Correct Timings and Inspection of States for Federated Learning Simulations

Marko Putnik[1]

Supervisor(s): Dr. Jérémie Decouchant[1]**, Bart Cox**[1]

[1]**EEMCS, Delft University of Technology, The Netherlands**

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 23, 2024

Name of the student: Marko Putnik
Final project course: CSE3000 Research Project
Thesis committee: Dr. Jérémie Decouchant, Bart Cox, Dr. Qing Wang

An electronic version of this thesis is available at http://repository.tudelft.nl/.

Abstract

Federated Learning is a machine learning paradigm where the computational load for training the model on the server is distributed amongst a pool of clients who only exchange model parameters with the server. Simulation environments try to accurately model all the intricacies of such a system. However, current simulators do not properly impose the concept of simulation time, leading to global model inaccuracies and difficulties of replicating reruns of the simulation, which is most prominent in the asynchronous scenarios. To this purpose, we propose a *discrete-event simulator* for the central server asynchronous case which timestamps all the events in the system prior to execution, reducing variability in client model updates on the server. We also introduce a log-structure used to keep states of the simulation, making client inspection possible based on time. We evaluate the proposed *discrete-event simulator* on the baseline simulator of *Flower*, reducing standard deviation amongst server model updates for 31.5% and improving accuracy with heterogeneous clients in the MNIST case for 3.3% on average.

1 Introduction

Federated Learning, exemplified with the *FedAvg* algorithm [1], is a machine learning approach where distributed devices collectively train a central global model. In this approach a central server each round selects a subset of clients that are used for independent training of the model with their own data, after which the server aggregates the model by combining the obtained model parameters from clients. The characteristics of such an environment are client resource and data heterogeneity, which exist due to differences of client computational resources and the non-IID distribution of data [2; 3].

The landscape of Federated Learning has expanded rapidly from the initial *FedAvg* algorithm to encompass a variety of different network topologies, expanding to multi-server [4; 5] and fully decentralized topologies [6], and communication primitives, where asynchronous algorithms [7], such as *FedAsync* [8], have been introduced. In order to quickly test and verify new ideas about FL algorithms, researchers resort to simulation environments due to costly effects of real-life deployments of Federated Learning.

Due to its relative recent appearance in the research field, Federated Learning simulation environments, exemplified with the popular framework *Flower* [9], are still very new and do not have all the features of an ideal distributed system simulation environment, such as of *discrete-event simulators* [10]. Popular distributed systems discrete-event simulators, such as the *ns3* [11] network simulator and the *PeerSim* [12] peer to peer simulator, provide tools for correct event execution based on simulation time and tools for tracking system state with simulation time.

Because of its heterogeneous nature, simulation environments catered for Federated Learning need to fulfill similar properties as the previously cited discrete-event simulators. Without those properties, especially the property of the global simulation time, Federated Learning simulations can as a consequence have non-accurate global states and showcase non-reproducibility between simulated runs of the same experiment. This is especially relevant in the asynchronous scenario where the global model is updated as soon as the client request arrives. For example, clients that update the global model more often on average, due to variability in hardware resources and OS scheduling and without proper simulation time, would bias the global model towards their data distribution.

To this purpose, we will try to address and solve for the following research questions:

- **RQ1:** How to ensure correct timings for a simulated FL system?
- **RQ2:** How can we inspect the results of the system for any point in time?

The main contributions can be summarized as follows:

1. We propose a formal model for a *discrete-event simulator*, based on the *decentralized-learning-simulator* [13], in the central server asynchronous Federated Learning scenario by defining all the events in such a system together with a network and a computation model used in the simulator. The final result of running the simulator is a directed acyclic graph of all the tasks in the system: from local client training to server level aggregation, which is resolved by one or multiple worker processes.
2. We propose a log-structure that keeps track of all the states of the simulator when worker processes are resolving the DAG. The log-structure serves the purpose of inspecting system state at any point in time.
3. We extend the GitHub repository of the *decentralized-learning-simulator* [13] to implement the central server asynchronous FL scenario based on the formal definition we give.
4. We validate the extended framework by comparing it with the baseline framework of *Flower* [9] with the aggregation algorithm of *FedAsync* [8] on global system accuracy and reduction in standard deviation between the reruns of the experiment.

The rest of the paper is organized in the following manner. Section 2 will give an overview on the required background knowledge and Section 3 will show the state of the art papers on the topic. Following that, Section 4 will go more in-depth into the problem that we are tackling, from which Section 5 will continue and introduce solutions to the research questions **RQ1** and **RQ2**. Section 6 describes the experimental setup and the results of the experiments, whose results are further reasoned about in Section 7. Continuing from there, Section 8 discusses the ethical principles of the conducted research and finally Section 9 wraps up the paper with the conclusion and gives insight into future research directions.

2 Background

The following section will cover the important background information needed to understand the paper, starting with the concept of Federated Learning, then proceeding to Federated Learning simulations and ending with an outlook on *discrete-event simulations*.

2.1 Federated Learning

In the paper of *FedAvg* [1] the following model for Federated Learning is presented, where K is the number of clients, t the global round number, P_k the set of data points for client k and $f_i(w) = l(x_i, y_i; w)$ the loss of the prediction on example (x_i, y_i) on model parameters w. The objective for each client is to compute the minimum loss for parameters w, which is described with the following equation:

$$w_k^t \leftarrow \arg\min_w F_k(w) = \frac{1}{n_k} \sum_{i \in P_k} f_i(w)$$

The goal of Federated Learning is then to minimize the loss on all client model parameters by aggregating the values into a global model:

$$w^{t+1} = \sum_{k=1}^{K} \frac{n_k}{n} w_k^t$$

In the synchronous approach, the server needs to wait for all the selected clients to respond with their new model parameters, thus faster clients are stuck waiting for the slow clients to finish [2]. Due to this, an asynchronous approach has been introduced, initially with the *FedAsync* [8] algorithm, which does not wait for the selected group of clients to finish before starting the aggregation process, rather the server aggregates the model parameters as they arrive one-by-one. The main objective can be described with the following equation:

$$w^{t+1} = (1 - \alpha) * w^t + \alpha * w_k^{t-\tau}$$

where w^t represents the global model before aggregation, $w_k^{t-\tau}$ represents the client's locally trained model on the global model $w^{t-\tau}$ and $\alpha \in (0, 1)$ represents the staleness factor which can be defined as:

$$\alpha = s_a(t - \tau) = \frac{1}{(t - \tau + 1)^a}$$

which is one of the options considered in the *FedAsync* [8] approach.

2.2 Federated Learning Simulations

State of the art simulation environments for Federated Learning do not resemble traditional simulation environments, such as *discrete-event simulators*, rather they serve the purpose of researchers quickly verifying new algorithms without having real-life deployments. This is demonstrated as most of the simulation environments for Federated and Decentralized Learning utilize a single-machine or a cluster environment [14].

Many existing frameworks, such as *Flower* [9], *Flute* [15] and *Pollen* [16], offer different Federated Learning algorithms already included in the framework and some of them, such as *Flower* [9], provide extensibility when it comes to the underlying Machine Learning platform used, providing the choice between frameworks of *PyTorch* [17] and *TensorFlow* [18].

From the computer architecture perspective, FL clients and servers are represented in lots of ways. One approach models each client and server instance as an individual process, as in *decentralize_py* [14], meanwhile other FL simulators, such as *FL_PyTorch* [19], model each client/server instance as an individual CPU thread from the thread pool inside the Python interpreter process. Other frameworks, such as *Flower* [9] and *Flute* [15], abstract the mapping of clients/servers to OS processes or CPU threads, thus leaving the task of mapping processes/threads to clients/servers to the underlying simulation engine used within the simulator.

2.3 Discrete-Event Simulations

In the sphere of Federated Learning simulations, there has not been much research done in the development of a *discrete-event simulator*. Only the *decentralized-learning-simulator* [13] implements a *discrete-event simulator* for the Decentralized Learning scenarios and a central server synchronous Federated Learning case. Discrete-event simulator is a simulation model in which the model state is defined as a sequence of events and where the simulation clock jumps from event time to event time, rather than the clock "ticking" at regular intervals. In this way the current simulation time is updated to the time of the next event and changes in the system that occur with the next event state are executed [20].

In order to properly model the operation of the system assumptions need to be made about the underlying components of the system. A popular approach is to *stochastically* model the underlying components by assuming a probability distribution based on the collected statistical data of each component [21]. For example, famous *discrete-event simulators*, such as network simulators *ns3* [11] and *PeerSim* [12], create a *stochastic* model for the underlying resources of the system, such as: CPU compuation rate, bandwidth and network latency [21].

3 Related Work

The following section will cover the state of the art solutions, starting from the description of significant Federated Learning simulators and finally the *decentralized-learning-simulator* [13] we use to implement the asynchronous central server scenario.

3.1 Federated Learning Simulators

The most widely used simulator for Federated Learning is *Flower* [9]. A significant downside of *Flower* is that it is a continuous simulator, without a proper concept of global simulation time. This means that it is vulnerable to global state inaccuracies between reruns of the same experiment, which is especially relevant in the asynchronous scenario. Another limitation of *Flower* is that it does not model heterogeneous clients efficiently within the simulation, which

is where the simulator of *Protea* [22] improves the baseline framework by increasing the performance of the simulator by properly modelling for heterogeneous clients.

Another downside of *Flower* and the state of the art Federated Learning simulators is in not properly modelling the network elements inside of the simulation. This is where the simulator of *ns3-fl* [23] introduces a network model into the baseline simulator by integrating the *ns3* [11] network simulator for properly modelling the latency of the communication. The authors of *ns3-fl* propose the following formula for modelling timing inside of the simulations:

$$L_i^r = t_d^r + t_c^r + t_u^r$$

where t_d^r represents the time it takes client i to establish a connection with the server and retrieve the full global model in round r, t_c^r represents the client local training time in round r, t_u^u represents the time it takes for the client to send the updated model to the server in round r.

The authors of *FEDL* [24] state that the value of t_c^r can be estimated based on the number of CPU cycles that the local iteration of model updating would take on the client process. On the other hand, authors of the simulator of *Pollen* [16] state that the client training time inside the GPU cannot be predicted based on dataset size and hardware specifications as factors such as OS scheduling and memory bandwidth introduce variability. Due to the discrepancy of conclusions from the previously cited papers, we will assume that the value t_c^r if tied to a CPU process is not predictable based on dataset size and hardware.

3.2 Discrete-Event Simulators

The only *discrete-event simulator* for Federated or Decentralized Learning is the *decentralized-learning-simulator* [13]. The simulator can be found in the GitHub repository and is not yet backed by a research paper. The simulator implements a *discrete-event simulator* for a large pool of Decentralized Learning scenarios and the central server synchronous Federated Learning scenario. It works by time-stamping all the events in the system through the execution of the *discrete-event simulator* which then builds a directed acyclic graph of all the tasks in the system which is then given to one or more worker processes to be resolved. For modelling computation delay occurring with local client training, the simulator increases the event time based on the following formula:

$$\text{AUGMENTATION\_FACTOR\_SIM} \times \text{local\_steps} \times \text{batch\_size} \times \left(\frac{\text{simulated\_speed}}{1000}\right)$$

and for modelling the network it implements a *bandwidth scheduler* that keeps track of transfer events within the simulation based on which it increases simulation time.

Another noteworthy work is the paper of *BlockSim* [21] which is a *discrete-event simulator* that simulates a blockchain. The simulator proposes a network model based on the model for latency and throughput. When it comes to latency, the authors gather statistical data about ping traces from different locations and for throughput, the authors gather statistical data by gathering bandwidth measurement data on two TCP endpoints in different geographical regions, after which they approximate the probability distribution over the data for latency and throughput.

4 Problem Description

This section will cover the main problems that we are trying to address and solve for when it comes to correct timings and inspecting simulation state correctly.

4.1 Formal Problem Description

A properly defined simulation environment needs to be able to reproduce the performance of a system and its progress over time [21]. This is not the case in majority of the Federated Learning simulation frameworks, exemplified with the framework of *Flower*, primarily because they lack the property of the global simulation time. This means that the simulations are susceptible to inaccurate results due to the heterogeneous data distribution between the clients and the client's local training variability.

The observed variability between clients in local client training time, t_c, is mostly influenced by the difference in the underlying hardware resources of simulated clients as some simulated clients can be run on the GPU and others can be run on the CPU. Further, each client state can vary with the operating system scheduling of internal processes or threads and the memory layout of the processes [16].

Having correct timings of events and a global simulation time is especially relevant in the asynchronous case, as the global model state is updated immediately as the client request arrives. An issue in simulations without a global simulation time is that faster clients would dominate the aggregation of the model, since they would have more updates on the global model compared to slower clients, thus having a global model show bias for the data distributed in the faster clients.

Another issue of simulations without a concept of global simulation time is the possibility of non-reproducibility of the simulated run. This is mostly prevalent in the asynchronous scenario, with simulations that deploy homogeneous clients, such as clients having the same batch size and the same underlying resources, as those clients are more dependent on the scheduling in the OS. The variability in client local computation time between the simulated runs means that the order of updating the global model on the server has the possibility of differing between multiple runs of the same simulation experiment.

Moreover, having a proper timing model inside of the simulation allows for inspection of the system at any point in time, which can foster debugging the simulated run.

5 Time and State Tracking System Design

The following section will introduce a formal model for the *discrete-event simulator* in the central server asynchronous case which imposes a single global simulation time. The section also showcases the log-structure used to inspect the state of the simulator at any point in time. Both models will be our proposed answers to problems defined in the previous section.

5.1 Timing Model

In order to resolve the research question **RQ1**, a formal model for the *discrete-event simulator* will be introduced

that imposes a global simulation time, which is used to timestamp all the events in the system by developing a directed acyclic graph of tasks within the system which is then resolved by one or multiple worker processes. The idea for such a *discrete-event simulator* stems from the GitHub repository of the *decentralized-learning-simulator* [13]. The *decentralized-learning-simulator* does not implement, thus neither formally models, the central server asynchronous case, which is what our contribution will be. Further, we will introduce a different approach to compuational and network times derived from the idea found in the *BlockSim* [21] simulator.

Components of the simulator: The main components of the simulator are: the global simulation clock, list of simulation events, random variables for *stochastic* modelling of network and computational resources and the directed acyclic graph of main tasks in the central server asynchronous FL scenario.

Global simulation clock: The simulation keeps track of a single global simulation time which is initialized to 0 at the start of the simulation. The clock moves from one event time to the next event time as the simulation evolves; e.g., the global simulation time is updated to be equal to the event time of the next to be executed event, which is updated as the event is "popped" from the event list.

Events list: The simulation maintains a list of events to be executed. Each event in the simulation is defined with the global simulation time at which the event occurs and its accompanied type. The main event types are the following:

- Initialization of client model parameters
- Start of local model client training
- End of local model client training
- Start of transfer of model parameters
- End of transfer of model parameters
- Server aggregation of updated client model parameters

The events list is organized as a priority queue sorted by event time i.e., global simulation time of event occurrence.

Random variables: The simulator will construct a *stochastic* model of the underlying resources: network latency, network throughput and client-side training time, by developing a probability distribution based on statistical data of each resource within the FL scenario. To achieve this we use the *Kolmogorov-Smirnov test* [25] to find the best probability distribution fitted over the data. For the network models we have considered the following distributions: normal, log-normal and Weibull, as some were considered a good fit over network data [26]. The presented models are the following:

- **Latency:** For the communication delay between clients and the server, the data is taken from the AWS Latency Monitoring service [27], whose small sample can be found in Table 1. Based on the data from the AWS service and the results of the *Kolmogorov-Smirnov test* [25] we assume a Gaussian distribution:

  $$X \sim \mathcal{N}\left(\mu, \sigma^2\right)$$

  with $\mu = 154.385$ and $\sigma = 93.297$.

Latency(ms)	Cape Town	Hong Kong	Sydney	Frankfurt
Cape Town	8.1	360.62	410.43	155.04
Hong Kong	365.1	1.27	129.27	219.68
Sydney	412.56	129.76	2.57	274.79
Frankfurt	162.01	217.15	275.78	3.54

Table 1: Sample of Latencies from the AWS Latency Monitoring Service

- **Throughput:** The throughput is calculated by measuring the bandwidth over two TCP endpoints, one from the client machine located in the Netherlands and the other from the server machine located in Finland, by using the *iPerf3* [28] tool to capture throughput. Based on the gathered data and the results of the *Kolmogorov-Smirnov test* [25] we assume a Weibull distribution:

  $$Y \sim f(x; \lambda, k, loc)$$

  with $\lambda = 114.7387$, $k = 29.105$ and an additional shifting parameter $loc = -76.1569$.

- **Client training:** Client training times are calculated by running multiple full asynchronous FL scenarios with varying batch sizes; 12, 64 and 100, and hardware resources; CPU and GPU, amongst the clients. Based on the results of the *Kolmogorov-Smirnov test* [25] we assume a Log-normal distribution:

  $$Z \sim \text{ Lognormal}\left(\mu, \sigma^2\right)$$

  with $\mu = 0.165$ and $\sigma = 0.8632$.

Directed Acyclic Graph: The DAG will be the outcome of running the *discrete-event simulator*. Nodes in the DAG will represent tasks in the central server asynchronous FL system which will be executed by one or more worker processes. Edges in the DAG will represent dependency constraints between the tasks in the FL scenario; e.g., a client local model update must occur before the server aggregates that model into the global model. Each task will include a list of input and output tasks, where the input list will form a set of dependency constraints, the tasks upon which the current task is dependent, and where the output list forms a set of tasks dependent on the current task. There will be three different types of tasks:

- **Initialization task:** This task will be the only source node in the DAG; i.e., the entry task before no other task can be executed.
- **Aggregate task:** The task represents server level aggregation of the trained client model at server age time t. Input tasks attached to the *aggregate* type of task are the previous *aggregate* task, *aggregate* task that happened at server age time $t-1$, and the *training* task which represents the task whose execution lead to the new client model update. When executing the *aggregate* task, the worker process will aggregate the model based on the approach of *FedAsync*.
- **Training task:** The task represents local model client training and has a single task in its input tasks list, which is the *initialize* or *aggregate* task. In its output list it also contains a single task, an *aggregate* task.

Protocol: The simulator behaves in the following way:

1. **Client initialization event** is the first event to occur and it updates the event list by appending *start transfer of model parameters* events for each client to the list who then update the event time as the current simulation time added with the communication delay, which is calculated as seen in Algorithm 1.

Algorithm 1 Communication Delay

$latency \leftarrow$ Sample X {sample only positive values}
$throughput \leftarrow$ Sample Y {sample only positive values}
$throughput\_bytes\_per\_sec \leftarrow \frac{throughput \times 1{,}000{,}000}{8}$
$transfer\_time \leftarrow \frac{\text{MODEL\_SIZE}}{throughput\_bytes\_per\_sec}$
$communication\_time \leftarrow latency + transfer\_time \times 1{,}000$
return $communication\_time$

2. **Event cycle** loop starts after the *initialization* event finishes and this part resembles the classic central server asynchronous FL scenario event loop, which contains the following events: client local training of the model, followed with sending the server new parameters, server aggregation of the parameters into a global model and transfer of the newly updated model to the client. Every client experiences the same event loop, which is visualized in Figure 1.

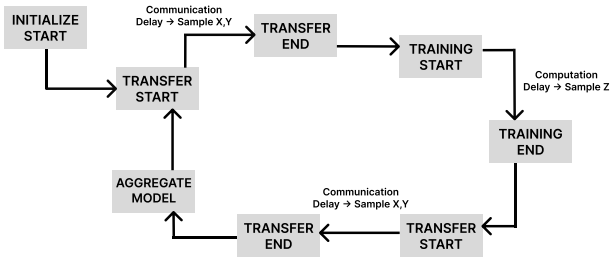

Figure 1: Simulation Event-Cycle for a Single Client-Server Pair

3. **Ending condition** stops the simulation and is represented as the *aggregation* event happening after a certain number of aggregations has occurred.

5.2 State Model

In order to resolve the research question **RQ2** we propose the following log-structure which will serve the purpose of gathering simulation traces which can be used to inspect the simulated run at any point in time.

The result of the *discrete-event simulator* will be a DAG of tasks. This DAG will be solved by one or many worker processes, obeying the precedence constraints within the DAG. In order to track system state accurately in the execution of the DAG, a log-structure L is introduced, which will form a mapping of each client to its list of states, where a single state represents the events in the system from the server sending the client the global model parameters, then to the client local training on the model and finally the server receiving the client's local model parameters.

Each state structure S_i^j, with client c_i and index of server-client communication j, will as its parameters contain the following:

- begin_timestamp: global simulation time when the client receives the server's model
- server parameters received at time $t_{begin\_timestamp}$
- server age received at time $t_{begin\_timestamp}$
- t_c: sampled global simulation time for client training
- client updated model parameters
- client set of gradients for E epochs:

$$\{\nabla\theta^{(e)}\}_{e=1}^{E}$$

- end_timestamp: global simulation time when the server receives the model from the client
- server model parameters at time $t_{end\_timestamp}$
- server age at time $t_{end\_timestamp}$

6 Experimental Setup and Results

The *discrete-event simulator* was implemented as an extension to the codebase of the *decentralized-learning-simulator* [13]. The original repository did not implement the asynchronous FL scenario, so it had to be extended for that scenario. For the baseline FL framework, *Flower* [9] was chosen because of its simplicity to run and configure. The framework of *Flower* was also extended as it does not implement the asynchronous FL scenario natively, for which we have gotten help from the GitHub repository of *flower_async* [29]. *PyTorch* [17] was chosen as the machine learning library in both frameworks due to it having more examples on GitHub compared to alternatives.

Datasets: The experiment is carried out on two image classification tasks: MNIST [30] and CIFAR-10 [31]. The MNIST dataset consists of 60000 training and 10000 testing 28x28 images of handwritten digits, consisting of a total of 10 different classes. The CIFAR-10 dataset consists of 60000 32x32 images of 10 different classes, representing airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships and trucks.

Architectures: We will consider a simple CNN architecture [32] consisting of 3 fully connected layers and 2 convolutional layers, the only difference being in the number of input channels for the first convolutional layer; 1 for the MNIST and 3 for the CIFAR-10 dataset.

Hardware: The machine running the experiments has a CPU AMD Ryzen 7 4800H with 8 cores and the GPU is NVIDIA GeForce GTX 1650 with 896 CUDA cores.

Hyperparameters: For MNIST we consider: the learning rate of 0.0005, number of epochs of 2 and momentum of 0.9. For CIFAR-10 we consider: the learning rate of 0.01, number of epochs of 3 and momentum of 0.9.

Sampling Strategy: In both experiments the clients will experience a non-IID data distribution. To this purpose, we have considered the approach of giving each client 2 classes for 80% of the training data with the rest of the training data being uniformly distributed over the remaining classes. The approach is visualized in Figure 2. The idea behind the approach came from trial-and-error experimentation, starting by giving each client 2 classes for 100% of the training data, which did not result in a convergence pattern on the MNIST dataset. By having 20% of the data be uniformly distributed over the rest of the classes, we saw a convergence pattern.

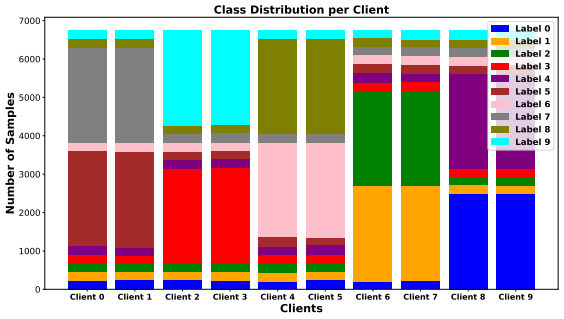

Figure 2: Non-IID Data Distribution over Clients with each Client Receiving 80% of Samples for 2 Labels and the Remaining 20% for the Remaining Labels for the CIFAR-10 Dataset

6.1 Experiments

Experiments will consider two different variations of clients: heterogeneous and homogeneous. In the experiment with the heterogeneous clients we will as one metric compare global model accuracy between the *discrete-event simulator*, written as *DES* in figures, and *Flower*. The second metric for heterogeneous clients is the standard deviation between the reruns on each framework, which serves the purpose of measuring variability between the frameworks. In the homogeneous case, we will only compare the standard deviation between reruns between the two frameworks. For comparing the global model accuracy in the asynchronous FL scenario, we will implement the aggregation approach of *FedAsync* with $a = 0.5$ whose formula can be seen in Section 2. The results will be averaged over different runs in order to reduce the variance and for different experiments different seeds will be used. Each experiment will consist of 10 clients, all running on the GPU.

Experiment 1: Heterogeneous Clients

The goal of the first experiment will be to capture the intricacies between 'faster' and 'slower' clients and their effects on the overall global state, specifically global model accuracy, and the amount of variation between the reruns of the experiment. Faster and slower clients will be modelled by being assigned different batch sizes, specifically half of the clients will be assigned the batch size of 10 and half of the clients will be assigned the batch size of 300. The approach with varying different batch sizes to introduce heterogeneity is found in the simulator of *Protea* [22]. The experiment consists of 8 worker processes and results are averaged on 3 different runs for each dataset-framework combination. For the *discrete-event simulator* we have used the seed of 2 in the MNIST case and the seed of 1 in the CIFAR-10 case.

Figures 3 and 4 showcase the results on MNIST.

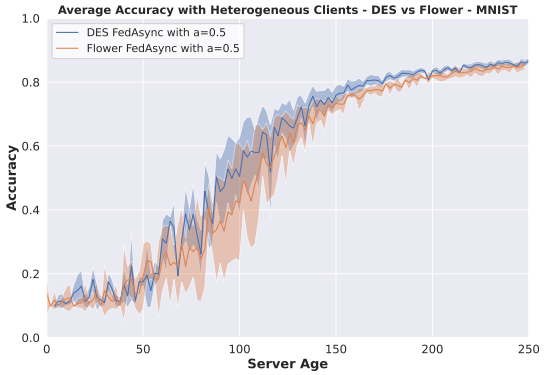

Figure 3: Average Global Accuracy with Heterogeneous Clients - DES vs Flower on MNIST

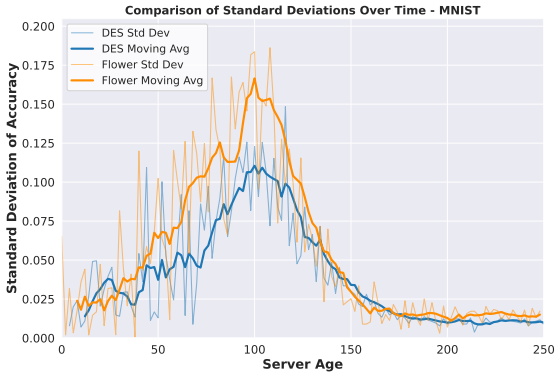

Figure 4: Standard Deviation Over Time Between Reruns with Heterogeneous Clients - DES vs Flower on MNIST

From Figure 3 we can notice a higher global model accuracy with the *discrete-event simulator* compared to the baseline *Flower*, especially in the interval range from the server age of around 60 to the server age of around 130. From the server age of 130 both frameworks converge to a similar accuracy value, but still the *discrete-event simulator* shows slightly better accuracy. When it comes to the standard deviation between the reruns, plotted in Figure 4, we can see similar trends but overall a decrease in standard deviation with the *discrete-event simulator*, especially visible around the median values, where the gap with the *Flower* is the largest. The mean value for standard deviation between the reruns for the *discrete-event simulator* is 0.039 and for *Flower* is 0.052.

Figures 5 and 6 showcase the results on CIFAR-10.

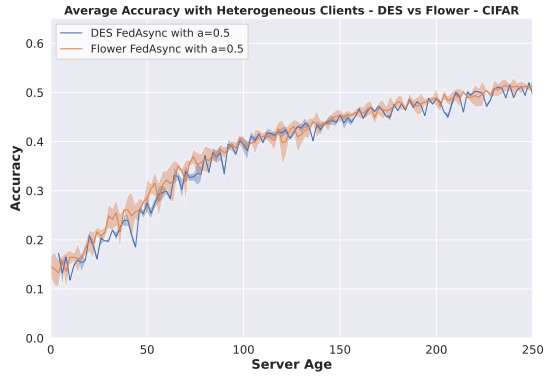

Figure 5: Average Global Accuracy with Heterogeneous Clients - DES vs Flower on CIFAR-10

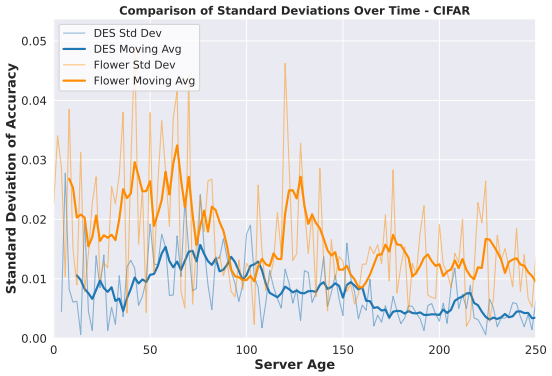

Figure 6: Standard Deviation Over Time Between Reruns with Heterogeneous Clients - DES vs Flower on CIFAR-10

From Figure 5 we notice similar accuracy trends for both frameworks, with the *discrete-event simulator* showing slightly worse accuracy, most noticeable in the initial phases. When it comes to the standard deviation, from Figure 6 we can notice that the *discrete-event simulator* shows reduced standard deviation between the reruns compared to *Flower*. The mean value for standard deviation between the reruns for the *discrete-event simulator* is 0.007 and is 0.016 for *Flower*.

Experiment 2: Homogeneous Clients

The goal of the second experiment is to capture the variability in the underlying simulators imposed by the internals of the OS under which the simulation is ran. In order to isolate the variable of OS scheduling and not have influence over it, clients will be homogeneous, having the same batch size of 64. The experiment consists of 4 worker processes and results are averaged on 4 different runs for each dataset-framework combination. For the *discrete-event simulator* we have used the seed of 4 in the MNIST case and the seed of 3 in the CIFAR-10 case.

Figures 7 and 8 and showcase the results of the experiment.

From Figure 7 we can see similar values for standard deviations on both frameworks, with the *discrete-event simulator* showing slightly smaller standard deviation. This is confirmed given that the mean value for standard deviation between the reruns for the *discrete-event simulator* is 0.012 and is 0.015 for *Flower*.

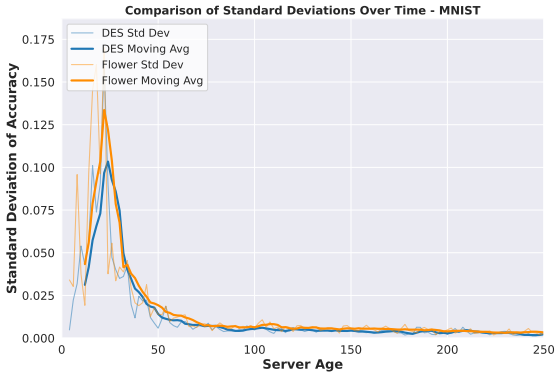

Figure 7: Standard Deviation Over Time Between Reruns with Homogeneous Clients - DES vs Flower on MNIST

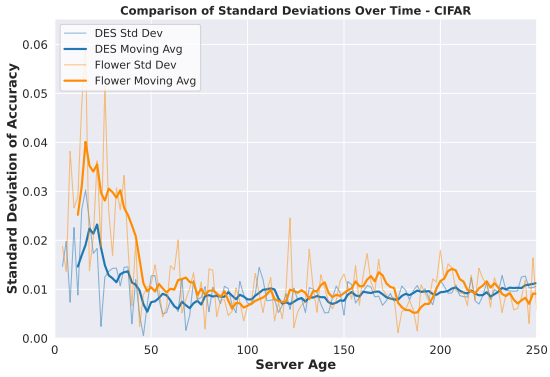

Figure 8: Standard Deviation Over Time Between Reruns with Homogeneous Clients - DES vs Flower on CIFAR-10

From Figure 8 we see that the *discrete-event simulator* has a smaller standard deviation between the reruns compared to *Flower*, which is confirmed given that the mean value for standard deviation for the *discrete-event simulator* is 0.009 and 0.012 for *Flower*.

7 Discussion

This section will take a closer look on what the results entail and will showcase the log-structure, defined in Subsection 5.2, used to inspect client progression over time.

7.1 Experimental Results

Global Model Accuracy: We have compared the two frameworks on global model accuracy in the experiment with heterogeneous clients. Our hypothesis was that with the continuous simulator, such as *Flower*, faster clients would dominate the aggregation and thus bias the global

model towards their data distribution in a non-IID scenario. Our proposed mitigation, the *discrete-event simulator*, only improves the global model accuracy in the MNIST case. One possible reason could be that our proposed sampling strategy does not capture the non-IIDness equally in both datasets. In MNIST, the feature space is much smaller and more homogeneous than in CIFAR-10, and with our non-IID sampling strategy faster clients might have been assigned more non-overlapping features compared to slower clients in MNIST than in CIFAR-10.

In general, our approach with comparing two frameworks on global model accuracy might have brought conflicting results from both datasets, but the idea that strict ordering of events prior to execution increases accuracy in heterogeneous scenarios is sound, as we have started testing the two frameworks by assigning clients the batch size of 10 and 200, which has yielded similar results on MNIST. We have also used a static learning rate which might not capture intricacies of CIFAR-10 fully, possibly explaining the observed results.

Standard Deviation Between Reruns: We see reduced standard deviation with the *discrete-event simulator* for all experiments and datasets. This is the expected result as our hypothesis was that the *discrete-event simulator* would reduce variability between reruns which is found in the continuous time simulator of *Flower*. The reduced standard deviation comes with imposing deterministic client update times with the *discrete-event simulator* which leads to slightly more deterministic learning process results. Even with the ordering of events, the *discrete-event simulator* is vulnerable to the intrinsic non-determinism of the training process itself, which is especially visible in MNIST cases, as in that dataset the initial convergence rate is high, thus even slight differences between reruns would lead to faster divergence in accuracies, which is observed in Figures 4 and 7.

7.2 Inspecting Client State

From the log-structure we can visualize the progression of clients and the server, exemplified in Figure 9.

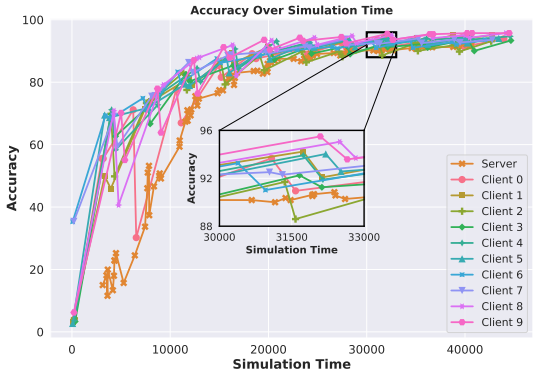

Figure 9: Accuracy of Clients and Server over Global Simulation Time on MNIST

This is made possible since the *disrete-event simulator* imposes a single simulation time, thus every client local update is happening in a fixed interval of simulation time.

8 Responsible Research

The following section will take a closer look on how the research we have conducted was done with integrity and reproducibility in mind.

Integrity: Every idea that was not originally ours was properly cited, as it is of great importance to give credit to previously accomplished work. For example, for the idea and the code of the *decentralized-learning-simulator* [13], which has no published paper, authors were contacted in order to get permission for us to reference them in this paper. To further ensure integrity, any ideas and results must be truthfully explained, without manipulating the results, which was done throughout the paper.

Reproduciblity: When it comes to reproducibility, it is of utmost importance for reviewers to be able to reproduce the results of the experiments. To this purpose all experiments and methods are defined in great detail, as can be seen in Section 6. Further, it is inevitable that randomness plays a factor in experimental results. In order to avoid randomness and provide better reproducibility, each experiment was ran multiple times after which the results were averaged. Further, for each different variant of the experiment we have used a different seed in the *discrete-event simulator*.

9 Conclusions and Future Work

In this section we will summarize what was accomplished throughout the paper and consider our downfalls and possible future improvements.

Conclusion: In the paper we have proposed a new approach to imposing time in Federated Learning simulations by formally modelling a *discrete-event simulator* and testing its efficiency in the asynchronous central server FL scenario against a continuous time simulator of *Flower*. Further, we have introduced a log-structure that is used to collect traces of the simulated run in order to foster debugging, which was made possible due to the *discrete-event simulator*. We have argued that with the new timing model the inefficiencies of the state-of-the-art simulators would be reduced: we would reduce variability between reruns and improve accuracy in the heterogeneous scenario. To this purpose we have conducted two experiments; one with heterogeneous clients and the other with homogeneous clients. From the experiment results, the *discrete-event simulator* has reduced standard deviation between reruns for 31.5% from the baseline framework of *Flower* and improved accuracy in the MNIST case with heterogeneous clients for 3.3% on average.

Future Work: The proposed *discrete-event simulator* can be improved by further developing the proposed random variables by gathering real-life FL deployment data. Also, with a sampling strategy that better encapsulates heterogeneity, the *discrete-event simulator* should be again tested for accuracy on CIFAR-10. Further, the simulator can be extended to incorporate multi-server synchronous and asynchronous scenarios, as those could introduce new conflicting variables when it comes to accuracy and timing dependencies between the clients in simulations.

References

[1] H. B. McMahan, E. Moore, D. Ramage, and B. A. y Arcas, "Federated learning of deep networks using model averaging," *arXiv preprint arXiv:1602.05629*, 2016.

[2] Y. Chen, Y. Ning, and H. Rangwala, "Asynchronous online federated learning for edge devices," *arXiv preprint arXiv:1911.02134*, 2019.

[3] B. Cox, L. Y. Chen, and J. Decouchant, "Aergia: leveraging heterogeneity in federated learning systems," in *Proceedings of the 23rd ACM/IFIP International Middleware Conference*, 2022, pp. 107–120.

[4] L. Liu, J. Zhang, S. Song, and K. B. Letaief, "Client-edge-cloud hierarchical federated learning," in *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, 2020, pp. 1–6.

[5] Y. Zuo, B. Cox, J. Decouchant, and L. Y. Chen, "Asynchronous multi-server federated learning for geo-distributed clients," *arXiv preprint arXiv:2406.01439*, 2024.

[6] A. G. Roy, S. Siddiqui, S. Pölsterl, N. Navab, and C. Wachinger, "Braintorrent: A peer-to-peer environment for decentralized federated learning," *arXiv preprint arXiv:1905.06731*, 2019.

[7] B. Cox, A. Mălan, J. Decouchant, and L. Y. Chen, "Asynchronous byzantine federated learning," *arXiv preprint arXiv:2406.01438*, 2024.

[8] C. Xie, S. Koyejo, and I. Gupta, "Asynchronous federated optimization," *arXiv preprint arXiv:1903.03934*, 2019.

[9] D. J. Beutel, T. Topal, A. Mathur, X. Qiu, T. Parcollet, and N. D. Lane, "Flower: A friendly federated learning research framework," *arXiv preprint arXiv:2007.14390*, 2020.

[10] E. Babulak and M. Wang, "Discrete event simulation: State of the art," in *Discrete Event Simulations*. InTech, 2010.

[11] G. F. Riley and T. R. Henderson, *The ns-3 Network Simulator*. Springer Berlin Heidelberg, 2010, pp. 15–34.

[12] A. Montresor and M. Jelasity, "Peersim: A scalable p2p simulator," in *2009 IEEE Ninth International Conference on Peer-to-Peer Computing*, 2009, pp. 99–100.

[13] sacs epfl, "decentralized-learning-simulator," https://github.com/sacs-epfl/decentralized-learning-simulator, 2024, accessed: 2024-05-27.

[14] A. Dhasade, A.-M. Kermarrec, R. Pires, R. Sharma, and M. Vujasinovic, "Decentralized learning made easy with decentralizepy," in *Proceedings of the 3rd Workshop on Machine Learning and Systems*, 2023, p. 34–41.

[15] M. H. Garcia, A. Manoel, D. M. Diaz, F. Mireshghallah, R. Sim, and D. Dimitriadis, "Flute: A scalable, extensible framework for high-performance federated learning simulations," 2022.

[16] L. Sani, P. P. B. de Gusmão, A. Iacob, W. Zhao, X. Qiu, Y. Gao, J. Fernandez-Marques, and N. D. Lane, "Pollen: High-throughput federated learning simulation via resource-aware client placement," 2024.

[17] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Z. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," *arXiv preprint arXiv:1912.01703*, 2019.

[18] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. J. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Józefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. G. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. A. Tucker, V. Vanhoucke, V. Vasudevan, F. B. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[19] K. Burlachenko, S. Horváth, and P. Richtárik, "Fl_pytorch: optimization research simulator for federated learning," *arXiv preprint arXiv:2202.03099*, 2022.

[20] M. D. Rossetti, *Simulation Modeling and Arena*, 3rd ed., 2021, licensed under the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License. [Online]. Available: https://rossetti.github.io/RossettiArenaBook/

[21] C. Faria and M. Correia, "Blocksim: Blockchain simulator," in *2019 IEEE International Conference on Blockchain (Blockchain)*, 2019, pp. 439–446.

[22] W. Zhao, X. Qiu, J. Fernandez-Marques, P. P. B. de Gusmão, and N. D. Lane, "Protea: client profiling within federated systems using flower," in *Proceedings of the 1st ACM Workshop on Data Privacy and Federated Learning Technologies for Mobile Edge Network*, 2022, p. 1–6.

[23] E. Ekaireb, X. Yu, K. Ergun, Q. Zhao, K. Lee, M. Huzaifa, and T. Rosing, "ns3-fl: Simulating federated learning with ns-3," in *Proceedings of the 2022 Workshop on Ns-3*, 2022, p. 97–104.

[24] C. T. Dinh, N. H. Tran, M. N. H. Nguyen, C. S. Hong, W. Bao, A. Y. Zomaya, and V. Gramoli, "Federated learning over wireless networks: Convergence analysis and resource allocation," *IEEE/ACM Transactions on Networking*, pp. 398–409, 2021.

[25] H. W. Lilliefors, "On the kolmogorov-smirnov test for normality with mean and variance unknown," *Journal of the American Statistical Association*, pp. 399–402.

[26] M. Alasmar, R. Clegg, N. Zakhleniuk, and G. Parisis, "Internet traffic volumes are not gaussian—they are log-normal: An 18-year longitudinal study with implications for modelling and prediction," *IEEE/ACM Transactions on Networking*, 2021.

[27] CloudPing, "Cloudping grid," 2024, accessed: 2024-05-30. [Online]. Available: https://www.cloudping.co/grid

[28] Software Freedom Conservancy, "iperf3: A tcp, udp, and sctp network bandwidth measurement tool," 2009–present. [Online]. Available: https://iperf.fr/

[29] r gg, "flower-async," https://github.com/r-gg/flower-async, 2024, accessed: 2024-06-08.

[30] L. Deng, "The mnist database of handwritten digit images for machine learning research [best of the web]," *IEEE Signal Processing Magazine*, pp. 141–142, 2012.

[31] A. Krizhevsky, "Learning multiple layers of features from tiny images," 2009. [Online]. Available: https://api.semanticscholar.org/CorpusID:18268744

[32] L. L. Ankile, M. F. Heggland, and K. Krange, "Deep convolutional neural networks: A survey of the foundations, selected improvements, and some current applications," *arXiv preprint arXiv:2011.12960*, 2020.