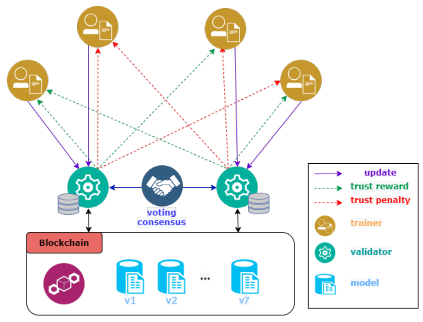
FLoBC: A Decentralized Blockchain-Based Federated Learning Framework

Mohamed Ghanem, Fadi Dawoud, Habiba Gamal, Eslam Soliman, Tamer El-Batt, Tamer El-Batt
The American University in Cairo
Email: oscar@aucegypt.edu, fadiadel@aucegypt.edu, habibabassem@aucegypt.edu, eslam98@aucegypt.edu,
tamer.elbatt@aucegypt.edu, hossam.sharara@aucegypt.edu

2022 Fourth International Conference on Blockchain Computing and Applications (BCCA) | 978-1-6654-9958-3/22/$31.00 ©2022 IEEE | DOI: 10.1109/BCCA55292.2022.9922258

***Abstract*—The rapid expansion of data worldwide invites the need for more distributed solutions in order to apply machine learning on a much wider scale. The resultant distributed learning systems can have various degrees of centralization. In this work, we demonstrate our solution FLoBC for building a generic decentralized federated learning system using the blockchain technology, accommodating any machine learning model that is compatible with gradient descent optimization. We present our system design comprising the two decentralized actors: *trainer* and *validator*, alongside our methodology for ensuring reliable and efficient operation of said system. Finally, we utilize FLoBC as an experimental sandbox to compare and contrast the effects of trainer-to-validator ratio, reward-penalty policy, and model synchronization schemes on the overall system performance, ultimately showing by example that a decentralized federated learning system is indeed a feasible alternative to more centralized architectures.**

Graphical Abstract



***Index Terms*—Byzantine Fault-Tolerance, Federated Learning, Blockchain, Decentralized Systems, Distributed Machine Learning, Privacy Preserving**

I. Introduction

In recent years, there has been an enormous corpus of research and development dedicated to and focused on accelerating the processes of machine learning in every conceivable shape or form. Long gone are the days when computer hardware was not adequate enough to handle already existing machine learning algorithms, yet it seems the rapidly growing demand for machine learning applications has put us in a situation that yet again undermines the capabilities of individual machines. Here, an intuitive question comes to mind: why not just have multiple machines collaborate on training the model at hand? That is, in fact, the idea that Google realized in their conception of Federated Learning (FL) [14]. The idea of federated learning solves another major problem: the scarcity of data at a single machine. This is solved through the collaboration of different nodes in the training, each using its local data then sharing its model updates, realizing a single model trained on the sum of their local data.

In this paper, we present an experimental framework FLoBC for building federated learning systems over blockchain. We further illustrate how it offers feasible solutions to common challenges in decentralized federated learning, namely, **privacy**, **efficiency**, and **Byzantine fault-tolerance**. FLoBC is a proof-of-concept used to spawn mini-systems that demonstrate several aspects of our solution on a relatively small scale, on which we, in later sections, conduct several experiments in order to verify the extent of its efficacy by comparing its performance to a control group for which the features in question are disabled.

II. Background and Related Work

Prior to the conception of *federated learning*, *blockchain* has been a prominent technology that aimed at realizing decentralized consensus. It is not hard to see the potential merit of combining blockchain with federated learning in order to achieve much more decentralization and privacy. There is a number of contributions in the field of decentralized federated learning, some of which involve blockchain while others employ other decentralized protocols. The core objective is to eliminate the need for a central server to gather user data and perform model training. The motivation for that is two-fold: on one hand, it provides more privacy, and on the other hand, it relaxes the minimum required amount of computational power by distributing computation across the network. Some of the existing contributions rely on an All-Reduce scheme in which each training node shares its updates with all the nodes in

[0]This paper is the result of BSc in Computer Engineering thesis at the American University in Cairo (Spring 2021)

the system resulting in a communication cost of $O(n^2)$ for n training nodes. One important aspect to leverage (wherever applicable) is network topology which can be exploited to reduce the communication cost. There are proposals that use the ring topology in [5], the tree topology in [11] and graph topology in [1], [10] to share the model updates of the node with its one-hop neighbors. However, all these approaches require multi-hops for the updates to reach all the nodes in the system, thus leading to slow convergence.

Following the same objective of reducing communication overhead, other contributions, such as [7], use the gossip protocol which relies on the fact that each peer sends its updates to another peer in the network, thus propagating the information to the whole network. This arguably puts too much responsibility on the trainer side to validate updates by other peers. To remedy this, our framework introduces validator nodes (similar to cryptocurrency miners) in order to alleviate the computational load of trainers and to have a clearer and easier-to-trace trust system.

Last but not least, and most relevant to our work, some existing work uses blockchain as the communication infrastructure of decentralized federated learning. Blockchain is used to facilitate uploading and tracking updates, to reward the users for participating in the training or the validation of the model, and to make the updates immutable and secure. In [12], the role of the central server in centralized federated learning is fulfilled by smart contracts. The node performs the local training, then the local model updates are sent to an elected consensus committee that verifies the updates and assigns scores to them. Subsequently, the updated global model is added to the blockchain. In [9] and [16], each training node is assigned a validator that validates said trainer's updates, computes proof of work (PoW) and rewards the trainer accordingly. After the updates are added to the blockchain, the miner is rewarded. [13] follows a similar algorithm but the trainer can send its updates to any validator and the rewards are given by the model owner, not the validator or the blockchain. [4] and [15] make use of a distributed peer-to-peer file sharing system to hold the model checkpoints. [4], [15], [16] are specially tailored for automotive in-vehicle training.
By contrast, our framework uses *proof of stake* (PoS) as a much more lightweight alternative to cryptographic PoW. In addition, our methodology is a generic one, hence, it is not bound to a specific task or purpose as long as the required consensus assumptions are met by the underlying network (refer to section III-B).

III. Methodology

In this section, we introduce our approach for designing a system that closely realizes a decentralized version of *Federated Learning*. In pursuit of that, a few principal standards were set forth to serve as a basis for features and solutions employed in the system. These standards are mainly:

- ***Generality***: The system design needs to be agnostic of the model specifics apart from it being compatible with gradient descent.
- ***Decentralization***: System politics shall be undertaken through decentralized consensus.
- ***No Data Sharing***: Nodes shall only communicate insights, never having to share their data.

We cover challenges as well as solutions pertaining to the performance of said distributed learning system in the context of supervised learning where we have a loss function that we aim to minimize, most prominently through gradient descent. Though it is mainly focused on computational aspects, the system also takes into account communicational efficiency. We shall discuss the three main dimensions of a distributed learning system: Parallelism, Degree of Centralization, and Synchronization.

A. Parallelism

Training an ML model is, in the great majority of cases, the heaviest part of the whole machine learning process [2], so how could distributed learning help break this down? For that, distributed learning generally offers two paradigms to parallelize the process: Model-Parallelism and Data-Parallelism [18], the latter of which was chosen for our system.

Model Parallelism: In model-parallelism, the model structure itself is split or distributed across multiple nodes, a paradigm often referred to as Split Learning [8]. For instance, in the case of a neural network, the network would be split at certain layers (called cut layers), then each trainer would be assigned a group of consecutive layers, and all layer groups are stitched together based on the layer order in the model architecture [6]. While Split Learning has the advantages of data privacy and enabling nodes with low computation power to participate in the training, it has some inherent sequentiality that ultimately limits its scalability [17]. For our purposes, the primary downside of model-parallelism is that it is model and algorithm specific, which makes it difficult to utilize for a general-purpose system for training.

Data Parallelism: In data-parallelism, each node trains the model in its entirety, but only on a subset of the training data, with each node doing this in parallel, then all generated model updates are aggregated into the global model. This approach has the merit of better parallel processing on stochastic processes such as stochastic gradient descent (SGD) which is widely used in distributed learning. Moreover, it can be readily generalized to any process that lends itself to parallelism such as gradient computation and approximation [3]. As such, we opted for this model of parallelism because it greatly simplifies the general model structure to a flat array of weights, from which the full model can be reconstructed.

B. Degree of Centralization

It is not hard to see that the two ends of this spectrum are: fully centralized to fully decentralized. Accordingly, a strictly fully centralized system would simply be a single device doing all of the training process from start to finish. A more distributed and less centralized version of that would be the classical federated learning with a central server or cluster of servers supervising the training done by other nodes.

Here, we present a system on a much higher degree of decentralization. Similar to most cryptocurrency networks, our system comprises two types of nodes: **trainer** and **validator**, the latter of which is analogous to a crypto-miner. Validators are responsible for maintaining models on the underlying blockchain network through decentralized consensus based on voting. Ideally, a validator's vote, on whether some update to the model (made by a *trainer* node) is acceptable, should be decided by validation on data presumably unrevealed to the trainer. That, combined with the voting-based nature of consensus, stipulates the following requirements and assumptions in order for the system to operate soundly:

- Strictly more than 2/3 of validators are non-Byzantine.
- Different validators' datasets should be sufficiently positively correlated in order to reach a meaningful consensus.
- The underlying network is at least partially synchronous.
- The quality of models produced by the system is limited by the amount of collective useful data used during training and validation.

One serious implication of the system's centralization degree is the choice of the gradient descent (or any optimization technique) variant that suits how the system is structured. Since the scope of this study is focused on distributed learning, we are only going to consider the distributed centralized version and its fully decentralized nemesis. As such, we won't be concerned with traditional gradient descent (as it is impossible to use for such a distributed system), but instead, a stochastic variant of it should be employed to approximate the true gradient bearing in mind the rate at which this approximation converges to the true value [3]. Much like most federated learning systems, our system uses ***Federated Averaging*** to perform parallel stochastic gradient descent (P-SGD). At its core, it follows a very simple assumption: different nodes training on different datasets whose updates get aggregated or averaged onto a single model would eventually converge to a valid global model.

C. Synchronization

One crucial aspect of the distributed learning process is that distributed model updates happen in tandem to prevent model degradation due to out of sync parameters [18]. Like many other parallel and massively parallel paradigms, the concept of a synchronization barrier applies to the case of a distributed learning system. In this context, each barrier would signal an update-sharing round in which updates are aggregated into an updated model. The more frequently synchronization barriers occur, the faster the model converges, and the less it degrades [8]. However, here lies an important tradeoff between rate of convergence and communicational cost. That is, each barrier incurs a price on the network and node bandwidth. The main scheme here is that the system has two types of phases: a computation phase, and communication phase for sharing the computational results. For that purpose, there are many techniques to handle node update synchronization, and the following subsections introduce the most common ones.

Bulk Synchronous Parallel (BSP): Abbreviated as BSP, it is known to be the simplest approach to preserve consistency by alternating between rounds of computation followed by communication. The main pro of this model is that it has the best consistency (hence, fastest convergence), yet its main bottleneck is that finished nodes have to wait for all worker nodes to finish their computation at each synchronization barrier [18]. In our variant of BSP, each training round is limited by a fixed period that ends with a strict deadline after which no further trainer updates are considered for that round.

Stale Synchronous Parallel (SSP): This model is similar to BSP but with more relaxed synchronization barriers that allow a certain number of iterations beyond the preset number. After that leeway is reached, all workers are paused [8]. Given its nature as a compromise to BSP, it has good convergence insurances in low to moderate staleness, beyond which convergence rates start to decay [18]. In our design, SSP is modelled as BSP with the possibility of a deadline extension which is a fraction of the original round period determined in proportion to the ratio of trainers that have not yet finished training for that round, denoted by *slack ratio*. Meanwhile, trainers that already finished by the deadline are allowed to continue training for at most N more steps before the deadline extension finishes.

Barrierless Asynchronous Parallel (BAP): This model represents the opposite end of the spectrum to BSP as it almost totally eliminates the cost of synchronization by allowing waitless, asynchronous communication between nodes, which achieves a very low overhead (hence, higher speedups), but it suffers from potentially slow and even incorrect convergence with increased delays [18]. In our variant of BAP, trainers are allowed to train for as long as a round may extend; that is, until a new model is released, trainers can keep advancing local training, periodically sharing it with a validator. A new model is released when a minimum ratio of labour has been submitted, and that's when trainers should pull in the new model.

Approximate Synchronous Parallel (ASP): Although our system does not support ASP, the scheme remains noteworthy. Unlike the SSP model, ASP is concerned with limiting parameter accuracy instead of staleness. It does so by ignoring updates that are not significant to spare the synchronization costs of them [18]. However, one downside of that approach is that it is not easy to decide which parameters are insignificant, along with increased complexity of implementation.

D. Byzantine Fault-Tolerance

Possible cases and scenarios of Byzantine behavior are countless, and decentralized systems are arguably most prone to it. In this work, we focus on lazy[1] and malicious trainer behavior given that malicious validator behavior is mostly taken care of by the blockchain's decentralized consensus. To remedy this, we employ a reward-penalty policy, under which each trainer i is assigned a trust score (or reputation) ϕ_i such that:

[1]Lazy, in this context, includes nodes that are not contributing useful updates to models.

$$0 \leq \phi_i \leq 1$$
$$\sum_{i=1}^{n} \phi_i = 1$$

A trainer's trust score is used as the weight factor for that trainer updates in the federated learning algorithm. The effect of that is two-fold. First, it enables validators to control the impact of said trainer's updates on the model based on its trust level. Secondly, it creates intrinsic competition in the system as any rise in one trainer's score effectively results in a relative decline in other trainers' scores. Trust scores are adjusted based on validation results. Upon receiving a trainer update, a validator adds its gradients to the latest model weights, and computes its validation score. Subsequently, it updates the respective trainer's trust score based on whether it leads to improvement or decline.

IV. Architecture

The purpose of this section is to present an overall description of a typical FLoBC[2] system such as ones we built to conduct our experiments. Said system was not designed with full deployment in mind, hence some details have been simplified. As such, FLoBC serves as a minimal proof-of-concept.

A. System Overview

FLoBC is a distributed system driven by two main actors: trainers and validators, the latter of which has more involved responsibilities. On a lower level, the system is composed of six main services, each providing a very specific functionality and has a strictly defined interface with other services and layers.

1) **Blockchain service**: Constitutes the main fabric of communication and data storage across all validator nodes. To ensure data consistency across all validators, this subsystem utilizes an elaborate schema for decentralized consensus, namely a variant of the Practical Byzantine Fault Tolerance (pBFT) family of consensus algorithms that are mainly based on voting and elections.
2) **Storage schema service**: Facilitates and enables structured access and modification of the local storage database on which the blockchain state is stored.
3) **Machine learning service**: Represents the machine learning layer on top of the blockchain. It is built as a service that handles the execution of machine learning transactions such as gradients sharing by interfacing with the local storage schema to reflect the transaction execution such as model creation and model aggregation.
4) **Training validation service**: Is a hybrid layer between the blockchain and the training layer. It is responsible for validating incoming updates and returning a verdict of whether a gradient transaction should be accepted or rejected.
5) **Reputation management service**: Handles the accounting aspects of the system, realizing a reward-punishment mechanism to ensure that only honest trainers get to stay in the network while Byzantine ones are dismissed to maintain the performance and quality of created models and ensure fairness.
6) **Model training service**: Performs model training including all needed intermediate transformations such as model flattening and rebuilding at the trainer side. This layer sends and receives flattened models and flattened gradients.
7) **Training flow management service**: Manages the communication between trainers and validators including model and gradients exchanging, along with synchronizing training iterations.

[2]The word FLoBC is sometimes used to refer to a system spawned through the framework rather than the framework itself.

B. Implementation Technology

FLoBC is built on top of an Exonum Blockchain that uses a variant of practical Byzantine Fault Tolerance (pBFT) consensus algorithm. The primary reason behind this choice is the voting-based consensus of Exonum being much more lightweight than the typical cryptographic Proof-of-Work (PoW) consensus [19]. This lightweight consensus becomes even more crucial when the already computationally heavy nature of model training is taken into consideration. The majority of system services are implemented using the Rust programming language while lightweight clients are implemented using JavaScript. The training and validation are carried out using easily pluggable Python scripts.

C. System Views

There are multiple ways to view FLoBC depending on the scope level, the most important of which are presented in the following.

1) Actor-level View: This view is the most high-level depiction of the system in terms of its main acting entities and how they are connected.

In our system, we have two types of actors: trainer and validator. While trainers constitute the main work labor in the system by performing gradient calculations using their data, validators are even heavier in composition as they are the ones undertaking elections to achieve consensus and validation to ensure model quality.

One important property of our architecture is that trainers are not fully connected to the validators while validators are fully interconnected. It is worth noting that while the roles (i.e., trainer or validator) are not fixed or dedicated, a node is not allowed to perform both roles simultaneously for the same subnetwork, yet a node can be a validator on a subnetwork while being a trainer for another. Further, each model subnetwork requires at least one validator to function.

2) Modular & Service-level Views: The modular view in Figure 1 gives a somewhat closer look on a validator's insides, showing the data flow from each high-level module to the others. It first shows that trainers communicate with validators over HTTP whose endpoint then passes the transactions to the validation module that uses the validation dataset to validate the model-update transactions in order to ultimately decide whether to accept of reject the update onto the blockchain. Validators also expose a Wire API for answering queries about the blockchain state (e.g., what's the latest model version?), which is helpful for system diagnosis/monitoring. On the other hand, the service-level view emphasizes the separation of concerns/responsibilities across the system into multiple services illustrated in Figure 2. The storage schema service is central to all services in the validator side as it represents the main interface for making persistent changes. The ML service is responsible for consolidating trainer updates and handling the trainer flow (e.g., synchronization) while the validation service is responsible for assessing trainer work and providing the results to the reputation service which can adjust trust scores accordingly. The Wire APIs on the sides represent auxiliary querying interfaces for miscellaneous purposes (e.g., for getting the latest released model version) besides the main blockchain interfaces used for sending transactions.

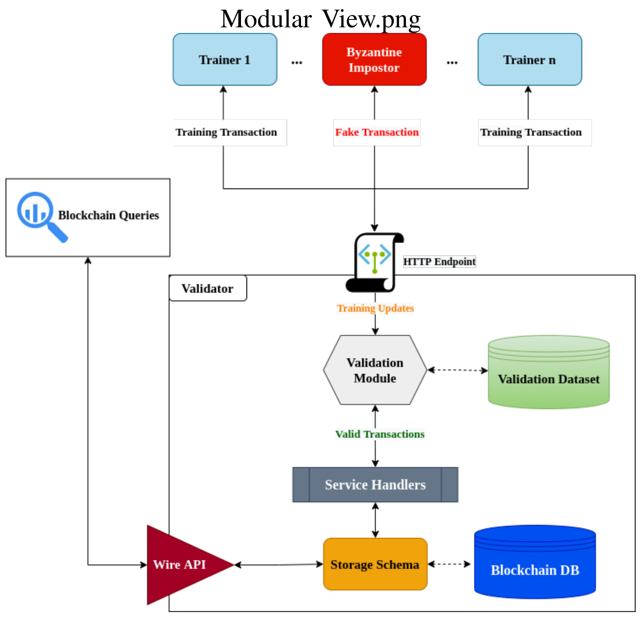


Fig. 1. Modular View

V. Experiments

In this section, we describe the experiments we have carried out using FLoBC. All the experiments carried out in this section were training and validating a convolutional neural network (CNN) predicting handwritten MNIST digits on the regular MNIST dataset with images of size 28×28 pixels. Each trainer in the system uses a random 30% sample of the MNIST training dataset. The metric used to evaluate training in these experiments is accuracy, but the system will work equivalently using any machine learning model metric. The metric is easily pluggable in the Python machine learning validation script.

Benchmark: Decentralized vs. Centralized Performance

In this experiment, we compare the performance of the centralized model with that of the decentralized model utilizing the best performing configuration (as demonstrated later by experiment 1) comprising 7 trainers and 3 validators with an active reward-penalty policy. The centralized model is the golden benchmark of our system. The aim of this experiment is to test whether the added benefits of decentralization would result in a significant cost of model quality.

Setup: In this experiment, the centralized model uses the whole data available in the MNIST training data-set. Since in the best performing decentralized configuration there are 7 trainers affecting the model each iteration, in our centralized benchmark, each training iteration is 7 epochs. The training lasts for 30 iterations for both the centralized and decentralized runs.

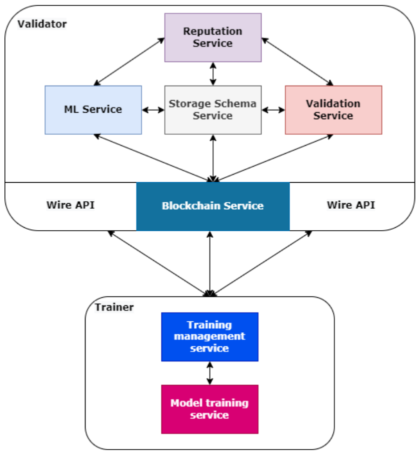


Fig. 2. Service-level View

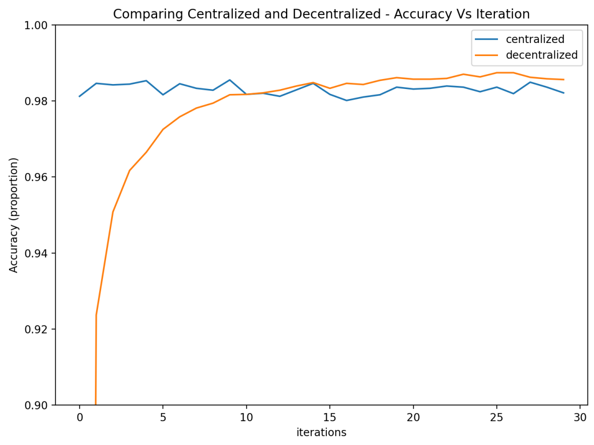


Fig. 3. Accuracy against iterations for the centralized and decentralized runs

Results and Discussion: As it can be seen in Figure 3, the centralized model had a higher accuracy than the decentralized alternative for 10 iterations, after which the decentralized model trained by 7 trainers and validated by 3 validators marginally outperformed the centralized model. The difference in accuracy is below 0.5%. This experiment shows that the added benefits of decentralization and privacy preserving training do not degrade the model performance. Note that this is not meant to demonstrate that the decentralized version is superior in performance to its centralized counterpart. Rather, it shows that the two are closely comparable in that regard.

Experiment 1: Trainers-to-Validators Ratio

In this experiment, we have maximum N participating nodes. Each run we split N into different numbers of trainers and validators. Through this experiment we want to determine the best trainers-to-validators split that makes use of the most computing recources and achieves the best model performance while still relying on multiple validators to generate a more trustworthy model. This experiment will also show the speed of convergence to the highest accuracy by pointing out the iteration at which the highest accuracy was achieved. There is a trade-off between the number of trainers and validators that this experiment aims to balance out. Note that increasing the number of trainers with good quality data should increase the model performance. Greedily, we would want all the nodes in the system to be trainers, however, this has a major drawback. Relying on a single validator means that the validator is a trusted entity whereas relying on multiple validators increases the trustworthiness of the model because of consensus. However, we want to refrain from having too many validators because this wastes computing resources that could have been utilized in training and adds on unnecessary communication costs. A trainer shares its updates with one validator that validates those updates then shares them with the rest of the validators for further validation and finally consensus. This being said, as we increase the number of the validators, we increase the communication cost associated with voting consensus. For these reasons, for the highest efficiency and the best performance, it is important to determine the best trainers-to-validators ratio that achieves the best model quality without compromising decentralization.

Setup: In this experiment, the N number of agents in the system is set to 10. For instance, with N = 10, one system run has 9 trainers and 1 validator while another run has 8 trainers and 2 validators, etc. Each run, we change the number of trainers and validators to try all their combinations that add up to 10 agents. Each run lasts for 30 iterations. We record the iteration and its accuracy for each run of the system. The new model version creation waits for all the trainers to share their updates with the validators since it is important in this experiment that all trainers submit their work for validation each model update iteration since we are studying the effect of the split of the validators and trainers on the model performance.

Results and Discussion: As it can be seen from Figure 4, the best configuration for this model is having 3 validators and 7 trainers. Using this configuration, the maximum accuracy of 0.9874 was reached at iteration 26. Note that the least accuracy was that for the single trainer configuration since much less computing resources and data have been dedicated for training.

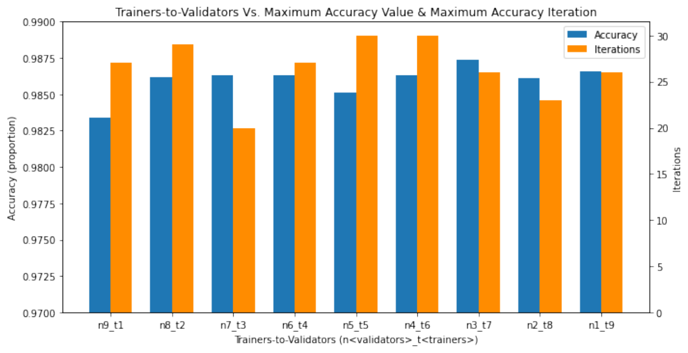


Fig. 4. Maximum accuracy and its iteration index for different splits of trainers and validators

Experiment 2: Reward-Penalty Policy

The purpose of this experiment is to evaluate the usefulness of computing a score for each trainer, where a trainer's score affects the extent to which this trainer's updates affect the overall model.

Setup: This experiment was performed using processes representing 6 trainers (indexed from 0 to 5) and 3 validators. Training ran for 30 iterations. The BSP synchoronization scheme was employed with a sufficiently large period to allow all trainers to submit updates every synchronization period.

The updates of each of the trainers were offset using an approximately normal noise. The mean of the noise is 0 and the standard deviation of the noise is linearly proportional to the index of the trainer, with proportionality constant k = 0.0545. Thus, the updates of trainer 0 received zero noise, and the updates of trainer 5 were significantly offset by the noise.

For purposes of this experiment, the minimum acceptance threshold for updates was reduced substantially to allow low-performing updates to affect the model generation and be affected by scoring. The control group had scoring disabled; that is, the scores of all 6 trainers were constant and equal throughout the training process. The scoring group allowed scoring for each trainer to change and to adapt to the relative qualities of the provided updates by each trainer.

Results and Discussion: During the operation of the scoring group, the scores of each trainer were updated after every synchronization period. A sample of the scores of all trainers over the 30 training iterations is displayed in Table I. We note that, shortly after training started, the validators realized that most trainers were providing low-quality updates, and thus, the scores of four trainers dropped to zero almost immediately. Over the remaining rounds, the score of trainer 0, whose updates were not affected by added noise, steadily increased against the score of the second-best trainer, trainer 1, whose updates were affected by a minimal amount of noise. Hence, we conclude that scoring enables the validators to accurately estimate the relative performance of each of the trainers.

For the two groups, after each training round, the current model accuracy is computed. The accuracy performance of the two groups is shown in Figure 5. We note that trainer scoring

TABLE I
SCORES OF ALL TRAINERS ACROSS TRAINING ROUNDS

Round	0	5	10	15	20	25	30
Trainer 0	0.16667	0.764622	0.823812	0.866141	0.907603	0.964613	1
Trainer 1	0.16667	0.235378	0.176188	0.133859	0.092397	0.035387	0
Trainer 2	0.16667	0	0	0	0	0	0
Trainer 3	0.16667	0	0	0	0	0	0
Trainer 4	0.16667	0	0	0	0	0	0
Trainer 5	0.16667	0	0	0	0	0	0

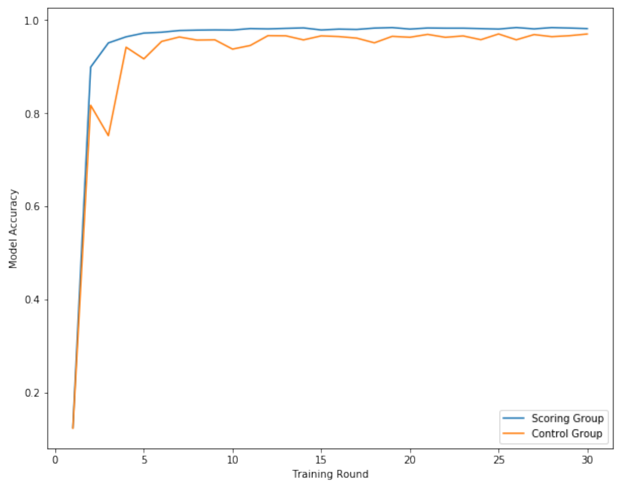


Fig. 5. Accuracy performance of scoring group vs control (uniform) group over 30 training rounds

improved the performance, stability, and convergence speed of the model, as only the updates from high-performing trainers were taken into consideration and contributed to improving the model.

Experiment 3: Synchronization Schemes

In this experiment, we compare some of the common synchronization schemes used in parallel computing after adapting our own variations of them. The schemes implemented in FLoBC and those tested in the experiment are Bulk Synchronous Parallel (BSP), Stale Synchronous Parallel (SSP), and Barrierless Asynchronous Parallel (BAP), as previously described in the methodology. The purpose of the experiment is to compare the different schemes in terms of two main metrics: model growth relative to the number of training rounds, and the average time per training round. The former indicates convergence while the latter indicates progression speed.

Setup: For the experiment setup, a system of three validators and six trainers is used in all runs with a uniform trust policy; that is, all trainers have the same trust scores. For both BSP and SSP, the same base synchronization barrier period is used. Different trainers are running at different paces to emulate real-world computational differences, with the synchronization barrier period set to be longer than the time most (but not all) trainers will take to finish one training job. To measure the model growth metric, the system will run for a fixed N=30 training rounds, comparing how long each scheme takes to converge on the highest model accuracy while also taking the three highest accuracies into consideration. As for the average training iteration delay, the system will run for a fixed period of 20 minutes. For BAP, we use two configurations: one with a slack ratio threshold of 0% (fully relaxed), and another with 40%. That is, in the first one, a sync barrier is thrown when all trainers have submitted at least one update in the current training round while the second one is contingent on only 60% of trainers having done so.

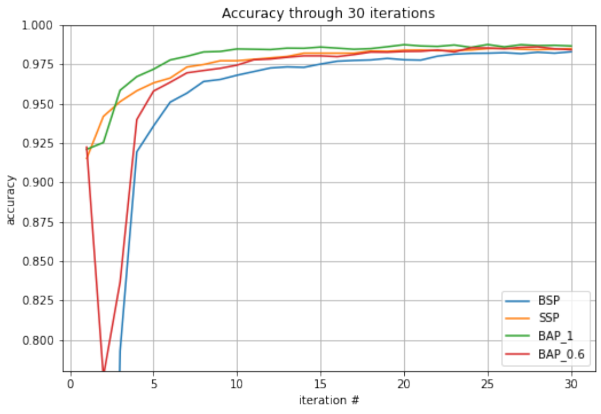


Fig. 6. Accuracy performance for 4 different schemes across 30 training iterations

Results and Discussion: As depicted in Figure 6, results seemingly meet their theoretical expectations, confirming the trade-offs for the three different schemes. BSP has the lowest performance as it is the most strict in terms of synchronization, leading to lower utilization of training power. However, it is quite stable as it offers more predictable limits on speed of progression (at a fixed period). As for the two BAP runs[3], BAP_1 with a majority ratio of 100% was expected to perform best since it has the best utilization of trainer work, beating its BAP_0.6 counterpart, which confirms the expected effect of decreasing the majority ratio in our variant of BAP. On the other hand, SSP appeared to strike a balance between BAP and BSP. This is expected since SSP is regarded as a compromise between strict synchronization (BSP) and full relaxation (BAP).

Looking at Figure 7, fully relaxed BAP performed the least number of iterations (i.e., training rounds) by far due to its re-

[3]Labelled as BAP_ <majority ratio>

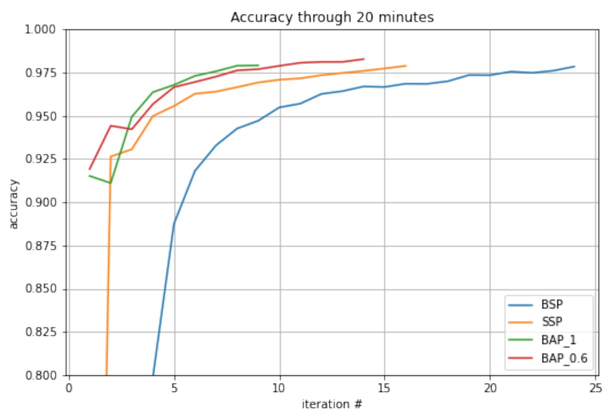


Fig. 7. Accuracy performance for 4 different schemes across 20 minutes of training time

laxed sync barriers. Naturally, partially relaxed BAP managed to perform significantly more rounds, eventually leading it to score the highest accuracy of all schemes. On the other hand, BSP had the furthest progression, managing to score very close to the top accuracy due to its faster pace. SSP performed fewer training iterations than BSP since it applies more relaxation on the sync barrier. However, SSP still outperforms BSP in terms of accuracy due to its better utilization of the training power. By and large, each one of these schemes suits certain system configurations (and purposes), depending on several factors, but the primary deciders are trainer pace and quality.

VI. Conclusion

In summary, FLoBC is a proof-of-concept that is intended to show that its components can come together to form a coherent system with the desired levels of generality, decentralization, efficiency, privacy, and Byzantine fault-tolerance. Through our benchmark experiments, our decentralized system has shown itself to be a viable match to pure centralized training, marginally outperforming the centralized benchmark by a factor of 0.5%. Using toy systems spawned by FLoBC, we first showed that there is a quasi-optimal balance to strike on trainer-to-validator partitioning, empirically determining that a 7:3 trainer-to-validator ratio works best. Secondly, we demonstrated that even a simple reward-penalty policy can have a notable positive effect on the quality of produced models. Finally, we compared three major synchronization schemes (BSP, SSP, & BAP), highlighting and contrasting their different trade-offs.

VII. Future Work

Despite the SGD-compatibility of a model being an assumption embedded in the system, FLoBC can be feasibly extended to handle other computational models. Further improvements can be applied to synchronization schemes by allowing more adaptability in the sync periods based on the expected time that most trainers take to submit their updates in one round. Furthermore, to ensure a higher level of data privacy, a layer of differential privacy could be incorporated into transaction communication to limit the extent to which gradients can be inverted to extract trainer data.

References

[1] Alekh Agarwal, Olivier Chapelle, Miroslav Dudík, and John Langford. A reliable effective terascale linear learning system. *The Journal of Machine Learning Research*, 15(1):1111–1133, 2014.

[2] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. *arXiv preprint arXiv:1606.04474*, 2016.

[3] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[4] Amin Fadaeddini, Babak Majidi, and Mohammad Eshghi. Secure decentralized peer-to-peer training of deep neural networks based on distributed ledger technology. *The Journal of Supercomputing*, pages 1–15, 2020.

[5] Andrew Gibiansky. Bringing hpc techniques to deep learning. *Baidu Research, Tech. Rep.*, 2017.

[6] Otkrist Gupta and Ramesh Raskar. Distributed learning of deep neural network over multiple agents. *Journal of Network and Computer Applications*, 116:1–8, 2018.

[7] István Hegedűs, Gábor Danner, and Márk Jelasity. Gossip learning as a decentralized alternative to federated learning. In *IFIP International Conference on Distributed Applications and Interoperable Systems*, pages 74–90. Springer, 2019.

[8] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Keith Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977*, 2019.

[9] Hyesung Kim, Jihong Park, Mehdi Bennis, and Seong-Lyun Kim. Blockchained on-device federated learning. *IEEE Communications Letters*, 24(6):1279–1283, 2019.

[10] Anusha Lalitha, Osman Cihan Kilinc, Tara Javidi, and Farinaz Koushanfar. Peer-to-peer federated learning on graphs. *arXiv preprint arXiv:1901.11173*, 2019.

[11] Hao Li, Asim Kadav, Erik Kruus, and Cristian Ungureanu. Malt: distributed data-parallelism for existing ml applications. In *Proceedings of the Tenth European Conference on Computer Systems*, pages 1–16, 2015.

[12] Yuzheng Li, Chuan Chen, Nan Liu, Huawei Huang, Zibin Zheng, and Qiang Yan. A blockchain-based decentralized federated learning framework with committee consensus. *IEEE Network*, 2020.

[13] Ismael Martinez, Sreya Francis, and Abdelhakim Senhaji Hafid. Record and reward federated learning contributions with blockchain. In *2019 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC)*, pages 50–57. IEEE, 2019.

[14] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*, pages 1273–1282. PMLR, 2017.

[15] Shiva Raj Pokhrel and Jinho Choi. A decentralized federated learning approach for connected autonomous vehicles. In *2020 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*, pages 1–6. IEEE, 2020.

[16] Shiva Raj Pokhrel and Jinho Choi. Federated learning with blockchain for autonomous vehicles: Analysis and design challenges. *IEEE Transactions on Communications*, 68(8):4734–4746, 2020.

[17] Chandra Thapa, Mahawaga Arachchige Pathum Chamikara, and Seyit Camtepe. Splitfed: When federated learning meets split learning. *arXiv preprint arXiv:2004.12088*, 2020.

[18] Joost Verbraeken, Matthijs Wolting, Jonathan Katzy, Jeroen Kloppenburg, Tim Verbelen, and Jan S. Rellermeyer. A survey on distributed machine learning, 2019. cite arxiv:1912.09789.

[19] Yury Yanovich, Ivan Ivashchenko, Alex Ostrovsky, Aleksandr Shevchenko, and Aleksei Sidorov. Exonum: Byzantine fault tolerant protocol for blockchains. *bitfury. com*, pages 1–36, 2018.