

Federated Learning With Heterogeneity-Aware Probabilistic Synchronous Parallel on Edge

Zhao, Jianxin ; Han, Rui; Yang, Yongkai ; Catterall, Benjamin ; Liu, Chi Harold; Chen, Lydia Y.; Mortier, Richard; Crowcroft, Jon; Wang, Liang

DOI

[10.1109/TSC.2021.3109910](https://doi.org/10.1109/TSC.2021.3109910)

Publication date

2022

Document Version

Final published version

Published in

IEEE Transactions on Services Computing

Citation (APA)

Zhao, J., Han, R., Yang, Y., Catterall, B., Liu, C. H., Chen, L. Y., Mortier, R., Crowcroft, J., & Wang, L. (2022). Federated Learning With Heterogeneity-Aware Probabilistic Synchronous Parallel on Edge. *IEEE Transactions on Services Computing*, 15(2), 614-626. Article 9529051. <https://doi.org/10.1109/TSC.2021.3109910>

Important note

To cite this publication, please use the final published version (if applicable). Please check the document version above.

Copyright

Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

Takedown policy

Please contact us and provide details if you believe this document breaches copyrights. We will remove access to the work immediately and investigate your claim.




Green Open Access added to TU Delft Institutional Repository

'You share, we take care!' - Taverne project

<https://www.openaccess.nl/en/you-share-we-take-care>

Otherwise as indicated in the copyright section: the publisher is the copyright holder of this work and the author uses the Dutch legislation to make this work public.

Federated Learning With Heterogeneity-Aware Probabilistic Synchronous Parallel on Edge

Jianxin Zhao , Rui Han , Yongkai Yang, Benjamin Catterall, Chi Harold Liu , *Senior Member, IEEE*, Lydia Y. Chen , *Senior Member, IEEE*, Richard Mortier, Jon Crowcroft , *Fellow, IEEE*, and Liang Wang 

Abstract—With the massive amount of data generated from mobile devices and the increase of computing power of edge devices, the paradigm of Federated Learning has attracted great momentum. In federated learning, distributed and heterogeneous nodes collaborate to learn model parameters. However, while providing benefits such as privacy by design and reduced latency, the heterogeneous network present challenges to the synchronisation methods, or barrier control methods, used in training, regarding system progress and model convergence etc. The design of these barrier mechanisms is critical for the performance and scalability of federated learning systems. We propose a new barrier control technique called Probabilistic Synchronous Parallel (PSP). In contrast to existing mechanisms, it introduces a sampling primitive that composes with existing barrier control mechanisms to produce a family of mechanisms with improved convergence speed and scalability. Our proposal is supported with a convergence analysis of PSP-based SGD algorithm. In practice, we also propose heuristic techniques that further improve the efficiency of a PSP. We evaluate the performance of proposed methods using the federated learning specific FEMNSIT dataset. The evaluation results show that PSP can effectively achieve good balance between system efficiency and model accuracy, mitigating the challenge of heterogeneity in federated learning.

Index Terms—Federated learning, edge computing, distributed computing, barrier control

1 INTRODUCTION

As a large amount of data is increasingly generated from mobile and edge devices (smart home, mobile phone, wearable devices, etc.), it becomes essential for many applications to train machine learning distributedly across many nodes. One of the emerging distributed training paradigms is the *Federated Learning* [1], [2]. Federated learning allows machine learning tasks to take place without requiring data to be centralised. There are a variety of motivations behind, e.g., maintaining privacy by avoiding individuals reveal their personal data to others, or latency by allowing data processing to take place closer to where and when the data is generated. Due to these reasons, it has been gaining increasing popularity in various research and application fields, such as vehicle network [3], edge cache optimising and computation offloading [4], health care [5], mobile keyboard prediction [6], etc.

One critical component of distributed and federated machine learning systems is barrier synchronisation: the

mechanism by which participating nodes coordinate in the iterative distributed computation. Currently, the barrier control methods can be generally categorized into three types. The *Bulk Synchronous Parallel (BSP)* is the strictest, which requires all workers to proceed in lock-step moving to the next iteration only when all the workers are ready. The *Asynchronous Parallel (ASP)* [7] is the least strict barrier control, since it allows each worker to proceed at its own pace without waiting for the others. The third one is the *Stale Synchronous Parallel (SSP)* [8], which relaxes BSP by allowing workers to proceed to the next iteration once all workers' iterations are within a certain limit with each other. These barrier methods provide different trade-offs between system performance and model accuracy.

However, Federated Learning brings challenges to the existing barrier control methods. As deployments move out from the data centre to the heterogeneous edge network, we must cope with unreliable networks that have higher latency, and churn in node availability arising from potentially much larger numbers of participating nodes. ASP can achieve fast progress, but cannot guarantee convergence of the model training with such heterogeneous networks. BSP, on the other hand, provides a tight synchronisation requirement for the trained model, but its system progress is prone to the delay caused by stragglers. The SSP provides a certain degree of trade-off between these two ends, but this centralised control method still falls short of supporting large scale heterogeneous networks.

Towards this end, we propose the *Probabilistic Synchronous Parallel (PSP)*. The basic idea is to introduce a *sampling* primitive in the system, and to use a sampled subset of participating workers to estimate progress of the entire system. PSP introduces a second dimension to this trade-off: from how many nodes must we receive updates before proceeding to

- Jianxin Zhao, Rui Han, and Chi Harold Liu are with the School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100811, China. E-mail: jianxin.zhao@cl.cam.ac.uk, {hanrui, chliu}@bit.edu.cn.
- Yongkai Yang is with the TravelSky Technology Limited, Beijing Engineering Research Center of Civil Aviation Big Data, Beijing 101318, China, and also with the Key Laboratory of Intelligent Passenger Service of Civil Aviation-CAAC, Beijing 101318, China. E-mail: ykyang@travelsky.com.
- Benjamin Catterall, Richard Mortier, Jon Crowcroft, and Liang Wang are with the Department of Computer Science and Technology, University of Cambridge, Cambridge CB2 3AX, U.K. E-mail: ben_catterall@hotmail.co.uk, {richard.mortier, jon.crowcroft, liang.wang}@cl.cam.ac.uk.
- Lydia Y. Chen is with the Department of Computer Science, Technology University Delft, 2628 Delft, The Netherlands. E-mail: lydiaychen@ieee.org.

Manuscript received 1 Jan. 2021; revised 30 July 2021; accepted 30 Aug. 2021.

Date of publication 3 Sept. 2021; date of current version 8 Apr. 2022.

(Corresponding author: Yongkai Yang.)

Digital Object Identifier no. 10.1109/TSC.2021.3109910

the next iteration. By composing our *sampling* primitive with traditional barrier controls, we obtain a family of barrier controls better suited for supporting iterative learning in the heterogeneous networks.

The contributions of this paper are shown as follows.

- We propose a new barrier control method, PSP, that utilises sampling methods to address the challenges caused by heterogeneous networks in Federated Learning. It also provides a new dimension to the design of barrier control methods compared to existing ones.
- We outline a theoretical analysis of these PSP control methods, providing probabilistic convergence guarantees as a function of sample size.
- We implement the PSP synchronisation method in a Parameter Server framework using PyTorch.
- We conduct extensive evaluation on benchmark and datasets that are designed specifically for Federated Learning. Both practical and theoretical results indicate that even a relatively small sample size brings most of the benefits of PSP, and that they can achieve better trade-offs than existing barrier methods.

The rest of this paper is organised as follows. We first review the background works that are related with distributed learning, barrier control, and federated learning Section 2. Next we present the design of probabilistic synchronous parallel Section 3 and its implementation Section 4. This section is followed by a theoretical analysis of the convergence of this method Section 5. Finally, we present the evaluation results Section 6, followed by discussions and conclusions.

2 BACKGROUND AND RELATED WORK

2.1 Distributed Learning

In distributed learning, a model is trained via the collaboration of multiple workers. One of the most commonly used training methods is the Stochastic Gradient Descent (SGD), which iteratively optimizes the given objective function until it converges by following the gradient direction of the objective. In each iteration of SGD, typically a descent gradient is calculated using a batch of training data, and then the model parameters are updated by changing along the direction of the gradient at a certain step. It is widely applied in fields such as wireless communication systems, and Internet of Things, etc. [16], [17], [18].

2.2 Barrier Control Methods

As noted above, the statistical and iterative nature of machine learning means that errors are incrementally removed from the system. To be perfectly consistent, where every node proceeds to the next iteration together risks reducing throughput. Relaxing consistency can improve system performance without ultimately sacrificing accuracy. This trade-off is embodied in the *barrier control* mechanism. Current barrier control mechanisms can be divided into three types, discussed in details below.

Bulk Synchronous Parallel (BSP) is a deterministic scheme where workers perform a computation phase followed by a synchronisation/communication phase to exchange updates,

under control of a central server [19]. BSP programs are often serialisable, i.e., they are equivalent to sequential computations, if the data and model of a distributed algorithm have been suitably scheduled, making BSP the strongest barrier control method [8]. Numerous variations of BSP exist, e.g., allowing a worker to execute more than one iteration in a cycle [20]. Federated Learning also uses BSP for its distributed computation [2]. Unfortunately, in BSP workers must wait for others to finish and so *stragglers* limit performance to that of the slowest node. Thus, BSP tends to offer consistency and high accuracy in each update but suffers from poor performance except in highly favourable environments. Moreover, BSP requires centralised coordination.

Asynchronous Parallel (ASP) takes the opposite approach to BSP, allowing computations to execute as fast as possible by running all workers completely asynchronously [7]. ASP can result in fast convergence because it permits the highest possible rate of iteration [19]. However, the lack of any coordination means that updates are calculated based on old model state, resulting in reduced accuracy. There are no theoretical guarantees as to whether algorithms converge. The Hogwild scheme proposed in [7] has many limits, e.g., it requires convex function and sparse update. Many work have tried to extend these limits in application and theoretical analysis [21], [22]. These studies often lead to carefully tuned step size in training. [23] proposes a delay-compensated SGD that mitigates delayed updates in ASP by compensating the gradients received at the parameter server. [14] introduces another variant of ASP specifically for wide-area networks: as communication is a dominant factor, it advocates allowing insignificant updates to be delayed indefinitely in WAN.

Stale Synchronous Parallel (SSP) is a bounded asynchronous model that balances between BSP and ASP. Rather than requiring all workers to proceed to the next iteration together, it requires only that the iteration of any two workers in the system differs by at most s , a pre-defined *staleness* bound. The staleness parameter limits error and allows SSP to provide deterministic convergence guarantees [8], [19], [24]. Built on SSP, [25] investigates the *n-softsync*, the synchronisation method that makes the parameter server updating its weight after collecting certain number of updates from any workers. [26] proposes to remove a small amount of “longtail” workers, or add a small amount of backup nodes to mitigate this effect while avoiding asynchronous noise. [27] tries to unified the existing synchronisation methods in a single framework and provide convergence analysis.

Table 1 summarises the synchronisation primitives used by different machine learning systems. BSP is the most commonly used as it is deterministic and has the most straightforward implementation. However, it is vulnerable to network and system dynamics where a node failing or experiencing connectivity problems can cause the entire system to slow down or stall. In contrast, ASP is resilient to such problems as no synchronisation between nodes is required – but the lack of synchronisation means that model updates from different nodes may arise from very different iterations and so result into significant errors.

This trade-off can be demonstrated using a small experiment shown in Fig. 1. Here we use the configuration from Section 6. The left side shows that, after the same number of

TABLE 1
Classification of Synchronisation Methods Used by Different Systems

System	Synchronisation Constraint	Barrier Method
MapReduce [9]	Requires map to complete before reducing	BSP
Hogwild! [7]	ASP but system-level bounds on delays	ASP, SSP
Parameter Servers [10]	Swappable synchronisation method	BSP, ASP, SSP
Hadoop [11], Spark [12]	Aggregate updates after task completion	BSP
Yahoo! LDA [13]	Checkpoints	SSP, ASP
SSPTable [8]	Updates delayed by up to $N - 1$ steps	SSP
Gaia [14]	Accumulate weight locally	BSP, SSP
Astraea [15]	Combination with Sequential update	BSP

updates exchanged between server and clients, the ASP has much larger jitters than BSP during the process and even fails to converge, while BSP keeps following a growing trend. The right side shows that, given the same time, the number of updates accumulated using the ASP is much larger than that of BSP, about 5x at 200s in this experiment.

SSP attempts to exploit this trade-off by bounding the difference in iterations between participating nodes. Fig. 2a depicts how existing barrier control methods balance consistency against iteration rate to achieve a high rate of convergence. As can be seen, moving from ASP to BSP, these methods are tuned along a 1-dimensional space. As will be shown in the next section, a new dimension of trade-off can be exploited.

2.3 Federated Learning

Nowadays, these existing distributed training methods are put to use in an emerging field, the Federated Learning (FL). FL is a machine learning setting where “multiple entities collaborate in solving a machine learning problem, under the coordination of a central server or service provider; each client’s raw data is stored locally and not exchanged or transferred.” [28]. It combines techniques from multiple fields, including distributed training, machine learning, and privacy, etc. It has increasingly drawn research interest recently [1]. One such prominent application field is in edge computing [29]. For example, [30] implements federated training algorithms on mobile edge computing frameworks, where the main challenge is to manage limited resources of heterogeneous clients.

One major cause of performance degradation in distributed training is the heterogeneity. It involves many different aspects: the computation power difference caused by hardware, random faults of worker devices, temporary worker slowdown or dropout due to resource sharing, network connection issue, single point failure, etc. Heterogeneity means

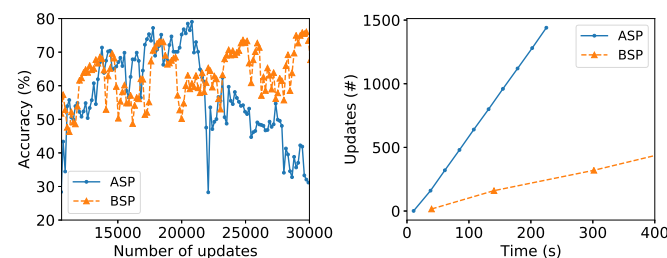


Fig. 1. The difference of ASP and BSP with regard to model accuracy and system progress.

the progresses of different devices vary greatly, and aggregating drastically different weights leads to degraded performance. Many studies have been proposed to address the challenge of heterogeneity in training from multiple perspectives.

One of the basic algorithms in Federated Learning is the FedAvg [31]. It works by averaging models across different workers, shown to be robust against common problems in FL such as the unbalanced data distribution across devices. Some work propose new algorithms. [32] introduces the MOCHA optimization method for the multi-task federated learning, where each worker trains its own model, so as to avoid the heterogeneity problem. Another technique frequently used is to manipulate the weight generated by the local workers. In [33] the authors propose the FedProx algorithm, which adds a regularization step in local training to limit the local weight’s change. Some work tries to explicitly remove the stragglers. To improve the performance of the SSP barrier control method in heterogeneous environments, the [26] proposes to enhance the SSP by removing stragglers and add backup workers in case some of them drop out of the training progress due to heterogeneity. [34] proposes Overlap SGP, a method that uses a gossip-based algorithm to mitigate the heterogeneity. Recently, the authors in [15] propose to mitigate this problem with data augmentation techniques.

Moreover, in Federated Learning, the training data are often non-IID [15]. That is, a device’s local data cannot be regarded as samples drawn from the overall distribution. The data distribution has an enormous impact on model training. The authors in [35] provide theoretical analysis about FedAvg on non-IID data. The work [15] proposes the Astraea system to address the imbalanced data problem. Besides data enhancement, its strategies include a combination of sequential update and BSP in updating, given how biased the data is.

The impact of data distribution can be demonstrated clearly using an experiment, shown in Fig. 3. We use BSP in distributed training for both IID and non-IID scenarios, whose hyper-parameters such as optimiser type, learning rate, epoch, batch size, worker number, etc. are the same as Section 6. As a result, with the IID data, the trained model accuracy increases steadily, and soon reaches a fairly high 80% accuracy. With the non-IID data, however, the model accuracy stays around 40% within the same timespan, and also shows large jitters in training. In the evaluation section of this work, we use a non-IID dataset to better model the Federated Learning setting.

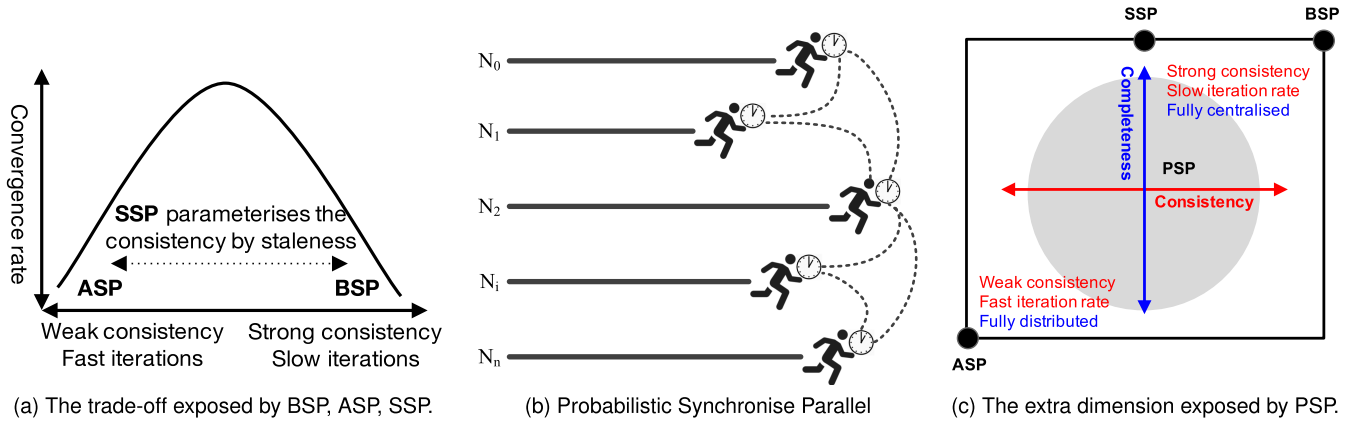


Fig. 2. The extra trade-off space enabled by PSP, and an illustration about how PSP works among workers.

3 PROBABILISTIC SYNCHRONOUS PARALLEL

In this section, we introduce the proposed barrier control method *Probabilistic Synchronous Parallel* (PSP), including its core design principles, how it expands the parameter tuning space in the design of synchronisation methods, and how it is compatible with existing methods.

Fig. 4 provides an overview of PSP barrier control methods. The core idea behind PSP is simple yet powerful: *we require that only some proportion, not all, of the working nodes be synchronised in progress* (step 1). By “progress” we mean the number of updates pushed to the server at the client’s side, and the total number of updates collected at the server’s side. In a centralised training framework, the server builds this subset of the training nodes based on system information, such as their current local progress. This subset can be sampled by various approaches. One common and intuitive way is to sample randomly among all the workers.

The parameter in PSP, the sampling size, therefore controls how precise this estimation is. *Assuming this random subset is not biased with respect to nodes’ performance*, the server can use the resulting distribution to derive an estimate of the percentage of nodes which have passed a given progress (step 2). This estimation depends on the specific method used within the subset, as will be discussed in Section 3.2. Accordingly the server can decide whether to allow the trainers to pass the barrier and advance their local progress (step 3).

As illustrated in Fig. 2b, each node keeps record of its own progress. It chooses to synchronise with only several other nodes to decide its own barrier, instead of all other nodes. To ignore the status of the other workers with impunity, we rely on the fact that, in practice, many iterative learning algorithms can tolerate a certain degree of error as

they converge to their final answers [36]. By controlling the sampling method and size, PSP reduces the impact of lagging nodes while also limiting the error introduced by nodes returning updates based on stale information.

In an unreliable environment, we can minimise the impact of outliers and stragglers by probabilistically choosing a subset of the total workers as estimation. As shown in the subsequent section, this method is proved, both in theory and practice, to be robust against the effect of stragglers and the gradient bias caused by non-IID data, while also ensuring an acceptable consistency between iterations as the algorithm progresses.

3.1 Dimension of Barriers Control Methods

The sampling strategy of PSP has a profound implication on design of barrier control methods. As shown in Fig. 2a, current methods tightly coupled model consistency and barrier control: one server is assigned to update model parameters and coordinate the progress of all nodes in an iterative learning algorithm. SSP provides a degree of flexibility in tuning systems but still requires a global knowledge of the entire system network.

To address this problem, by introducing a system primitive *sampling*, PSP decouples the barrier control from the model consistency. This primitive can be composed with existing mechanisms, specifically BSP and SSP, to construct fully distributed barrier control mechanisms that are more scalable: less communication to avoid throttling the server as the system grows much bigger.

The decoupling of the degree of synchronisation from distribution is depicted in Fig. 2c. PSP introduces *completeness* as a second axis by having each node sampled from the population. This axis goes from fully complete (all working nodes are considered in synchronisation) to not complete (each single node is considered separately). Therefore, a

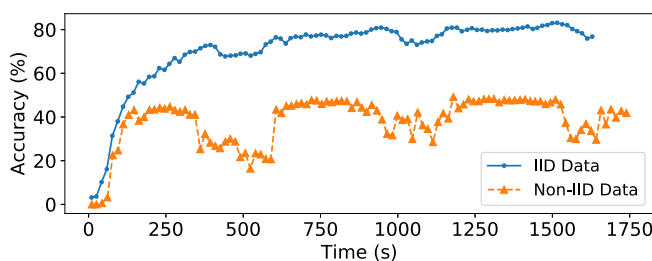


Fig. 3. Comparison of distributed training using IID and non-IID data.

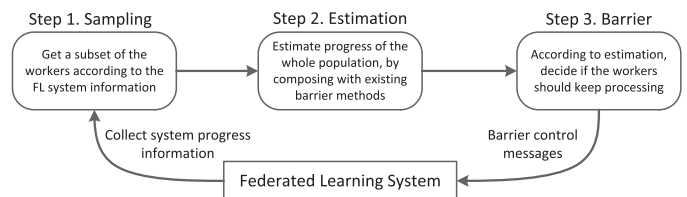


Fig. 4. Overview of PSP.

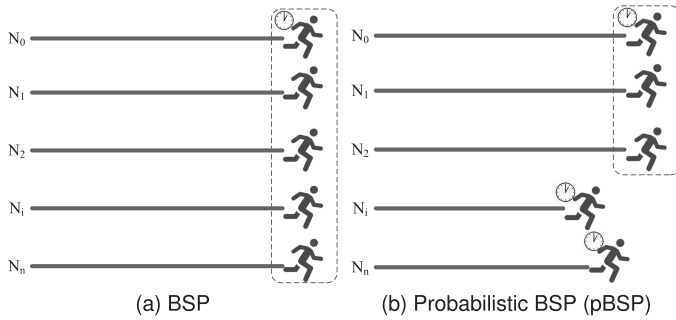


Fig. 5. Compose PSP with bulk synchronise parallel.

2-dimensional tuning space can be explored in synchronisation method design: to be robust against the effect of stragglers while also ensuring a degree of consistency between iterations as the algorithm progresses.

3.2 Compatibility

One noteworthy advantage of PSP lies in that it is straightforwardly compatible with existing synchronisation methods. In classic BSP and SSP, their barrier control mechanisms are invoked by a central server to test the synchronisation condition with the given inputs. For BSP and SSP to use the *sampling* primitive, they simply need to use the sampled states rather than the global states when evaluating the barrier control condition. Within each sampled subset, these traditional mechanisms can then be applied. We can thus easily derive probabilistic versions of BSP and SSP, namely *pBSP* and *pSSP*.

Formally, at the barrier control point, a worker samples β out of P workers without replacement. If one lags more than s updates behind the current worker then the worker waits. This process is *pBSP* (based on BSP) if the staleness parameter $s = 0$ and *pSSP* (based on SSP) if $s > 0$. If $s = \infty$, PSP reduces to ASP.

Fig. 5 depicts PSP, showing to compose BSP with PSP, a subset of the population of nodes is chosen, and then the BSP is applied within the subset (*pBSP*). The composition of PSP and SSP (*pSSP*) follows the same idea. When compared with BSP and SSP, we can obtain faster convergence through faster iterations and with no dependence on a single centralised server. When compared with ASP, we can obtain faster convergence with stronger guarantees by providing greater consistency between updates. In all cases, the original synchronisation control requires a centralised node to hold the global state while the PSP control methods do not and so can be executed independently on each individual node, giving a fully distributed solution.

Besides existing barrier control methods, PSP is also compatible with both centralised and decentralised training approaches. In a decentralised setting, based on the information it gathers from its neighbouring nodes, a trainer node may either decide to pass the barrier control by advancing its local progress, or wait until the threshold is met, in a similar fashion as we have discussed before. Since we are mainly discussing the Federated Learning setting in this paper, we focus on the centralised training approach. In Federated Learning, nodes may join and leave the training system randomly, which would cause the other nodes to wait indefinitely when synchronised barriers such as BSP and SSP are used. PSP solves this issue with probabilistic

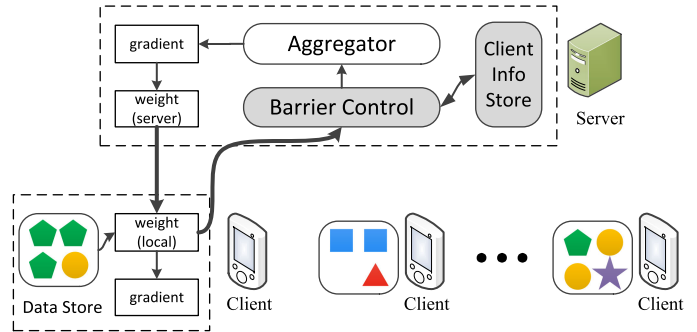


Fig. 6. Implementation architecture of parameter server in federated learning.

sampling, so that for each node, its synchronisation group is assigned dynamically. The system is thus aware of the status change of any node with certain probability during a training round.

3.3 Sampling Strategy

Here we formally introduce the probabilistic sampling strategy. By default, we can simply choose the trainers randomly. Indeed, as we show in the theoretic analysis and experiments in the sections below, this strategy works sufficiently well.

The choice of samples has a great impact on the performance of PSP. As explained in Section 3, the sampling of PSP provides an estimate of the total distribution of the progress of all the workers. In a worst case scenario where the sampled subset happens to be all stragglers, this subset cannot provide a very efficient estimation of all the workers.

Towards this end, we provide two heuristic *sampling strategies* to address this issue. The first is to randomly re-sample the subset in PSP. The second is to choose the workers according to its previous computation time. Specifically, at each round, we categorize all the workers into two groups according to their historical computing time per iteration, one slow and one fast, and then choose equal numbers of workers from both groups to form the target subset. Here we can use clustering algorithms such as K-Means. For simplicity, in later evaluation, we group the workers according to a pre-defined threshold.

4 IMPLEMENTATION

Parameter Server is the most widely used training framework used in Federated Learning systems such as in [2]. Therefore we focus on applying barrier control methods to this framework in this work.

The system architecture is shown in Fig. 6. One main component of the server is the barrier control mechanism. It collects gradient updates from part of the participating clients, and decides if the current collected gradients can be passed to aggregator or block to wait for further updates to arrive. The different types of barrier methods are explained in the previous sections. If the barrier control decides that the current gradients can be passed, they are aggregated and used to update the model weights maintained by the server. During this process, worker information such as iteration progress and timestamps is stored in the server for applying barrier methods. The logic on the client side is

simple: accept the weight from the server, compute its own gradient based on its local data, and push it to the server. Here the different shape of blocks denotes different types of data. They are distributed in a non-IID way across clients. Formally, the progress we have implemented is described in Algorithm 1.

Algorithm 1. Federated Learning With Barrier Control.

Server:

```

initialise progress information  $P$ 
for each iteration  $k$  do
  aggregate pushed gradients in set  $S_k$ 
  if Barrier( $P, S_k$ ) == True then
    aggregate  $g^{(k)} \leftarrow \sum_{i \in S_k} g_i^{(k)}$ 
    for each client  $s$  in  $S_k$  do
       $P_s \leftarrow P_s + 1$ 
    end for
    update weight  $w^{(k+1)} \leftarrow w^k - \eta g^{(k)}$ 
    Push  $w^{(k+1)}$  to clients
  end if
end for

```

Client:

```

for each iteration  $t$  do
  accept weight  $w^{(t)}$  from the server
  load training data  $x_t, y_t$ 
  gradient  $g^{(t)} \leftarrow \partial \mathcal{L}(x_t, y_t, w^{(t)})$ 
  push gradients  $g^{(t)}$  to server
end for

```

We implemented the training system using the open source framework PyTorch v1.6. The communication between the server and clients uses PyTorch’s Distributed RPC framework. Each client uses the RReF, a mechanism to handle remote object reference, to access the states of the server’s RPC instance. We use the Docker containers to manage the server and workers. Each container can be assigned a different number of CPUs or memory.

To test the accuracy of the trained model, the validation is conducted repeatedly after a fixed number of iterations, using an extra process on the server. For the model used in the framework, we choose to use a LeNet-like convolutional neural network as the trained model. It consists of two convolution layers that are followed by a ReLU activation layer, and then two fully connected layers.

To better model the heterogeneous computing power among participating workers in Federated Learning, we randomly choose a certain number of workers, and make them m times slower in computing the back-propagation.

Note that our focus is to evaluate the barrier methods, so in the implementation we don’t consider techniques such as optimised SGD, merging multiple updates or compressing them before sending them to the server, even though these techniques are proved to be useful in improving training efficiency.

5 CONVERGENCE ANALYSIS

In this section, we present a theoretical analysis of PSP and show how it affects the convergence of ML algorithms (SGD used in the analysis). The analysis mainly shows that: 1)

TABLE 2
Notation Table

Notation	Explanation
$f(r)$	Probabilities of a node lagging r steps
$F(r)$	Cumulative distribution function of $f(r)$
$R[X]$	Difference between optimal and noisy state
a	$F(r)^\beta$, where β is sample size
γ_t	Sequence inconsistency at length t
T	Total length of update sequence
L	Lipschitz constant
σ	Initial learning rate, constant
P	Total number workers, constant

under PSP, the algorithm only has a small probability not to converge, and the upper limit of this probability decreases with the training iterations; 2) instead of choosing large sampling size, it is proved that a small number is already sufficient to provide a good performance.

We apply an analysis framework similar to that of [24]. Notations used in our analysis are listed in Table 2. At each barrier control point, every worker A samples β out of P workers without replacement. If one of these sampled workers lags more than s steps behind worker A, it waits. The probabilities of a node lagging r steps are drawn from a distribution with probability mass function $f(r)$ and cumulative distribution function (CDF) $F(r)$. Without loss of generality, we set both staleness r and sample size β parameters to be constants.

In a distributing machine learning process, these P workers keep generating updates, and a shared model is updated with them continuously. We count these updates by first looping over all workers at one iteration, and then across all the iterations. In this process, each one is incrementally indexed by integer t . The total length of this update sequence is T . Ideally, in a fully deterministic barrier control system such as BSP, the ordering of updates in this sequence should be fixed. We call it a *true sequence*. However, in reality, what we get is often a *noisy sequence*, where updates are reordered irregularly due to sporadic and random network and system delays. These two sequences share the same length. We define *sequence inconsistency* as the number of index difference between these two sequences, and denote it by γ_t . It shows how much a series of updates deviate from an ideal case. If the sequence inconsistency is bounded, it means that what a true sequence achieves, in time, a noisy sequence can also achieve, regardless of the order of updates. This metric is thus a key instrument in theoretically proving the convergence property of an asynchronous barrier method.

Let $R[X] = \sum_t^T f_t(\tilde{x}_t) - f_t(x^*)$. This is the sum of the differences between the optimal value of the function and the current value given a noisy state. To put in plain words, it shows the difference between “the computation result we get if all the parameter updates we receives are in perfect ideal order” and “the computation result we get in real world when using e.g., PSP barrier”. Now we show a probabilistic bound on $R[X]$, which shows the noisy system state, \tilde{x}_t , converges in expectation towards the optimal, x^* , in probability.

Theorem 1 (SGD under PSP, convergence in probability). Let $f(x) = \sum_{i=1}^T f_i(x)$ be a convex function where each

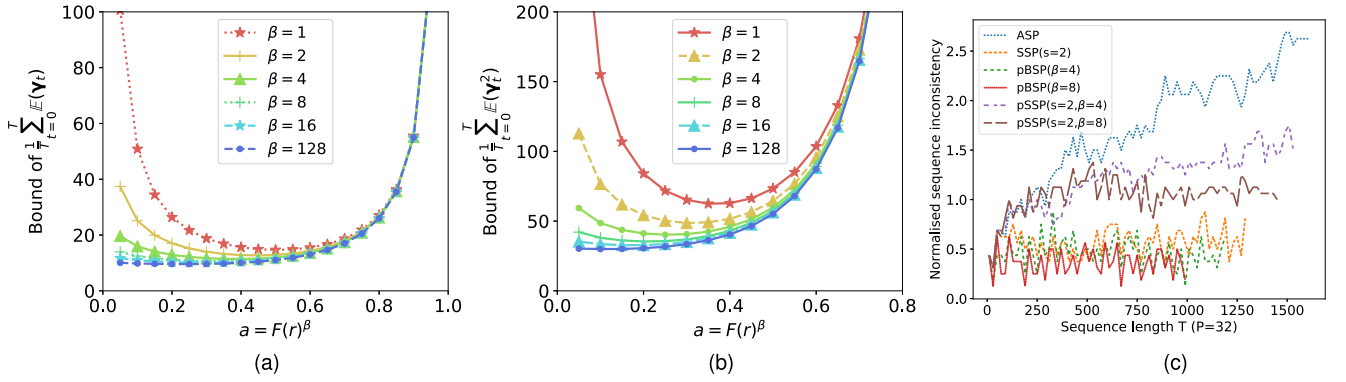


Fig. 7. (a-b) Bounds on mean and variance of sampling distribution as a function of $F(r)^\beta$. The staleness r is set to 4 with T equal to 10000. (c) Sequence inconsistency observed in empirical training.

$f_t \in \mathbb{R}$ is also convex. Let $\mathbf{x}^* \in \mathbb{R}^d$ be the minimizer of this function. Assume that f_t are L -Lipschitz and that the distance between two points \mathbf{x} and \mathbf{x}' is bounded: $D(\mathbf{x}|\mathbf{x}') = \frac{1}{2} \|\mathbf{x} - \mathbf{x}'\|_2^2 \leq F^2$, where F is constant. Let an update be given by $\mathbf{u}_t = -\eta_t \nabla f_t(\tilde{\mathbf{x}}_t)$ and the learning rate by $\eta_t = \frac{\sigma}{\sqrt{t}}$. We have bound:

$$\mathbb{P}\left(\frac{R[X]}{T} - \frac{1}{\sqrt{T}} \left(\sigma L^2 - \frac{2F^2}{\sigma}\right) - q \geq \delta\right) \leq e^{-\frac{T\delta^2}{c + \frac{b\delta}{3}}}, \quad (1)$$

where δ is a constant and $b \leq 4PTL\sigma$. The b term here is the upper bound on the random variables which are drawn from the lag distribution $f(r)$. The q and c are two values that are related to the mean and variance of \mathbf{y}_t . If we assume that $0 < a < 1$, then it can be proved that both q and c are bounded. Furthermore, if we assume with probability Φ that $\forall t. APL\sigma\gamma_t < O(T)$, then $b < O(T)$. That means $\frac{R[X]}{T}$ converges to $O(T^{-1/2})$ in probability with an exponential tail bound with probability Φ .

To put this theorem to plain words, it claims that, as long as the difference between the noisy update sequence and the ideal sequence is bounded, and that the nodes in the system do not lag behind too far away, PSP guarantees that (with probability) the difference between the result we get and the optimal result diminishes as more updates are generated by workers. The full proof of Theorem 1 is listed in the report [37].

5.1 How Effective is Sampling

One key step in proving Theorem 1 is to prove the sequence inconsistency \mathbf{y}_t is bounded. We have proved that the mean and variance of vector \mathbf{y}_t are both bounded. Specifically, the average inconsistency (normalised by sequence length T) is bounded by:

$$\frac{1}{T} \sum_{t=0}^T \mathbb{E}(\mathbf{y}_t) \leq S \left(\frac{r(r+1)}{2} + \frac{a(r+2)}{(1-a)^2} \right), \quad (2)$$

and the variances has a similar bound:

$$\frac{1}{T} \sum_{t=0}^T \mathbb{E}(\mathbf{y}_t^2) < S \left(\frac{r(r+1)(2r+1)}{6} + \frac{a(r^2+4)}{(1-a)^3} \right), \quad (3)$$

where

$$S = \frac{1-a}{F(r)(1-a) + a - a^{T-r+1}}. \quad (4)$$

These bounds can be treated as constants for fixed a , T , r and β . They provide a means to quantify the impact of the PSP sampling primitive and provide stronger convergence guarantees than ASP. They do not depend upon the entire lag distribution, which ASP does.

The intuition is that, when applying PSP, the update sequence we get is not too different from the true sequence. To demonstrate impact of the sampling primitive on bounds quantitatively, Figs. 7a and 7b show how increasing the sampling count, β , (from 1 to 128, marked with different line colours on the right) yields tighter bounds. Notably, only a small number of nodes need to be sampled to yield bounds close to the optimal. This result has an important implication to justify using sampling primitive in large distributed learning systems due to its effectiveness. This will be further verified in the evaluation section.

The discontinuities at $a = 0$ and $a = 1$ reflect edge cases of the behaviour of the barrier method control. Specifically, with $a = 0$, no probability mass is in the initial r steps so no progress can be achieved if the system requires $\beta > 0$ workers to be within r steps of the fastest worker. If $a = 1$ and $\beta = 0$, then the system is operating in ASP mode so the bounds are expected to be large. However, these are overly generous. Better bounds are $O(T)$ for the mean and $O(T^2)$ for the variance, which we give in our proof. When $a = 1$ and $\beta \neq 0$, the system should never wait and workers could slip as far as they like as long as they returned to be within r steps before the next sampling point.

Besides theoretical analysis, an intuitive visualisation of sequence inconsistency \mathbf{y}_t is shown in Fig. 7c. We run a distributed training experiment with various barrier methods for 100 seconds, and measure the number of difference between true and noisy sequence at a fixed interval during the whole process. The result shows that the sequence inconsistency using ASP keeps growing linearly, while in SSP it increases and decrease within a certain bound, which is decided by the staleness parameter. Applying sampling to SSP relaxes that bound, but unlike ASP, inconsistencies using pSSP grows sub-linearly with sequence length. BSP is omitted in the figure, since its true and noisy sequence is

always the same. pBSP shows a tight bound (about 0.5) even with only 5% sampling.

6 EVALUATION

In this section, we investigate performance of the proposed PSP in experiments. We focus on two common metrics in evaluating barrier strategies: the accuracy Section 6.2 and system progress Section 6.3. Using these metrics, we explore the impact of sample settings Section 6.5, and stragglers Section 6.4 in the Federated Learning system and the proposed PSP. Besides, we also propose to use the *progress inconsistency* as a metric of training accuracy, but without the impact of specific application hyper-parameters Section 6.6. We compare PSP with the other barrier methods, including ASP from [7], BSP used in the paper [15], and SSP proposed in [8].

6.1 Setup

We perform extensive experiments on real-world dataset FEMNIST, which is part of LEAF, a modular benchmarking framework for learning in federated settings and includes a suite of open-source federated datasets [38]. Similar to MNIST, the FEMNIST dataset is for image classification tasks. But it contains 62 different classes (10 digits, 26 lowercases, and 26 uppercase). Each image is of size 28 by 28 pixels. The dataset contains 80,5263 samples in total. The number of samples distributed evenly across different classes.

The CNN model we use has two convolutional layers and two fully-connected layers. The two convolutional layers have 32 and 64 output channels respectively. Both have 3x3 kernel size and stride size of 1. Both layers are then followed by a Relu layer and a dropout layer. The first fully-connected layer has 128 units activated by ReLu, and the second fully-connected layer has output size of 62, followed by a softmax output layer. During training, the loss function is categorical cross-entropy and the measurement metric of model accuracy is top-1 accuracy.

To better study the performance of the proposed method with non-IID data distribution in Federated Learning, we follow the data partition setting in [31]. We first sort the data by class labels, divide them into $2n$ shards, and assign each of n workers 2 shards. This pathological non-IID partition makes the training data on different workers overlap as little as possible. The validation set is 10% of the total data. Besides, we pre-process it so that the validation set is roughly balanced in each class. As for training hyper-parameters, we use a batch size of 128, and we use the Adam optimiser [39], with learning rate of 0.001, and coefficient of (0.9, 0.999).

We conduct our experiment on a server that has 56 Intel (R) Xeon(R) CPU E5-2680 v4, and a memory of 256G. In the rest of this section, if not otherwise mentioned, we use 16 workers by default. Besides, one extra worker is used for model validation to compute its accuracy. In the rest of this section, we aim to show the wide range of tuning space enabled by the sampling parameter, and how existing barrier methods can be incorporated into PSP.

6.2 Accuracy

First, we demonstrate the performance advantage of PSP compared to the existing barrier control methods. To do that, we execute the training process using each method on

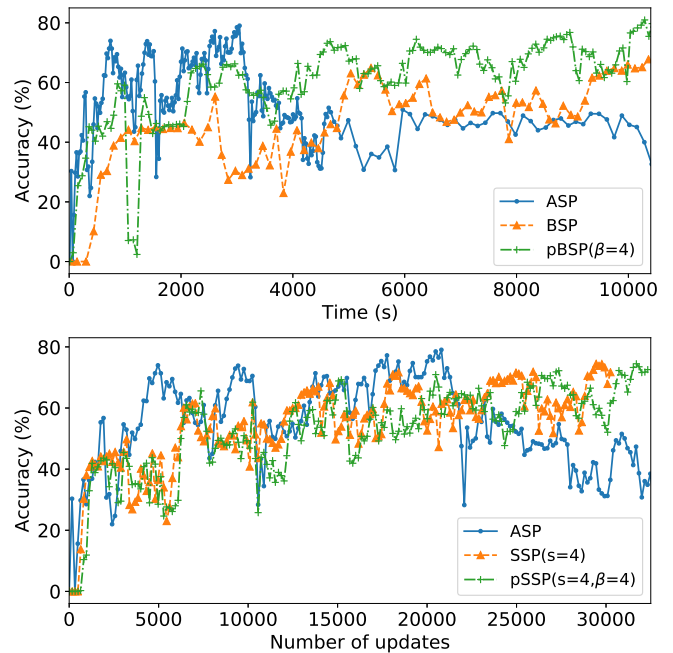


Fig. 8. Performance comparison between different synchronisation methods.

the non-IID FEMNIST dataset for about 8 epochs. The results are shown in Fig. 8. The sub-figure uses time as the x-axis. It shows the change of trained model accuracy in about 10,000 seconds. It compares the ASP, BSP, and pBSP (composing PSP with BSP) where the sampling size equals 4. Following the setting of SSP in [8], the staleness parameter of pSSP is also set to 4.

The first thing to note here is, though the performance of ASP looks optimal at the beginning due to its quick accumulation of updates from different workers, it quickly deteriorates and fails to converge. Compared to the unstable performance of ASP, BSP steadily converges. Then the pBSP clearly outperforms these two regarding model accuracy, especially in the later part of training. Due to its probabilistic nature, the pBSP line shows larger jitters than BSP, but also follows the general trend of BSP towards convergence steadily.

The strength of PSP lies in that it combines the advantages of existing methods. In the lower sub-figure Fig. 8b, we use the accumulated total number of updates the Parameter Server has received as x-axis to compare the “efficiency” of the updates in ASP, SSP, and pSSP. The staleness parameter of SSP and pSSP is set to 4 here. We can see that as updates are accumulating, despite using sampling, the accuracy increase of pSSP is similar to that of SSP.

Meanwhile, pSSP is much faster than SSP with regard to the update progress, or the rate at which the updates accumulate at the Parameter Server. Fig. 9 shows the number of updates at the server with regard to time (here we show only results from the beginning of evaluations). As can be seen, at any given time, both pBSP and pSSP progress faster than BSP and SSP correspondingly. Of course, ASP progresses the fastest since it does not require any synchronisation among workers, but its non-converged updates makes this advantage obsolete.

Note that the difference of number of updates can be directly interpreted as the communication cost, since each

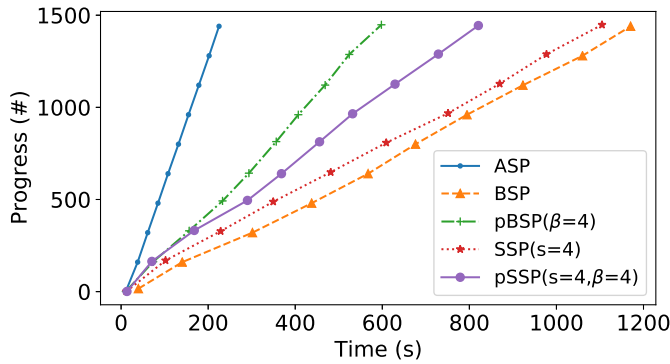


Fig. 9. Number of updates accumulated at the parameter server for different barrier methods.

update means the transmission of weight and gradient between the server and clients. For example, *at about 600s, the pSSP enables 35% more traffic than SSP; and pBSP even doubles the traffic in BSP.* According to our previous evaluation result, the proposed PSP achieves such reduction without sacrificing the final model accuracy.

Results. PSP combines the best of two worlds. On one hand, it has similar update efficiency as SSP and BSP; on the other hand, it achieves faster update progress that is similar to ASP. As a result, it outperforms the existing barrier control methods.

6.3 System Progress

In this section, we are going to further investigate iteration speed PSP achieves. We use 32 workers, and run the evaluation for 400 seconds. Fig. 10a shows the distribution of all nodes' progress, i.e., the local iteration number of nodes after training finishes.

As expected, the most strict BSP leads to a tightly bounded progress distribution, but at the same time, using BSP makes all the nodes progress slowly. At the end of the experiment, all the nodes only proceed to about the 80th update. As a comparison, using ASP leads to a much faster progress of around 200 updates. But the cost is a much loosely spread distribution, which shows no synchronisation at all among nodes. SSP allows certain staleness (4 in our experiment) and sits between BSP and ASP.

PSP shows another dimension of performance tuning. We set sample size β to 4, i.e. a sampling ratio of only 12.5%. The result shows that *pBSP is almost as tightly bound as BSP and also much faster than BSP itself.* The same is also true when comparing pSSP and SSP. In both cases, PSP improves the iteration efficiency while limiting dispersion.

To further investigate the impact of sample size, we focus on BSP, and choose different sample sizes. In Fig. 10b, we vary the sample size from 0 to 24. As we increase the sample size, the curves start shifting from right to left with tighter and tighter spread, indicating less variance in nodes' progress. With sample size 0, the pBSP exhibits exactly the same behaviour as that of ASP; with increased sample size, pBSP starts becoming more similar to SSP and BSP with tighter requirements on synchronisation.

Another interesting thing we notice in the experiment is that, with a very small sample size of one or two (i.e., very small communication cost on each individual node), pBSP can already effectively synchronise most of the nodes compared to ASP. The tail caused by stragglers can be further

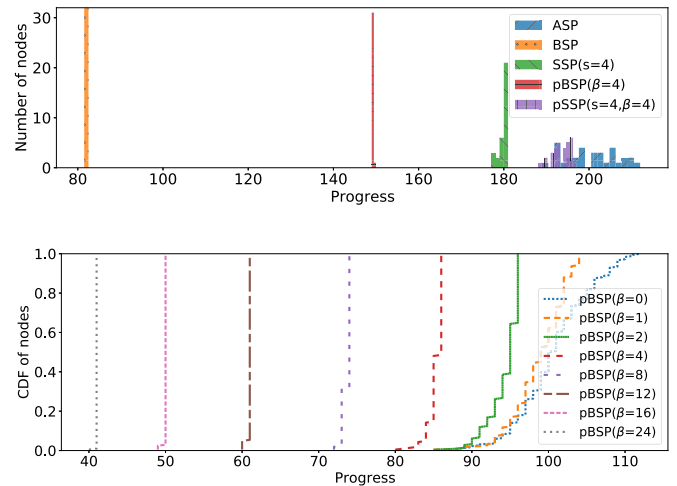


Fig. 10. (a) System progress distribution; (b) pBSP parameterised by different sample sizes, from 0 to 24. Increasing the sample size make the curves shift from right to left with decreasing spread, covering the whole spectrum from the most lenient ASP to the most strict BSP.

trimmed by using larger sample size. This observation confirms our theoretical analysis in Section 5, which explicitly shows that a small sample size can effectively push the probabilistic convergence guarantee to its optimum even for a large system size, which further indicates the superior scalability of the proposed solution.

Results. When composed with BSP, PSP can increase the system progress of BSP by about 85%, while retaining the almost the same tight bound on progress distribution. Besides, by tuning the sample size, the evaluation result shows that a small size such as 2 or 4 in a system or 32 workers, can effectively provide a tight convergence guarantee.

6.4 Robustness to Straggler

Stragglers are not uncommon to see in traditional distributed training, and are pervasive in the workers of Federated Learning. In this section we show the impact of stragglers on system performance and accuracy of model updates, and how probabilistic synchronisation control by sampling primitive can be used to mitigate such impacts.

As has explained before, we model the system stragglers by increasing the training time of each slow trainer to n -fold, namely on average they spend n times as much time as normal nodes to finish one iteration. The parameter n here is the "slowness" of the system. In the experiment shown in Fig. 11, we keep the portion of slow nodes fixed, and increase the slowness from 2 to 8. Then we measure accuracy of using a certain barrier control method at the end of training. To be more precise, we choose a period of results before the ending and use their mean value and standard for each observation point.

Fig. 11 plots the decreasing model accuracy due to stragglers as a function of the straggler slowness. As we can see, both ASP and BSP are sensitive to stragglers, both dropping about 20% accuracy by increasing slowness from 2x to 8x, while that of pBSP only drops by less than 10%. For BSP, this is mainly because the stragglers severely reduces the training update progress; for ASP, this can be explained as the result of its asynchronous nature, where updates from slow workers are delayed. This problem is exacerbated by

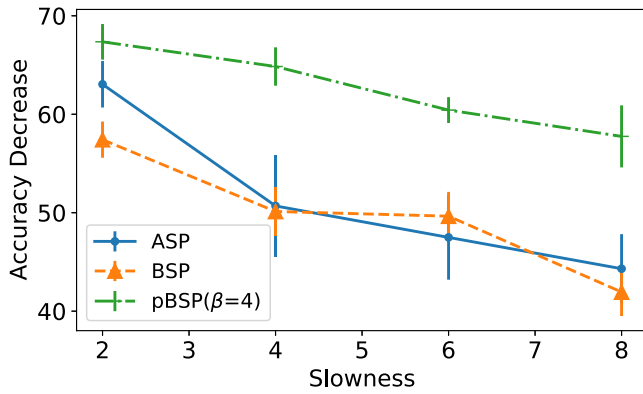


Fig. 11. Stragglers impact both system performance and accuracy of model updates. Probabilistic synchronisation control by sampling primitive is able to mitigate such impacts.

the non-IID data, where the data overlap between different workers is limited, if not none at all. Once again, PSP takes the best of both worlds. As we have shown before, its probabilistic sampling mitigates the effect of data distribution, and also less prone to the progress reduction caused by stragglers.

Results. PSP is less prone to the stragglers in the system. When the slowness increases from 2x to 8x, both ASP and BSP are sensitive to stragglers, both dropping about 20% accuracy, while that of pBSP only decreases by less than 10%.

6.5 Sampling Settings

In section 6.3, we investigate how the choice of sampling size affects the progress in PSP. One question is then: how to choose the suitable sample size? As pointed out in section 5, one important observation that can be derived from our theory proof is that, a small number of sampling can achieve similar performance as that using large sample numbers.

To demonstrate this point in evaluation, we choose different numbers of sampling size, from 2 to 8, in a 16-worker training, and compare them to SSP. The training lasts for a fixed time for all the used methods. In Fig. 12, we can see that, *even though the number of samples changes, the performance of pSSP is still close to that of SSP.* In this scenario, choosing a smaller number of sampling leads to better performance than the others, due to its fast progress of updates. However, it is not a rule of thumb to always use a small sample size. Choosing suitable parameters in a wide tuning space enabled by PSP is non-trivial tasks, and we are working to illustrate this challenging problem in our future work.

Another thing to investigate is the choice of sampling strategy in PSP. In Section 3, we discuss three different

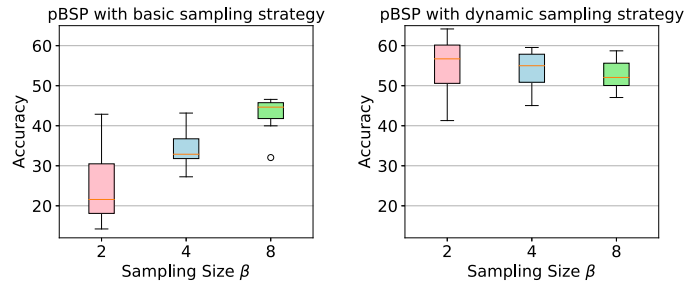


Fig. 13. Compare the different strategies of sampling in PSP.

sampling strategies. First is the basic strategy that chooses a certain number of workers as a subset and uses them to estimate training progress of all the workers. The second one, the *dynamic sampling* strategy, re-choose this subset dynamically instead of keeping it fixed. The third is a grouping strategy that pre-cluster workers into two groups according to their execution speed, and then chooses T samples equally from both groups.

To compare these strategies, we use 24 workers, and set 6 of them to be stragglers (1x slower in computing back-propagation). We use pBSP, and increase its sampling size from 2 to 8. Both training use the same number of epochs. The results are shown in Fig. 13. Each box shows the distribution of model accuracy numbers near the end of each training; the last 10 results are used in this experiment.

The result shows that, compared to the basic strategy, the dynamic one can effectively increase the efficiency of PSP. The increase ranges from about 25% to 2 fold for different sampling sizes. The low accuracy of the basic strategy shows that it tends to result in a more asynchronous training, which is more similar to ASP than BSP. The grouping strategy achieves similar results as the dynamic one, but shows smaller deviation of box, which means a smoother curve in training (result figure omitted due to space limit). Besides, in dynamic strategy, the sampling size does not visibly affect the model accuracy, which means that the smaller sample size can be used to increase system progress without sacrificing model accuracy. Also note that in both cases, larger sampling size leads to smaller deviation. This also agrees with the design and analysis of PSP shown in previous sections.

Results. First, by varying the setting of sampling of size from 2 to 8 in pSSP by using a worker size of 16, it can be seen that a small sampling size can still achieve that of a large one, regarding model accuracy. Second, the proposed dynamic and grouping sampling strategy can both effectively improve the performance. Compared to the basic strategy, both can effectively increase the efficiency of PSP. The increase ranges from about 25% to 2 fold for different sampling sizes.

6.6 Progress Inconsistency

In the previous section we have evaluated the impact of barrier control methods on the accuracy of three different models. However, the training accuracy is affected not only by the barrier method, which controls training inconsistency, but also hyper-parameters such as learning rate. The tolerance of error in training for different applications also varies greatly. To better understand the impact of barriers on model consistency during training without considering the

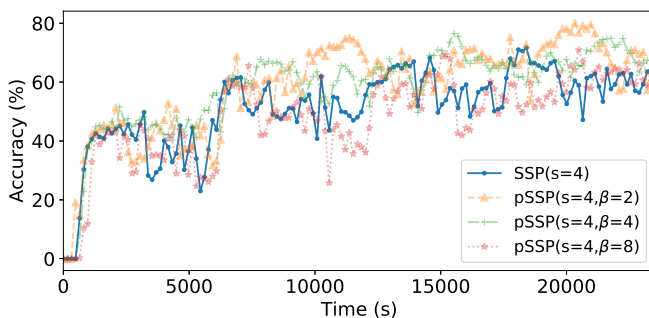


Fig. 12. Varying sample size in pSSP.

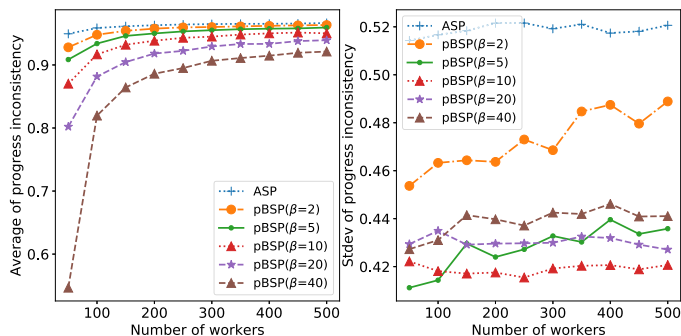


Fig. 14. Average and variance of normalised progress inconsistency in PSP with regard to sample size (100 nodes in total).

influence of these factors, we use *progress inconsistency* as a metric to compare barriers.

In distributed training, for a worker, between the time it pulls a model from a server and updates its own local model, the server likely has already received several updates from other workers. These updates are the source of training inconsistency. We define *progress inconsistency* as the number of these updates between a worker's corresponding read and update operations. In this experiment, we collect the progress inconsistency value of each node at its every step during training.

We investigate the relationship between the number of nodes and inconsistency of pBSP. All executions run for 100 seconds, and we increase workers from 50 to 500. We measure the average and variance of progress inconsistency, both normalised with number of workers, as shown in Fig. 14. The average inconsistency of ASP is mostly unaffected by size. With smaller sample size, that of pBSP becomes close to ASP, but note that only the initial increase of network size has a considerable impact. With sample size fixed and network size growing, the average inconsistency grows sub-linearly, which is an ideal property. As to the standard deviation values of pBSP, they mostly keep stable regardless of network size.

Results. According to these observations, we can see that for PSP, both the average training inconsistency (denoted by mean) and the noise (denoted by variance) grow sub-linearly towards a certain limit for different sample size, limited by that of ASP and BSP/SSP.

7 CONCLUSION

In this paper, we propose a novel barrier control method called Probabilistic Synchronous Parallel. PSP is suitable for data analytic applications deployed in large and unreliable distributed systems such as Federated Learning. The proposed PSP strikes a good trade-off between the efficiency and accuracy of iterative learning algorithms by probabilistically controlling how data is sampled from distributed workers. We implement the proposed PSP on the state-of-the-practice distributed learning systems, with a core system primitive of "sampling". We show that the *sampling* primitive can be combined with existing barrier control methods to derive fully distributed solutions. We evaluate the solution both analytically and experimentally in a realistic setting and our results show that PSP outperforms existing barrier control solutions in various settings.

The effectiveness of PSP in different application scenarios depends on the suitable parameter, i.e., the sample size. Similar to the performance tuning in numerical computation, we suggest resorting to prior knowledge and empirical measurement for its parameter tuning and regard this as the challenge for the future exploration.

ACKNOWLEDGMENTS

This work was supported by TravelSky Technology Limited. Benjamin Catterall participated in this work when he was a student in the University of Cambridge.

REFERENCES

- [1] Q. Li, Z. Wen, Z. Wu, S. Hu, N. Wang, and B. He, "A survey on federated learning systems: Vision, hype and reality for data privacy and protection," 2019, *arXiv:1907.09693*.
- [2] K. Bonawitz *et al.*, "Towards federated learning at scale: System design," 2019, *arXiv:1902.01046*.
- [3] Y. M. Saputra, D. T. Hoang, D. N. Nguyen, E. Dutkiewicz, M. D. Mueck, and S. Srikanteswara, "Energy demand prediction with federated learning for electric vehicle networks," in *Proc. IEEE Glob. Commun. Conf.*, 2019, pp. 1–6.
- [4] X. Wang, Y. Han, C. Wang, Q. Zhao, X. Chen, and M. Chen, "In-Edge AI: Intelligentizing mobile edge computing, caching and communication by federated learning," *IEEE Netw.*, vol. 33, no. 5, pp. 156–165, Sep./Oct. 2019.
- [5] T. S. Brisimi, R. Chen, T. Mela, A. Olshevsky, I. C. Paschalidis, and W. Shi, "Federated learning of predictive models from federated electronic health records," *Int. J. Med. Inform.*, vol. 112, pp. 59–67, 2018.
- [6] A. Hard *et al.*, "Federated learning for mobile keyboard prediction," 2018, *arXiv:1811.03604*.
- [7] B. Recht, C. Re, S. Wright, and F. Niu, "HOGWILD: A lock-free approach to parallelizing stochastic gradient descent," in *Proc. Neural Inf. Process. Syst.*, 2011, pp. 693–701.
- [8] Q. Ho *et al.*, "More effective distributed ML via a stale synchronous parallel parameter server," in *Proc. Neural Inf. Process. Syst.*, 2013, pp. 1223–1231.
- [9] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.
- [10] M. Li *et al.*, "Scaling distributed machine learning with the parameter server," in *Proc. USENIX Symp. Oper. Syst. Des. Implementation*, 2014, pp. 583–598.
- [11] K. Shvachko *et al.*, "The hadoop distributed file system," in *Proc. IEEE Symp. Mass Storage Syst. Technol.*, 2010, pp. 1–10.
- [12] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *Proc. USENIX Conf. Hot Topics Cloud Comput.*, 2010, Art. no. 95.
- [13] A. Ahmed, M. Aly, J. Gonzalez, S. Narayanamuthy, and A. Smola, "Scalable inference in latent variable models," in *Proc. ACM Int. Conf. Web Search Data Mining*, 2012, pp. 123–132.
- [14] K. Hsieh *et al.*, "Gaia: Geo-distributed machine learning approaching LAN speeds," in *Proc. USENIX Symp. Netw. Syst. Des. Implementation*, 2017, pp. 629–647.
- [15] M. Duan, D. Liu, X. Chen, R. Liu, Y. Tan, and L. Liang, "Self-balancing federated learning with global imbalanced data in mobile systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 1, pp. 59–71, Jan. 2021.
- [16] J. Park *et al.*, "Communication-efficient and distributed learning over wireless networks: Principles and applications," *Proc. IEEE*, vol. 109, no. 5, pp. 796–819, Feb. 2021.
- [17] X. Qiu, W. Zhang, W. Chen, and Z. Zheng, "Distributed and collective deep reinforcement learning for computation offloading: A practical perspective," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 5, pp. 1085–1101, May 2020.
- [18] Z. Li, M. Xu, J. Nie, J. Kang, W. Chen, and S. Xie, "NOMA-enabled cooperative computation offloading for blockchain-empowered internet of things: A learning approach," *IEEE Internet Things J.*, vol. 8, no. 4, pp. 2364–2378, Feb. 2020.
- [19] E. P. Xing, Q. Ho, P. Xie, and D. Wei, "Strategies and principles of distributed machine learning on big data," *Engineering*, vol. 2, no. 2, pp. 179–195, 2016.

- [20] H. Cui *et al.*, "Exploiting bounded staleness to speed up big data analytics," in *Proc. USENIX Annu. Tech. Conf.*, 2014, pp. 37–48.
- [21] X. Lian, Y. Huang, Y. Li, and J. Liu, "Asynchronous parallel stochastic gradient for nonconvex optimization," in *Proc. Neural Inf. Process. Syst.*, 2015, pp. 2737–2745.
- [22] C. M. De Sa, C. Zhang, K. Olukotun, and C. Ré, "Taming the wild: A unified analysis of HOGWILD-style algorithms," in *Proc. Neural Inf. Process. Syst.*, 2015, pp. 2674–2682.
- [23] S. Zheng *et al.*, "Asynchronous stochastic gradient descent with delay compensation," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 4120–4129.
- [24] W. Dai, A. Kumar, J. Wei, Q. Ho, G. Gibson, and E. P. Xing, "High-performance distributed ML at scale through parameter server consistency models," in *Proc. AAAI Conf. Artif. Intell.*, 2015, pp. 79–87.
- [25] W. Zhang, S. Gupta, X. Lian, and J. Liu, "Staleness-aware async-SGD for distributed deep learning," 2015, *arXiv:1511.05950*.
- [26] J. Chen *et al.*, "Revisiting distributed synchronous SGD," in *Proc. Int. Conf. Learn. Representations*, 2017, pp. 1–10.
- [27] J. Wang and G. Joshi, "Cooperative SGD: A unified framework for the design and analysis of communication-efficient SGD algorithms," 2018, *arXiv:1808.07576*. [Online]. Available: <http://arxiv.org/abs/1808.07576>
- [28] P. Kairouz *et al.*, "Advances and open problems in federated learning," 2019, *arXiv:1912.04977*.
- [29] W. Y. B. Lim *et al.*, "Federated learning in mobile edge networks: A comprehensive survey," *IEEE Commun. Surv. Tut.*, 2020, pp. 1–34.
- [30] T. Nishio and R. Yonetani, "Client selection for federated learning with heterogeneous resources in mobile edge," in *Proc. IEEE Int. Conf. Commun.*, 2019, pp. 1–7.
- [31] H. Brendan McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Arcas, "Communication-efficient learning of deep networks from decentralized data," *Proc. Int. Conf. Artif. Intell. Statist.*, vol. 54, pp. 1273–1282, 2017.
- [32] V. Smith, C. K. Chiang, M. Sanjabi, and A. Talwalkar, "Federated multi-task learning," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 4425–4435.
- [33] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," 2018, *arXiv:1812.06127*. [Online]. Available: <http://arxiv.org/abs/1812.06127>
- [34] M. Assran, N. Loizou, N. Ballas, and M. Rabbat, "Stochastic gradient push for distributed deep learning," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 514–523.
- [35] X. Li, K. Huang, W. Yang, S. Wang, and Z. Zhang, "On the convergence of FedAvg on Non-IID Data," 2019, *arXiv:1907.02189*. [Online]. Available: <http://arxiv.org/abs/1907.02189>
- [36] J. Cipar *et al.*, "Solving the straggler problem with bounded staleness," in *Proc. USENIX Conf. Workshop Hot Topics Oper. Syst.*, 2013, p. 22, doi: [10.5555/2490483](https://doi.org/10.5555/2490483).
- [37] L. Wang, B. Catterall, and R. Mortier, "Probabilistic synchronous parallel," 2017, *arXiv:1709.07772*.
- [38] S. Caldas *et al.*, "LEAF: A benchmark for federated settings," 2018, *arXiv:1812.01097*.
- [39] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.



Jianxin Zhao received the bachelor's and master's degrees in software engineering from the Beijing Institute of Technology, Beijing, China, in 2013 and 2015, respectively, and the PhD degree in computer science from the University of Cambridge (affiliated with Corpus Christi College), supervised by professor Jon Crowcroft. He is currently a postdoc with the Beijing Institute of Technology, supervised by professor Chi H. Liu. His research interests include numerical computation, high performance computing, machine learning, and their applications in the real world.



Rui Han received the MSc degree (Hons.) in 2010 from Tsinghua University, China, and the PhD degree from the Department of Computing, Imperial College London, UK., in 2014. He is currently an associate professor with the School of Computer Science and Technology, Beijing Institute of Technology, China. His research interests include cloud resource management and system optimization for data centre workloads. He has authored or coauthored more than 40 publications in these areas, including papers at the *IEEE Transactions on Parallel and Distributed Systems*, *INFOCOM*, *ICDCS*, *ICPP*, *CCGrid*, and *CLOUD*.



Yongkai Yang received the master's degree in engineering from Beijing Jiaotong University in 2003. He was with civil aviation Informationization, especially in GDS and PSS, airlines e-commerce, fare systems and revenue systems. Since 2014, he has led the team to develop TavelSky, a high performance civil aviation and international shopping system which reaches the international advanced level, where he is currently the manager with the Pricing and Shopping System Department. His research interests include big data, data mining, cloud computing, and operational research in the civil aviation.



Benjamin Catterall received the BA and MEng degrees in computer science from the University of Cambridge in 2013 and 2017, respectively. He is currently a software engineer with the High Frequency Trading Industry. His research interests include developing theory and constructing simulations for distributed machine learning models using the probabilistic synchronous parallel barrier method and evaluating high throughput neural network implementations on multi-core platforms.



Chi Harold Liu (Senior Member, IEEE) received the BEng degree in electronic and information engineering from Tsinghua University, China, in 2006 and the PhD degree in electronic engineering from Imperial College, U.K., in 2010. He is currently a full professor and the vice dean with the School of Computer Science and Technology, Beijing Institute of Technology, China. He has authored or coauthored more than 90 prestigious conference and journal papers and owns 17 patents. His research interests include the big

data analytics, mobile computing, and machine learning. He was an associate editor for the *IEEE Transactions on Network Science and Engineering*, the area editor for the *KSII Transactions on Internet and Information Systems*, the symposium chair for IEEE ICC 2020 on next generation networking, and was the Lead guest editor for the *IEEE Transactions on Emerging Topics in Computing* and *IEEE Sensors Journal*. He was also the book editor for nine books published by Taylor & Francis Group, USA, and China Machine Press, China. He also was the general chair of IEEE SECON'13 workshop on IoT networking and control, IEEE WCNC'12 workshop on IoT enabling technologies, and ACM UbiComp'11 workshop on networking and object memories for IoT. He is a fellow of IET, British Computer Society, and Royal Society of Arts. He was the recipient of the IBM First Plateau Invention Achievement Award in 2012 and IEEE DataCom'16 Best Paper Award.



Lydia Y. Chen (Senior Member, IEEE) received the BA degree from National Taiwan University and the PhD degree from the Pennsylvania State University. From 2007 to 2018, she was a research staff member with the IBM Zurich Research Lab. She is currently an associate professor with the Department of Computer Science, the Technology University Delft. She has authored or coauthored more than 80 papers in journals, including the *IEEE Transactions on Distributed Systems*, *IEEE Transactions on Service Computing*, and conference proceedings, including the INFOCOM, Sigmetrics, DSN, and Eurosys. Her research interests include dependability management, resource allocation, and privacy enhancement for large scale data processing systems and services. She was the corecipient of the Best Paper Awards at CCgrid'15 and eEnergy'15.



Richard Mortier is reader in computing and human-data interaction with the Cambridge University Computer Lab. He was in a variety of roles, including academic research, Internet platform architecture, and networked system implementation. He has also consulted and worked for a broad range of companies, including startups and corporates in both the U.S. and U.K. He is currently working at the intersection of systems and networking with human-computer interaction, and is focused on how to build user-centric systems infrastructure that enables people to better support themselves in a ubiquitous computing world through human-data interaction. His past research interests include internet routing, distributed system performance analysis, network management, aesthetic designable machine-readable codes, and home networking.



Jon Crowcroft (Fellow, IEEE) received the graduation degree in physics from Trinity College, University of Cambridge in 1979, and the MSc degree in computing and the PhD degree from University College London in 1981 and 1993, respectively. Since October 2001, he has been the Marconi professor of communications systems with the Computer Laboratory. From 2016 to 2018, he was the programme chair with the Turing, the U.K.'s National Data Science and AI Institute, where he is currently a researcher-at-large. He has worked in the area of Internet support for multimedia communications for more than 30 years. His three main topics of interest have been scalable multicast routing, practical approaches to traffic management, and the design of deployable end-to-end protocols. His current research interests include opportunistic communications, social networks, privacy preserving analytics, and techniques and algorithms to scale infrastructure-free mobile systems. He leans towards a build and learn paradigm for research. He is a fellow of the the Royal Society, ACM, British Computer Society, IET, and the Royal Academy of Engineering.



Liang Wang received the BEng degree in computer science and mathematics from Tongji University, Shanghai, China, in 2003, and the MSc and PhD degrees in computer science from the University of Helsinki, Finland, in 2011 and 2015, respectively. He is currently a research associate with the Computer Laboratory, University of Cambridge, U.K. His research interests include system and network optimization, modelling and analysis of complex networks, information-centric networks, and distributed data processing.

▷ For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/csdl.