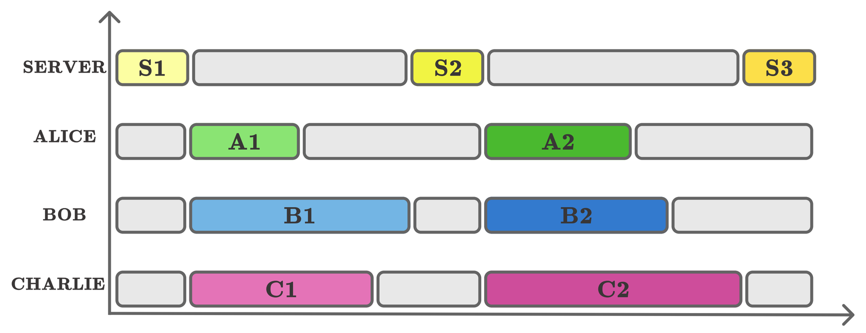
Fast Algorithms for Simulating Federated Learning Systems

Todor Slavov[1]

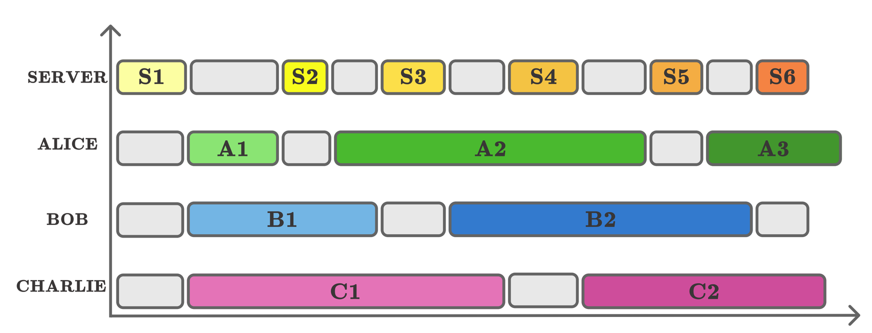
[1]Delft University of Technology

Introduction

In current times user devices hold significant amounts of data that's valuable for learning. Federated Learning (FL) has gained traction due to privacy and sensitivity concerns. FL preserves user privacy while effectively capturing data heterogeneity. In FL, a parameter server distributes a global model to user devices, which train it on local data. Client nodes send updates back to the server, which aggregates them into a local model, repeating this process iteratively.



(a) Synchronous Federated Learning Deployment



(b) Asynchronous Federated Learning Deployment

Figure 1. The coloured bars represent working time and the grey bars represent idle time. In the synchronous scenario, the parameter server waits for all clients to finish their jobs before aggregating. In the asynchronous scenario the parameter server aggregates as soon as any client is done updating.

Simulations

Deploying FL systems presents many technical challenges. To surmount these obstacles, researchers often turn to simulations, as a way to assess the efficacy of FL algorithms. In an FL simulation, one or potentially multiple machines do the work of a parameter server and client nodes to iteratively train local models and aggregate them into a global model.

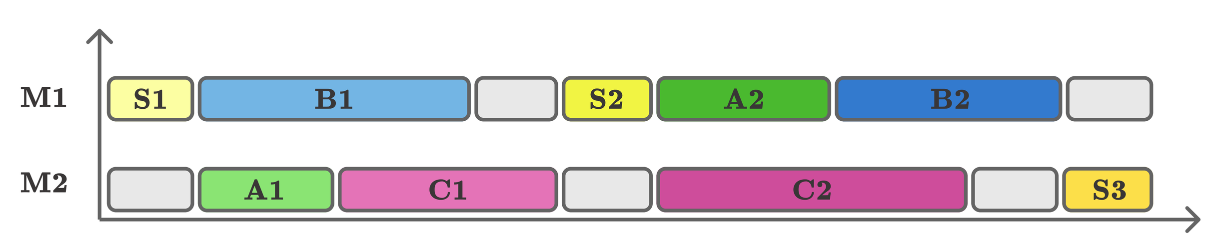
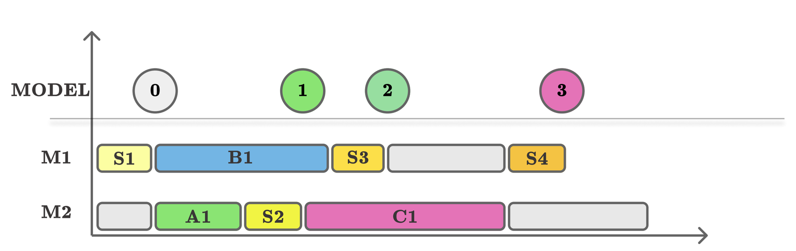


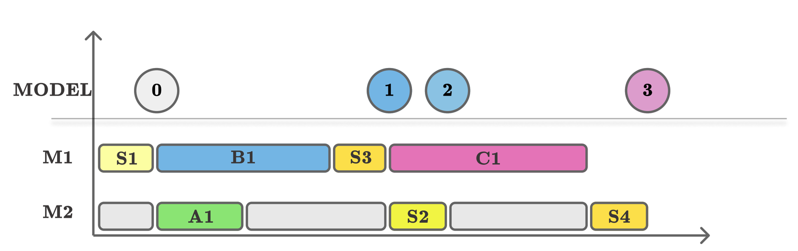
Figure 2. Simulation of the synchronous FL example from Fig 7a on two machines

Variance of resulting models

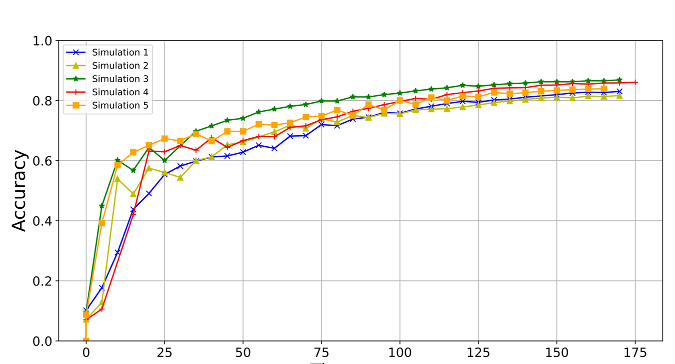
Due to the non-IID distribution of data over client nodes, if client C_A finishes its update and aggregation before client C_B, the resulting global model will be different than if the order was reversed. The more heterogenous the setting becomes the bigger the variability can be.



(a) First scenario



(b) Second scenario



(a) 5 runs of an FL simulation on identical dataset distributions.

Proposed Solution

In order to minimise variability we choose to sequence simulation by constraining the order of client updates. Every client update can be scheduled for execution only once all its predecessors have finished simulating. Hypothesis: simulating under precedence constraints reduces the variance of final trained model accuracies.

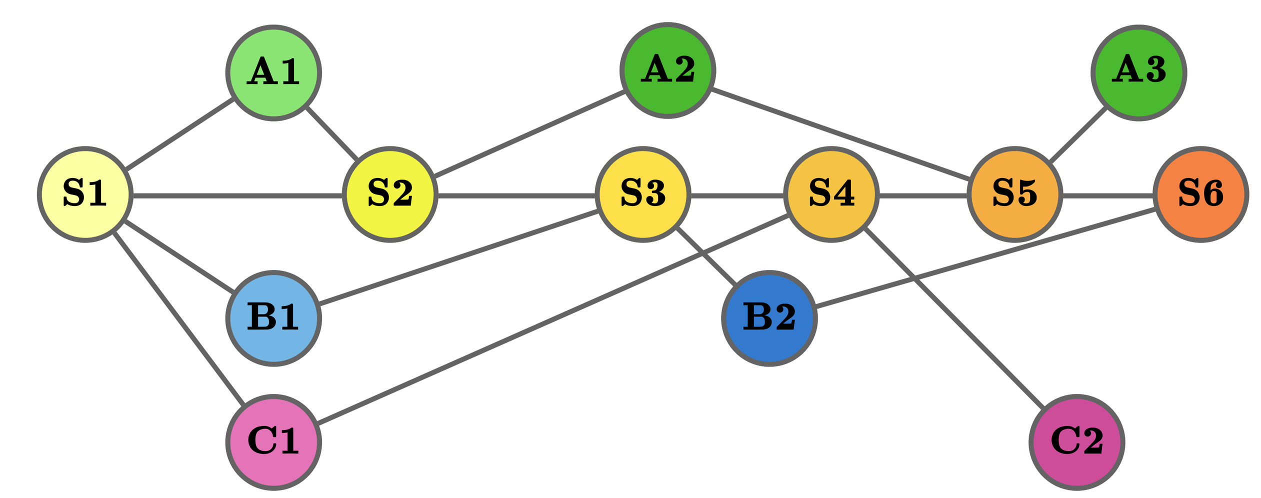
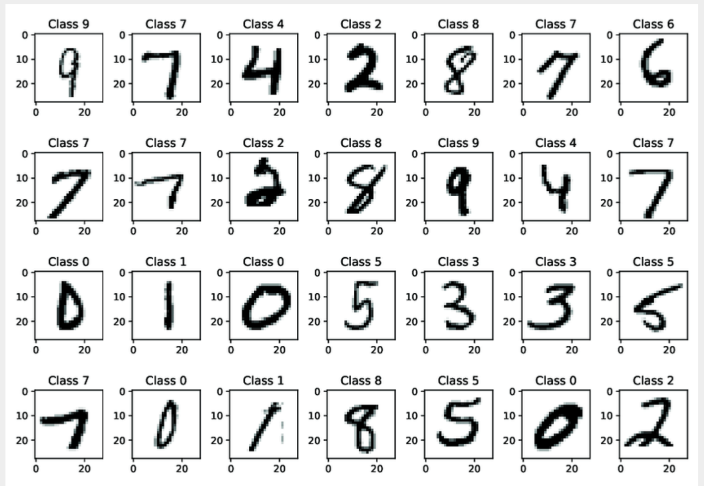


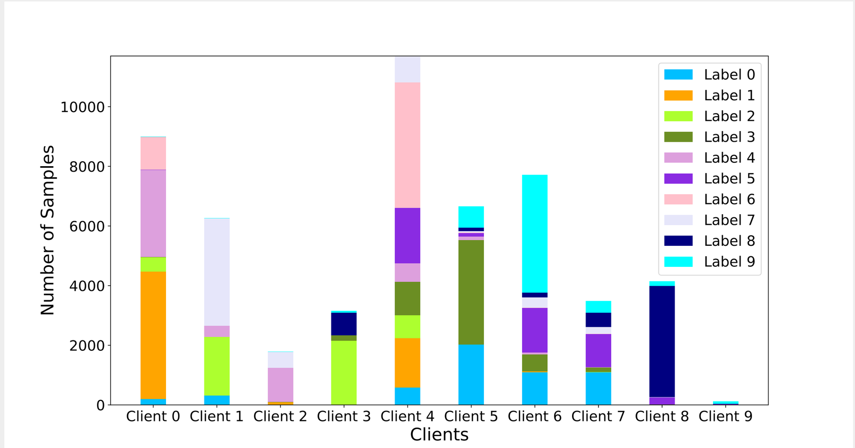
Figure 6. DAG for the asynchronous scenario - Fig 7b

Methodology

We conducted experiments to assess the performance and variability of Federated Learning algorithms using the MNIST dataset. We utilized a Multi-Layer Perceptron (MLP) model with a 784-neuron input layer, a 128-neuron hidden layer, and a 10-neuron output layer. The dataset, comprising 60,000 training and 10,000 test images, was split among clients in a highly unbalanced, non-IID manner, reflecting real-world data distribution scenarios. Our experiments, implemented using the Flower framework with PyTorch, examined how simulation scheduling constraints affect model accuracy variance and the makespan of different scheduling algorithms.



(a) MNIST dataset



(b) Class labels distribution over clients

Scheduling

$$P_m|prec, ST_{sd}|C_{\max} \tag{1}$$

The scheduling problem was shown to be NP-Hard by Ulman [1] so we propose two heuristic algorithms to solve the scheduling problem - Ant Colony Optimisation, and an Evolutionary Algorithm. Both are described below and pseudocode is provided. Furthermore, we propose a consistent heuristic that is used in combination with the A* algorithm to find optimal schedules for small problem instances.

Results

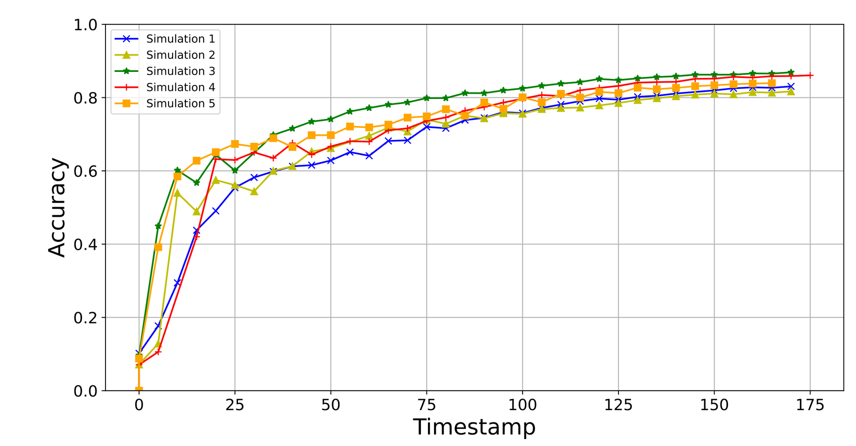


Figure 8. Speed of simulation of FL system.

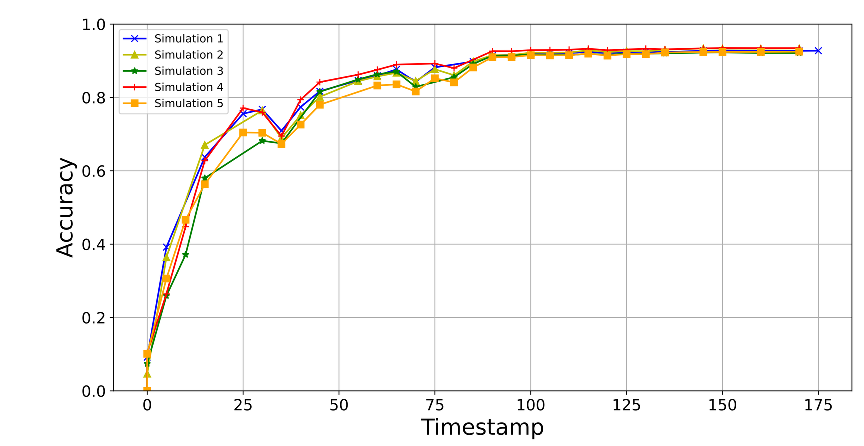


Figure 9. Variance in simulated FL models.

	S 1	S 2	S 3	S 4	S 5	μ	σ^2
normal	0.830	0.816	0.830	0.830	0.830	0.843	4.5×10^{-4}
	232.31	248.87	247.11	235.18	239.21	240.22	50.7
prec	0.926	0.936	0.923	0.930	0.929	0.929	2.3×10^{-5}
	314.31	321.45	328.41	309.81	326.1	319.6	64.3

Table 1. The five final model accuracies are given for two sets of simulations, one under no constraints, and one under precedence constraints.

		4	9	13	15	18
Dijkstra's	Nodes	28	1 646	1 012 715	5 428 190	
	Time	0.08	0.2	2.68	21.889	
	Makespan	15	21	35	36	
A^{*-LC}	Nodes	12	884	68 231	2 019 696	
	Time	0.04	0.16	0.218	5.110	
	Makespan	15	21	35	36	
A^{*-OF}	Nodes	28	1 132	163 022	337 268	5 230 117
	Time	0.07	0.19	0.489	3.754	14.695
	Makespan	15	21	35	36	45

Table 3. Results of Dijkstra's algorithm, A^* with LC heuristic, and A^* with OF heuristic run on 5 problem instances of varying sizes.

		4	9	13	15	18
Random	Time	$4.3{\times}10^{-5}$	$7.7{\times}10^{-5}$	$1.1{\times}10^{-4}$	$8.1{\times}10^{-5}$	$1.3{\times}10^{-4}$
	Makespan	15	23	36	44	51
ACO	Time	0.004	0.016	0.025	0.031	1.036
	Makespan	15	24	36	39	47
EA	Time	0.138	0.325	0.389	0.434	0.511
	Makespan	15	21	35	36	45

Table 2. Random schedule, ACO, and EA assesed on 5 problem instances with 4, 9, 13, 15, and 18 jobs to be scheduled on 2 machines. For each algorithm, it is given how much time (in seconds) it took the algorithm to arrive to a solution, and what was the makespan of the final schedule.

		50	100	250	1000
Random	Time	$2.9{\times}10^{-4}$	$1.1{\times}10^{-3}$	$1.3{\times}10^{-3}$	$5.9{\times}10^{-3}$
	Makespan	99	296	618	3019
ACO	Time	2.146	3.501	4.712	67.98
	Makespan	90	275	512	2458
EA	Time	1.414	3.533	16.14	216.79
	Makespan	78	269	554	2552

Table 4. Random schedule, ACO, and EA assesed on 5 problem instances with 50, 100, 250, and 1000 jobs to be scheduled on 3 machines. For each algorithm, it is given how much time (in seconds) it took the algorithm to arrive to a solution, and what was the makespan of the final schedule.

References

[1] J. D. Ullman, "Np-complete scheduling problems," *Journal of Computer and System sciences*, vol. 10, no. 3, pp. 384–393, 1975.