PLATO: An Open-Source Research Framework for Production Federated Learning

Baochun Li, Ningxin Su, Chen Ying, Fei Wang
Department of Electrical and Computer Engineering
University of Toronto

***Abstract*—As existing works on federated learning (FL) have not typically shared their implementations as open-source, and existing open-source FL frameworks fell short of evaluating FL mechanisms appropriately, in the past two years, we have designed and implemented PLATO, a new open-source research framework for scalable federated learning research from scratch. Development on PLATO started in November 2020, and so far involved more than 50 person-month of research and development time. PLATO is designed and built with several key objectives in mind: it is *scalable* to a large number of clients; *extensible* to accommodate a wide variety of datasets, models, and FL algorithms; and *agnostic* to deep learning frameworks such as TensorFlow and PyTorch. In PLATO, clients communicate with servers over industry-standard WebSockets, while servers may either run in the same GPU-enabled physical machine as its clients — suitable for an emulation research testbed — or deployed in a cloud datacenter. We provided a large variety of popular datasets and models, as well as algorithms proposed in the literature as examples.**

I. INTRODUCTION

In reviewing the literature in federated learning (FL), it is hard to overlook the fact that there does not exist a federated learning framework that is the *de facto* standard on evaluating the performance of newly proposed mechanisms. Each proposed work typically used its own proprietary testbed for the purpose of performance evaluations, and did not typically share their testbed implementation as open source, making it almost impossible to reproduce their experimental results. But such a glaring lack of *reproducibility* is not the only issue outstanding in the existing works: they have not been compared using a consistent set of performance metrics. Various performance metrics, such as the number of transmitted local updates or communication rounds till convergence or reaching a target accuracy, have been employed in previous studies. Nevertheless, none of them were shown to be sufficiently representative when it comes to the two fundamental metrics of convergence: the amount of time it takes for the global model to converge, and the validation accuracy that the global model converged to.

These problems motivated us to design and implement a new open-source FL research framework that is easy to run, scalable to an unlimited number of clients, easy to extend to new FL mechanisms, and accessible to a wide variety of existing machine learning models. In the past two years, we invested over 50 person-months of research time, and designed PLATO, a new open-source research framework for scalable FL research, from scratch. It is currently available as open-source (under the Apache 2.0 license) at https://github.com/TL-System/plato.

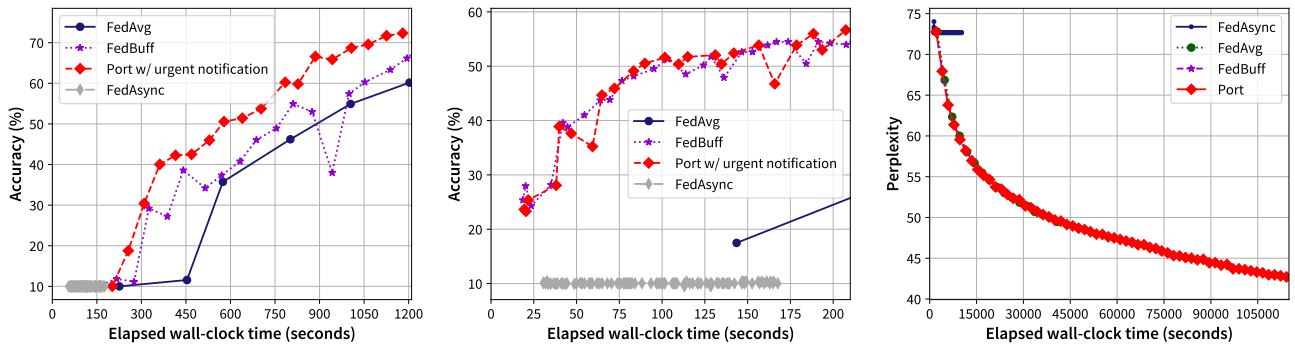
The research was supported in part by a RGC RIF grant under the contract R6021-20.

II. PLATO: DESIGN OBJECTIVES

With over two years of intensive work on PLATO, we have succeeded to achieve the following design objectives:

***Scalable* to an unlimited number of clients.** To scale up the number of clients that can be emulated with a limited amount of GPU memory, our implementation runs clients in batches, and each client's training loop is executed *in its own process*, so that GPU memory is guaranteed to be released after the process completes a round of training. PLATO is also able to take full advantage of multiple GPUs to further improve the level of concurrency.

***Extensible* to accommodate a wide variety of models, datasets, and FL methods.** PLATO provides access to a wide variety of existing machine learning models and datasets, including image classification and natural language processing. Moreover, all possible elements of a new FL mechanism — algorithms, clients, servers, model trainers, data sources, and data samplers — are implemented as extensible classes following the same design patterns of inheritance and callbacks, so that new mechanisms can be easily implemented.



(a) Selected algorithms with `CIFAR-10`. (b) Selected algorithms with `CINIC-10`. (c) Selected algorithms with `Tiny-Shakespeare`.

Fig. 1. Port, FedBuff, FedAsync and federated averaging with the `CIFAR-10`, `CINIC-10` and `Tiny-Shakespeare`, and a mild non-i.i.d. Dirichlet data distribution.

***Agnostic* to multiple machine learning frameworks.** So far, PLATO supports three widely used machine learning frameworks, PyTorch, TensorFlow, and MindSpore.

***Real-world* system implementations, not just emulations.** In PLATO, communication between the central server and clients in two-layer FL, as well as between the central server and edge servers and between edge servers and clients in three-layer FL, is over industry-standard WebSockets, which provide options to conduct not only emulations but also experiments with real-world system implementations, depending on where the central server runs. The servers can run on either a cloud datacenter in a production system, or in the same GPU-enabled physical machine along with its clients, which is suitable as an emulation research testbed. An advantage of such an implementation is that actual wall-clock time it takes to complete a training session can be accurately measured, including all the time required for communication.

***Reproducibility* in FL experiments.** In FL experiments, random number generators are used for sampling participating clients and local datasets. For fair comparisons across different FL mechanisms, improving the reproducibility of our experiments by seeding, saving, and restoring our random number generators is critically important. In PLATO, we can not only specify random seeds but also use `random.getstate()` and `random.setstate()` to protect random number generation from the effects of third-party frameworks.

III. PLATO: AN EXAMPLE

For fair comparison studies, a large number of examples were included in the PLATO GitHub repository, which implemented popular existing algorithms in the literature. When comparing existing algorithms, we focus on two performance metrics: (1) *Wall-clock time.* The number of communication rounds is not a suitable performance metric in production FL, since different rounds would take substantially different durations in wall-clock time. Instead, the wall-clock time elapsed before converging to a target accuracy is the most suitable performance metric that should be evaluated when evaluating new mechanisms. (2) *Accuracy.* The accuracy after convergence is arguably one of the important metrics as well.

As an example, PLATO is used to evaluate in the context of asynchronous federated learning PORT [1], a recently proposed asynchronous FL algorithm. Three datasets have been used for a comparison study: `CIFAR-10`, `CINIC-10` and `Tiny Shakespeare`. Three baseline algorithms, including Federated Averaging as a synchronous algorithm, as well as Fedbuff and FedAsync as asynchronous algorithms, were used for comparisons. Fig. 1 illustrates our experimental results obtained with PLATO, with a total number of 100 clients and non-i.i.d. settings, and using the elapsed wall-clock time and accuracy as the performance metrics in production FL. PLATO natively supports measuring the wall-clock times even in its emulation mode running clients in batches with limited GPU memory, using sophisticated mechanisms such as priority queues to sort clients by their finish times after multiple batches finish in the same communication round. It is also straightforward to generate non-i.i.d. distributions of data across clients, or to simulate varying training speeds across heterogeneous clients: all one needs to do is specify parameter settings using a configuration file.

REFERENCES

[1] N. Su and B. Li, "How Asynchronous can Federated Learning Be?" in *Proc. IWQoS*, 2022.