{1}{T})$, where T is the number of SGD iterations. Regarding PAS, we can prove that it also yields a linear model convergence rate (with even better coefficients); due to space limitation, we only describe the main proof sketch here. In existing proofs of FedAvg [24], [8], to bound $\mathbb{E}[F(\omega_T)] - F(\omega^\star)$, a key step is to bound the instantaneous *divergence* among clients' local parameters (i.e., $\frac{1}{N}\sum_{i=1}^{N}\mathbb{E} \parallel \bar{\omega}_k - \omega_k^i \parallel^2$ where $\bar{\omega}_k = \frac{1}{N}\sum_{i=1}^{N}\omega_k^i$) by $\mathcal{O}(\tau^2)$—please refer to Lemma 3.3 in [24] or Lemma 3 in [8] for more details. With PAS, we monotonously decrease τ, thus the divergence among clients' local parameters would always be smaller than the original case. Meanwhile, the other items composing $\mathbb{E}[F(\omega_T)] - F(\omega^\star)$ won't be changed. In this way, we can prove that PAS attains an even better linear convergence rate.

V. Implementation

Overall Workflow. We have implemented our PAS solution atop the PyTorch framework, and Fig. 4 describes the overall architecture. In our PAS implementation, we create a `PAS_Daemon` on each client, and a `PAS_Master` on the FL server. A `PAS_Daemon` handles all the synchronization operations for that client, making the technique details of PAS transparent to end users. The `PAS_Master` conducts global parameter aggregation and also tunes the synchronization periods.

Alg. 1 further elaborates the detailed PAS workflow. After a local iteration, each client passes the latest model parameters to `PAS_Daemon` via the `TryEagerSync()` function. That function identifies the parameters that shall be immediately synchronized based on $\vec{\tau}$, and communicates them to the `PAS_Master` via `PAS_Master.SyncAndTune()`. `PAS_Master.SyncAndTune()` also updates the synchronization periods for the synchronized parameters based on the principles in Sec. IV-A. After all the parameters have been synchronized, the `PAS_Daemon` calls `ConfigNextRound()` to refresh the local model parameters and the number of computing iterations for the next round.

Key Implementation Techniques. In particular, there are three critical techniques in our implementation:

1) Batched Communication. While each parameter has a distinct synchronization period in PAS, communicating each parameter totally independently would be inefficient: As observed by Shi et al. [23], each synchronization operation incurs a non-negligible startup overhead and it would be more efficient to synchronize parameters in larger batches. Therefore, we group the parameters with the same synchronization periods into one batch—with the `masked_select()` API provided by PyTorch, which can efficiently extract and repack the masked tensor subset into a new tensor; here the mask is set by checking whether the synchronization period of a parameter equals to the iteration number relative to round start.

2) Non-blocking Synchronization. To enable computation-communication overlapping, we launch the `PAS_Daemon` as a dedicated long-running process. For each parameter batch to synchronize, the `PAS_Daemon` creates a background thread with the `threading.Thread()` API. Moreover, the returned parameters from the `PAS_Master` are temporally cached in `PAS_Daemon`; when all synchronization threads complete, those cached parameters would substitute the old ones and be used in the next round.

3) Incremental Control-plane Update. Under PAS, the `PAS_Daemon` and `PAS_Master` each maintains a copy of $\vec{\tau}$—the control-plane metadata for synchronization. Such metadata is dynamically adjusted on the `PAS_Master`, and shall be kept up to date on each `PAS_Daemon`. When making such control-plane update we shall try to minimize the extra communication overhead. To that end, we choose to update τ incrementally—the `PAS_Master` only notifies `PAS_Daemons` the indices of the parameters whose synchronization periods should be tuned; then they respectively scale down the τ values of the involved parameters by a preset factor γ. This way, we can amortize the control-plane updating overhead.

Algorithm 1 Workflow with PAS

Require: $\gamma, \eta, \omega_0, \vec{\tau}_0$ ▷ γ: scale down divisor of τ; η: learning rate.

Client: i=1, 2, ..., N:

Init: $k_{\text{round\_start}} \leftarrow 0$, $\hat{\tau} \leftarrow \max(\vec{\tau}_0)$ ▷ $\hat{\tau}$ is a scalar

procedure ClientIterate(k)
  $\omega^i_{k+1} \leftarrow \omega^i_k - \eta \nabla F^i(\omega^i_k)$ ▷ regular local iteration
  PAS_Daemon.TryEagerSync($\omega^i_{k+1}, k - k_{\text{round\_start}}$)
  if $k - k_{\text{round\_start}} = \hat{\tau}$ **then** ▷ prepare for the next round
    ω^i_k, $\hat{\tau} \leftarrow$ PAS_Daemon.ConfigNextRound()
    $k_{\text{round\_start}} \leftarrow k$

PAS_Daemon: i=1, 2, ..., N:

Init: $\omega \leftarrow \omega_0$, $\vec{\tau} \leftarrow \vec{\tau}_0$ ▷ ω_0: initial model parameter

function TryEagerSync(ω^i, k)
  $\omega^i[\{x\}] \leftarrow \omega^i$.`masked_select`$(\vec{\tau}[x] = k)$
  if $\omega^i[\{x\}] \neq \emptyset$ **then** ▷ launch sync-thread in the background
    $\omega[\{x\}], X_\Delta \leftarrow$ PAS_Master.SyncAndTune($\omega^i[\{x\}], k$)
    ω.`masked_select`$(\vec{\tau}[x] = k) \leftarrow \omega[\{x\}]$
    $\vec{\tau}$.`masked_select`$(x \in X_\Delta)$ /= γ

function ConfigNextRound()
  wait if any synchronization thread is still active
  return ω, $\max(\vec{\tau})$ ▷ refresh model and control update

PAS_Master:

Init: $\vec{\tau} \leftarrow \vec{\tau}_0$ ▷ $\vec{\tau}_0$: initial synchronization period

procedure SyncAndTune($\omega^1[\{x\}]$, $\omega^2[\{x\}]$, ..., $\omega^N[\{x\}]$, k)
  $\omega[\{x\}] \leftarrow \frac{1}{N}\sum_i \omega^i[\{x\}]$ ▷ aggregate reported parameter batch
  calculate $\mathcal{R}[x]$ where $\vec{\tau}[x] = k$
  $X_\Delta \leftarrow$ the set of indices where $\mathcal{R}[x]$ no longer decreases
  $\vec{\tau}$.`masked_select`$(x \in X_\Delta)$ /= γ
  return $\omega[\{x\}]$, X_Δ ▷ incremental model and control update

Overhead Analysis. Our PAS solution incurs additional overheads in computation, memory and communication. Computation overhead is incurred when calculating gradient consistent rate and selecting the parameters for eager synchronization; memory overhead is incurred because PAS needs to record the synchronization period for each parameter; communication overhead is incurred when remotely updating the synchronization periods. Yet, those overheads are indeed marginal. First, PAS computations (e.g, `masked_select()`) are very efficient with the native tensor-processing APIs of PyTorch. Meanwhile, PAS memory consumption is comparable to the model size, which is orders-of-magnitudes smaller than the memory occupied by input data and feature maps [25]. Finally, with incremental control-plane update, the communication overheads can be amortized to many rounds.

VI. Evaluation

A. Experimental Setup

Hardware Setup. To evaluate PAS performance, we build an EC2 cluster with 100 `c6i.large` instances and 1 `c5a.8xlarge` instance. Each `c6i.large` instance has 2 vCPUs, 4GB RAM and 1Mbps link bandwidth (bounded with the wondershaper [26] tool). Such configurations are similar

to a smart phone. Meanwhile, the `c5a.8xlarge` instance—which has 32 vCPUs, 64GB RAM and a link bandwidth of 10 Gbps—works as the FL server.

Local Training Setup. In our evaluation we use three datasets: CIFAR-10 [27], EMNIST [28] and KWS [29] (KeyWord Spotting dataset, a subset of the Speech Command dataset with 10 keywords). Upon the EMNIST dataset, we train a CNN model containing 2 convolutional layers with 5×5 kernels; upon the CIFAR-10 dataset, we train ResNet-20 [30], a tiny ResNet model optimized for CIFAR-10 dataset; upon the KWS dataset, we train a LSTM network with 2 recurrent layers (with a hidden size of 64). Moreover, to emulate non-IID data setup, we generate each client's local dataset following the Dirichlet distribution [31], which controls the label class composition with a concentration parameter α ($\alpha \to \infty$ indicates IID data among clients, while $\alpha = 0$ indicates that the dataset on each client only has one label class). In our experiments we set α to 0.5, under which 60% clients host no more than 3 classes. Meanwhile, we set the batch size, learning rate and weight decay respectively to 100, 0.01 and 0.01.

Synchronization Setup. In our experiments, the default synchronization period of each parameter is set to 200 (i.e., each value of $\vec{\tau}_0$ is 200). Meanwhile, confronting dynamic client participation, a FL server collects only a fraction of updates reported the earliest; in our experiment we set that fraction to 40%. Regarding our PAS scheme, we set the EMA smoothing factor θ to 0.9, and set γ to 2; besides, considering that too frequent synchronization would incur high communication overhead, we set 12 as the lower-bound value in $\vec{\tau}$.

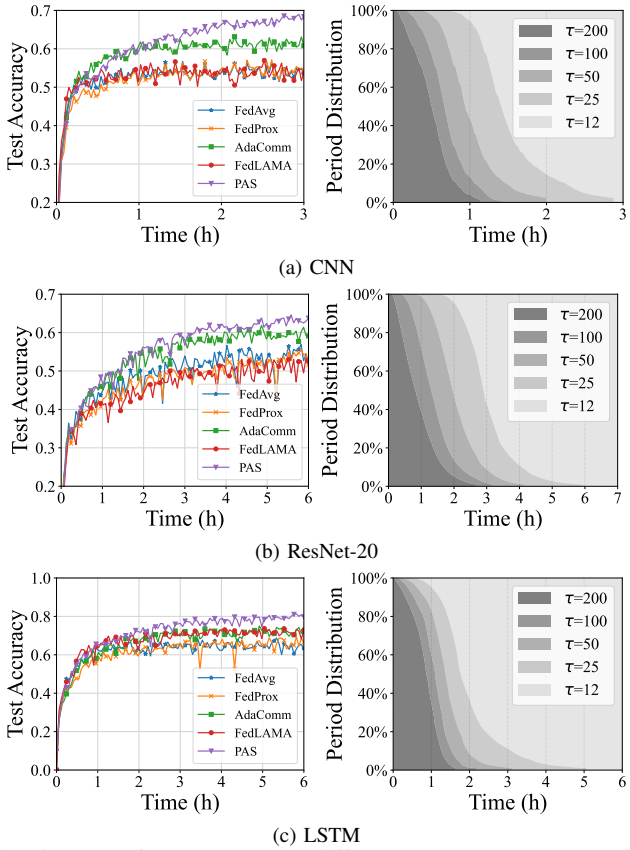
Baselines. We compare PAS against 4 baselines: FedAvg [1], FedProx [32], AdaComm [14] and FedLAMA [17]. FedProx [32] is a famous FL algorithm that introduces an additional regularization term to bound the divergence between the local model and the global one; the weight of the regularization item (μ) is set to the recommended value 0.01. AdaComm [14] is an adaptive method which tunes the synchronization period of the entire model based on the instantaneous loss values (introduced in Sec. II-B). FedLAMA [17] assigns different synchronization periods to different layers; in our experiments, the important layers as judged in [17] are synchronized with a synchronization period of 200, yet the others with 400.

B. End-to-end Performance Comparison

TABLE II: The highest accuracy attained after training CNN, LSTM, and ResNet-20 respectively for 3/6/6 hours.

Method	CNN	ResNet-20	LSTM
FedAvg	57.59	57.66	69.70
FedProx	57.71	56.51	70.32
AdaComm	63.84	61.88	75.45
FedLAMA	57.39	53.38	74.58
PAS	**69.74**	**64.60**	**81.95**

In Table II, we present the end-to-end performance comparison between PAS and the baseline schemes. It shows the highest test accuracy achieved after training CNN, ResNet-20 and LSTM model respectively for 3, 3 and 6 hours. As implied by Table II, our approach PAS can always attains the best accuracy performance. For instance, when training LSTM, the accuracy value attained by PAS is 12.25% higher than vanilla FedAvg, and 6.50% higher than the second-best (AdaComm).



(a) CNN

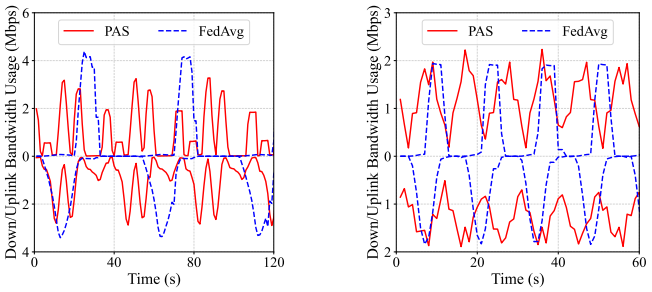
(b) ResNet-20

(c) LSTM

Fig. 5: FL performance under different schemes together with the synchronization period distribution under PAS.

The performance benefit of PAS comes from two aspects. First, by independently tuning the synchronization frequency for each parameter, we can attain the best adaptivity, being able to reach the highest accuracy after a fixed number of training rounds. For example, after training the CNN model for 1000 rounds, the accuracy performance of FedAvg, FedProx, AdaComm, FedLAMA and PAS are 57.59%, 57.76%, 63.35%, 57.39% and 69.04%, respectively. Second, by enabling intra-iteration communication-computation overlapping, PAS can mitigate the network bottleneck and reduce the per-round time. Our measurements show that, when training CNN for the first 1000 rounds, the average per-round time with PAS is 17.9% shorter than that with FedAvg. We will elaborate the communication speedup effect of PAS later in Sec. VI-C.

Further in Fig. 5, we depict the instantaneous test accuracy and the distribution of parameter synchronization periods under PAS. As shown in the left columns of Fig. 5, the accuracy enhancement under PAS is the most rapid among all the schemes in each case. Meanwhile, recall that as training proceeds, model parameters would gradually reach the



(a) ResNet-20 (b) LSTM

Fig. 6: Instantaneous uplink/downlink bandwidth usage of the FL server during a randomly-selected observation window (The upper half of each figure describes the uplink).

gradient-counteracting moments, and PAS would accordingly reduce their synchronization periods. As shown in the right columns of Fig. 5, the proportion of parameters with large synchronization periods always keep shrinking in PAS, and finally almost all the parameters are with the smallest value τ (12). This way, models trained with PAS can be refined with more accurate gradients, and thus attain a higher accuracy.

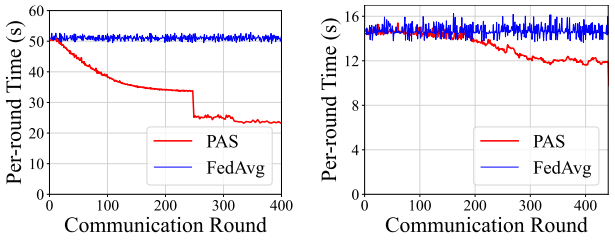
C. Communication Speedup Deep Dive

To better understand the positive effect of PAS on communication efficiency, we measure the instantaneous bandwidth usage during the FL process. In Fig. 6, we show the uplink and downlink bandwidth usage of the FL server during a randomly-selected time window when training ResNet-20 and LSTM. For example, during the observation window of ResNet-20, approximately 20% parameters are with a synchronization period of 200, 30% with 100, 40% with 50, and the remaining 10% with 25. As shown in Fig. 6, by eagerly synchronizing parameters in multiple batches under PAS, both the uplink and downlink can be utilized more thoroughly (active for most of the time). This is contrary to vanilla FedAvg where the links are mostly idle. Therefore, we can observe for both models that the per-round time does become shorter under PAS.

To illustrate the above efficiency speedup more clearly and in a wider time scope, we resort to Fig. 7, which depicts how per-round time varies in the first 400 rounds. As training proceeds, the synchronization periods of different parameters become more heterogeneous, yielding a higher level of communication-computation overlapping and thus a decreasing per-round time. In particular, at round-400, PAS can speed up each training round by 54.5% and 20.3% for ResNet-20 and LSTM, respectively. Note that, after training ResNet-20 for 250 rounds, there is a sudden reduction in per-round time; this is because the maximal synchronization period (i.e., $\hat{\tau}$ in Alg. 1) decreases from 200 to 100 after 250 rounds, further halving the computation time.

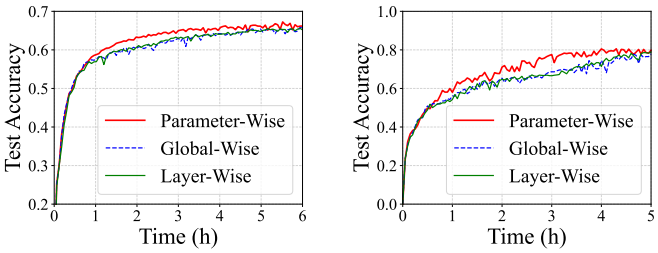
D. Ablation Study on Per-parameter Granularity

PAS has proven effective with (1) adaptive synchronization period tuning (2) at per-parameter granularity. What would happen if we tune the synchronization period still adaptively but not at per-parameter granularity? In this part, we conduct



(a) ResNet-20 (b) LSTM

Fig. 7: The variation of per-round time during the FL process.



(a) ResNet-20 (b) LSTM

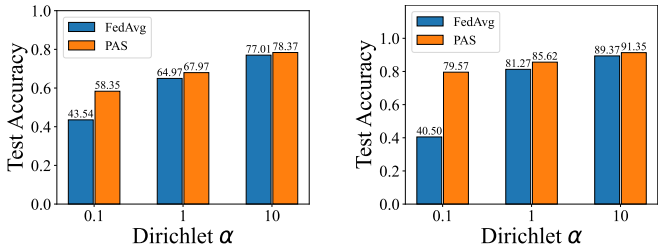
Fig. 8: PAS performance with different tuning granularities.

ablation experiments by comparing our PAS scheme with two variants: *model-wise* period tuning and *layer-wise* period tuning. The former ensures that all the model parameters share an identical (yet dynamic) synchronization period, and the later adjusts synchronization period at *per-layer* granularity. For those variants, the synchronization period of a model (layer) is tuned based on the *average* gradient consistent rate of all the parameters in that model (layer). In Fig. 8, we measure the PAS performance when training ResNet-20 and LSTM in a 40-node cluster with different tuning granularities. As shown in Fig. 8, by tuning $\vec{\tau}$ at the scalar granularity, our PAS scheme attains the best adaptivity and the highest communication-computation overlapping level, thus achieving the most rapid accuracy enhancement.

E. Sensitivity Analysis

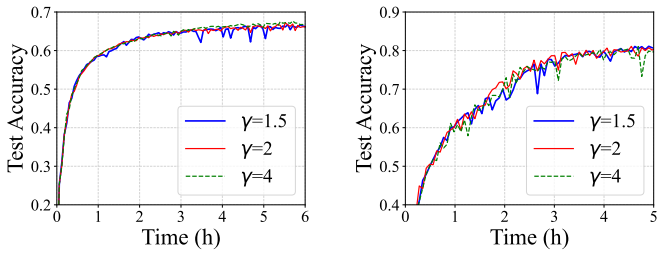
Sensitivity to the Non-IID Level. How does the data heterogeneity level affect PAS performance? To explore that, we follow the experimental setup in Sec. VI-B, and set the hyper-parameter α of the Dirichlet distribution respectively to 0.1, 1 and 10 (the smaller α, the higher the data non-IID level). In Fig. 9, we present the highest test accuracy when training ResNet-20 and LSTM under FedAvg and PAS. We note that the accuracy improvement of PAS is more pronounced with smaller α values (i.e., when facing more heterogeneous data), aligning with our original insights and design rationales.

Sensitivity to hyper-parameter. In our default setup, each time a parameter reaches its *gradient-counteracting* moment, we scale down its synchronization period to a half (i.e., $\gamma = 2$ in Alg. 1). Is that γ hyper-parameter critical to PAS performance? We respectively change γ to 1.5 and 4, and compare the resultant PAS performance with the initial case where $\gamma = 2$. In Fig. 10, we show the FL performance when training ResNet-20 and LSTM with different γ values. As



(a) ResNet-20 (b) LSTM

Fig. 9: The highest accuracy achieved under FedAvg and PAS for ResNet-20 and LSTM with different data non-IID levels.



(a) ResNet-20 (b) LSTM

Fig. 10: PAS performance when setting the scale-down factor γ to different values.

shown in Fig. 10, the performance of PAS is quite stable across different γ: This is because sync-period tuning is triggered periodically as described in Sec. IV; when adopting a larger scale-down factor, sync-period tuning would be triggered for fewer times. Consequently, a parameter would reach a given synchronization period value after a similar amount of time. In sum, PAS is robust to the γ hyper-parameter.

F. Overheads

TABLE III: Additional Overheads Incurred by PAS

Cost	Scheme	CNN	ResNet-20	LSTM
Computation Time (per-round)	FedAvg	11.48s	28.47s	11.54s
	PAS	11.55s (0.58%↑)	28.87s (1.41%↑)	11.62s (0.71%↑)
Memory Occupation Amount	FedAvg	255.3MB	555.7MB	352.9MB
	PAS	260.5MB (2.04%↑)	577.3MB (3.89%↑)	355.3MB (0.68%↑)
Transmission Amount (400 rounds)	FedAvg	136.0MB	433.4MB	85.4MB
	PAS	137.3MB (0.93%↑)	437.5MB (0.95%↑)	85.98MB (0.68%↑)

PAS does incur additional overheads in computation, memory and communication (as elaborated in Sec. V). In Table III, we measure the average PAS overheads for the three models. We find that PAS computations take only around 1% of the original per-round time. Meanwhile, the additional memory cost of PAS is also marginal ($\leq 4\%$). Moreover, we observe from the first 400 rounds that the control-plane updating overhead takes up less than 1% of the total transmission amount. To summarize, the overheads of PAS are negligible compared with its performance benefits.

VII. Additional Related Work and Discussion

In this section, we briefly summarize the related works that respectively improve FL in accuracy and efficiency.

Improving Training Accuracy. To reduce the accuracy loss caused by non-IID data, a primary methodology is to mitigate the heterogeneity among clients' local datasets: Zhao et al. proposed to augment each client's local dataset with shared samples or synthetic samples generated from GAN models [6], [33], and Lai et al. proposed to only select a portion of participating clients with high-quality datasets [20], [34]. Another methodology to improve FL accuracy is to rectify the optimizing strategy: Some proposed to add an extra regularizing term to the loss function [32], [33], and Li et al. proposed to add a specific momentum to the optimizer [35]. All those methods are orthogonal to PAS, and they fail to simultaneously optimize the communication efficiency.

Improving Communication Efficiency. Many research works have been proposed on improving communication efficiency for distributed learning, and we classify them into two categories: *content compression* (i.e., quantization and sparsification) and *scheduling optimization*. Under the content-compression scope, quantization [36], [37], [38] means to transfer parameters in lower bits (e.g., 8-bits or even 1-bit), while sparsification [19], [39], [10] only synchronizes the significant parameters. Those methods can be integrated with PAS. Under the scheduling-optimization scope, communication efficiency is improved by relaxing the synchronization barriers [40], [41], by communicating the early-consumed parameters at higher priorities [42], [43], and also by overlapping communication with computation across different *layers* [22], [23]. To our knowledge, we are the first that propose to overlap communication and computation across different *iterations*.

Discussion on PAS Compatibility with Differential Privacy. So far we have skipped the encryption aspect of FL. A commonly-adopted gradient encryption method for FL is differential privacy (DP), which works by adding random noises (following a Gaussian or Laplace distribution) to the original gradients [44], [45]. Since the noise distribution has a mean value of 0, the added noises do not change the overall gradient bifurcation pattern among clients. Therefore, our PAS method can still work with such gradient variants.

VIII. Conclusion

In this work, we propose Parameter-Adaptive Synchronization (PAS) to simultaneously improve FL performance in both accuracy and efficiency. PAS works by adaptively tuning the synchronization period of each parameter based on its instantaneous gradient bifurcation status. In this way, FL can persistently improve model quality with more accurate refinement in each round, and the communication efficiency can also be enhanced by enabling communication-computation overlap. Prototype evaluations in a realistic FL setup demonstrate that our PAS scheme can improve model accuracy by over 10% while reducing per-round time by over 50%.

ACKNOWLEDGEMENT

This work is supported by the National Natural Science Fundation of China (NSFC) (62202300) and Shanghai Pujiang Program (22PJ1404600). The work of Yifei Zhu is supported by the National Key R&D Program of China (Grant No. 2023YFB2704400). Chen Chen, Yifei Zhu and Minyi Guo are the corresponding authors.

REFERENCES

[1] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *arXiv preprint arXiv:1610.05492*, 2016.

[2] H. B. McMahan, E. Moore, D. Ramage, S. Hampson *et al.*, "Communication-efficient learning of deep networks from decentralized data," *arXiv preprint arXiv:1602.05629*, 2016.

[3] T. Yang, G. Andrew, H. Eichner, H. Sun, W. Li, N. Kong, D. Ramage, and F. Beaufays, "Applied federated learning: Improving google keyboard query suggestions," *arXiv preprint arXiv:1812.02903*, 2018.

[4] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konečnỳ, S. Mazzocchi, B. McMahan *et al.*, "Towards federated learning at scale: System design," *Proceedings of machine learning and systems*, vol. 1, pp. 374–388, 2019.

[5] D. Huba, J. Nguyen, K. Malik, R. Zhu, M. Rabbat, A. Yousefpour, C.-J. Wu, H. Zhan, P. Ustinov, H. Srinivas *et al.*, "Papaya: Practical, private, and scalable federated learning," in *MLSys*, 2022.

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

[7] K. Hsieh, A. Phanishayee, O. Mutlu, and P. B. Gibbons, "The Non-IID Data Quagmire of Decentralized Machine Learning," in *ICML*, 2020.

[8] X. Li, K. Huang, W. Yang, S. Wang, and Z. Zhang, "On the convergence of fedavg on non-iid data," *arXiv preprint arXiv:1907.02189*, 2019.

[9] M. K. Nori, S. Yun, and I.-M. Kim, "Fast federated learning by balancing communication trade-offs," *IEEE Transactions on Communications*, vol. 69, no. 8, pp. 5168–5182, 2021.

[10] C. Chen, H. Xu, W. Wang, B. Li, B. Li, L. Chen, and G. Zhang, "Communication-efficient federated learning with adaptive parameter freezing," in *IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2021.

[11] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, "When edge meets learning: Adaptive control for resource-constrained distributed machine learning," in *IEEE INFOCOM*, 2018.

[12] S. Wang, T. Tuor, K. K. Leung, C. Makaya, T. He, and K. Chan, "Adaptive federated learning in resource constrained edge computing systems," *Journal on Selected Areas in Communications*, vol. 37, no. 6, pp. 1205–1221, 2019.

[13] B. Luo, X. Li, S. Wang, J. Huang, and L. Tassiulas, "Cost-effective federated learning design," in *IEEE INFOCOM*, 2021.

[14] J. Wang and G. Joshi, "Adaptive communication strategies to achieve the best error-runtime trade-off in local-update sgd," in *SysML*, 2019.

[15] J. Zhang, X. Cheng, C. Wang, Y. Wang, Z. Shi, J. Jin, A. Song, W. Zhao, L. Wen, and T. Zhang, "Fedada: Fast-convergent adaptive federated learning in heterogeneous mobile edge computing environment," *World Wide Web*, vol. 25, no. 5, pp. 1971–1998, 2022.

[16] X. Li, K. Huang, W. Yang, S. Wang, and Z. Zhang, "On the Convergence of FedAvg on Non-IID Data," in *ICLR*, 2020.

[17] S. Lee, T. Zhang, and A. S. Avestimehr, "Layer-wise adaptive model aggregation for scalable federated learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, no. 7, 2023, pp. 8491–8499.

[18] M. D. Zeiler, "ADADELTA: an adaptive learning rate method," *arXiv preprint arXiv:1212.5701*, 2012.

[19] K. Hsieh, A. Harlap, N. Vijaykumar, D. Konomis, G. R. Ganger, P. B. Gibbons, and O. Mutlu, "Gaia: Geo-distributed machine learning approaching LAN speeds," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2017, pp. 629–647.

[20] F. Lai, X. Zhu, H. V. Madhyastha, and M. Chowdhury, "Oort: Efficient federated learning via guided participant selection." in *OSDI*, 2021, pp. 19–35.

[21] S. Hochreiter and J. Schmidhuber, "Flat minima," *Neural Computation*, vol. 9, no. 1, pp. 1–42, 1997.

[22] H. Zhang, Z. Zheng, S. Xu, W. Dai, Q. Ho, X. Liang, Z. Hu, J. Wei, P. Xie, and E. P. Xing, "Poseidon: An efficient communication architecture for distributed deep learning on gpu clusters." in *USENIX Annual Technical Conference*, vol. 1, no. 1, 2017, pp. 1–2.

[23] S. Shi, X. Chu, and B. Li, "Mg-wfbp: Efficient data communication for distributed synchronous sgd algorithms," in *IEEE Conference on Computer Communications (INFOCOM)*, 2019.

[24] S. U. Stich, "Local sgd converges fast and communicates little," *arXiv preprint arXiv:1805.09767*, 2018.

[25] M. Rhu, N. Gimelshein, J. Clemons, A. Zulfiqar, and S. W. Keckler, "vdnn: Virtualized deep neural networks for scalable, memory-efficient neural network design," in *49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2016.

[26] "wondershaper," https://github.com/magnific0/wondershaper, 2020.

[27] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[28] G. Cohen, S. Afshar, J. Tapson, and A. Van Schaik, "Emnist: Extending mnist to handwritten letters," in *2017 international joint conference on neural networks (IJCNN)*. IEEE, 2017, pp. 2921–2926.

[29] P. Warden, "Speech commands: A dataset for limited-vocabulary speech recognition," *arXiv preprint arXiv:1804.03209*, 2018.

[30] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

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

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

[33] E. Jeong, S. Oh, H. Kim, J. Park, M. Bennis, and S.-L. Kim, "Communication-efficient on-device machine learning: Federated distillation and augmentation under non-iid private data," *arXiv preprint arXiv:1811.11479*, 2018.

[34] Y. J. Cho, J. Wang, and G. Joshi, "Towards understanding biased client selection in federated learning," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2022.

[35] C. Li, R. Li, P. Zhou, H. Wang, Y. Li, S. Guo, and K. Li, "Gradient scheduling with global momentum for non-iid data distributed asynchronous training," *arXiv preprint arXiv:1902.07848*, 2019.

[36] N. Shlezinger, M. Chen, Y. C. Eldar, H. V. Poor, and S. Cui, "Federated learning with quantization constraints," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020.

[37] F. Seide, H. Fu, J. Droppo, G. Li, and D. Yu, "1-bit stochastic gradient descent and its application to data-parallel distributed training of speech dnns," in *INTERSPEECH*, 2014.

[38] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic, "QSGD: Communication-efficient SGD via gradient quantization and encoding," in *NeurIPS*, 2017.

[39] W. Luping, W. Wei, and L. Bo, "Cmfl: Mitigating communication overhead for federated learning," in *IEEE international conference on distributed computing systems (ICDCS)*, 2019.

[40] C. Chen, W. Wang, and B. Li, "Round-robin synchronization: Mitigating communication bottlenecks in parameter servers," in *IEEE Conference on Computer Communications (INFOCOM)*, 2019.

[41] Q. Ho, J. Cipar, H. Cui, S. Lee, J. K. Kim, P. B. Gibbons, G. A. Gibson, G. Ganger, and E. P. Xing, "More effective distributed ml via a stale synchronous parallel parameter server," in *NeurIPS*, 2013.

[42] Y. Peng, Y. Zhu, Y. Chen, Y. Bao, B. Yi, C. Lan, C. Wu, and C. Guo, "A generic communication scheduler for distributed dnn training acceleration," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, 2019.

[43] A. Jayarajan, J. Wei, G. Gibson, A. Fedorova, and G. Pekhimenko, "Priority-based parameter propagation for distributed dnn training," *Proceedings of Machine Learning and Systems*, vol. 1, pp. 132–145, 2019.

[44] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *ACM CCS*, 2015.

[45] N. Wu, F. Farokhi, D. Smith, and M. A. Kaafar, "The value of collaboration in convex machine learning with differential privacy," in *IEEE Symposium on Security and Privacy*, 2020.