Optimizing Execution Time and Costs of Cross-Silo Federated Learning Applications with Datasets on different Cloud Providers

Rafaela C Brum, Pierre Sens, Luciana Arantes, Maria Clicia Castro, Lucia Maria de A. Drummond

Optimizing Execution Time and Costs of Cross-Silo Federated Learning Applications with Datasets on different Cloud Providers

Rafaela C. Brum
Fluminense Federal University
Institute of Computing
Niterói, Brazil
Sorbonne Université, CNRS, INRIA, LIP6
Paris, France
rafaelabrum@id.uff.br

Pierre Sens
Sorbonne Université, CNRS, INRIA, LIP6
Paris, France
pierre.sens@lip6.fr

Luciana Arantes
Sorbonne Université, CNRS, INRIA, LIP6
Paris, France
luciana.arantes@lip6.fr

Maria Clicia Castro
State University of Rio de Janeiro
Institute of Mathematics and Statistics
Rio de Janeiro, Brazil
clicia@ime.uerj.br

Lúcia Maria de A. Drummond
Fluminense Federal University
Institute of Computing
Niterói, Brazil
lucia@ic.uff.br

***Abstract*—Under the coordination of a central server, Federate Learning (FL) enables a set of clients to collaboratively train a global machine learning model without exchanging their local data. When such clients have powerful machines, it is called cross-silo FL, and they store their data in private repositories denoted silos. We are interested in this paper in cross-silo FL where silos are geographically located in different regions of multi-cloud providers. Thus, aiming at minimizing financial costs and execution times of a cross-silo FL application, we propose a model based on a scheduling problem mathematical formulation, which receives as input both the application parameters and the cloud providers' resource features where clients' data are stored and renders the best assignment of clients and server to virtual machines. This formulation is part of a framework proposal to execute FL applications in different cloud providers. Taking as a use case a Tumor-Infiltrating Lymphocytes Classification problem, an FL application whose clients' datasets spread over different cloud providers' data repositories, evaluation results show that our model is scalable and improves the execution time and financial costs of the FL application by up to 53.70% and 48.34% in a scenario with 50 clients, executing in around 200 seconds, when compared to results where VMs are randomly selected. Experimental results with client silos in different Google (GCP) and Amazon (AWS) cloud regions also confirmed the effectiveness of our proposed model in a real multi-cloud environment.**

***Index Terms*—Scheduling Problem, Federated Learning, Multi-Cloud**

I. Introduction

Proposed by McMahan *et al.* [1], Federated Learning (FL) is an emerging distributed Machine Learning (ML) technique where the participants (clients) collaboratively train a ML model without sharing their respective private data [2]. Under the coordination of a central server, each client trains the model locally and communicates only the model weights to the server that, in its turn, updates its global model. Federated Learning has received much attention during the past few years due to the increasing concern with data privacy, particularly with current data protection laws (*e.g.*, GPDR[1] in Europe).

The client-server architecture of FL, also called Model-Centric Federated Learning [3], can be classified into Cross-Device or Cross-Silo Federated Learning. The former has low-powered clients, such as mobile phones [1], while the latter has few clients (*e.g.*, companies, institutions, hospitals [4]) with private large dataset repositories, denoted silos.

This work focuses on Cross-Silo Federated Learning, also taking into account that, due to datasets gradually growing [5], many clients keep their silos in cloud storage repositories [6], [7] which offer data privacy guarantees and availability. In such a context, we also consider that client silos can be stored in different cloud providers (*e.g.* Amazon (AWS) S3, Google Cloud (GCP) Storage, etc.). Moreover, the physical infrastructure of a cloud provider is usually divided into independent and isolated geographic regions [8], [9], each of them offering several types of virtual machines with different performance and financial costs. Consequently, a multi-cloud platform offers multiple choices of virtual machine types, regions, and cloud providers, with different execution and communication times and costs, for executing the clients and the server of an FL application.

Therefore, considering different types, regions, and cloud providers, the question is how to select the best set of virtual machines to execute the clients and the server of an

[1]https://gdpr-info.eu/

FL application, whose datasets were previously allocated in one or more storage systems of one of the cloud providers, minimizing both the execution time and the financial cost of the FL application. In order to solve this scheduling problem, we propose in this paper a mathematical formulation model which receives as input information about the FL application as well as about the multi-cloud environment and provides a weighted optimization function whose aim is to minimize the conflicting objectives cost and time. Our mathematical model is part of a framework proposal to execute FL applications in a multi-cloud environment, aiming to minimize time and costs while keeping data privacy.

This framework receives as input information concerning the FL application (e.g. number of clients, location of each client dataset, number of communication rounds, etc) and the environment (e.g., number of CPUs and GPUs in each VM, the price of each VM in each region, limits of VMs per region, etc) and outputs a scheduling map. It also deploys the selected VMs and starts the tasks following the obtained map.

The model has been evaluated theoretically and in a multi-cloud platform with AWS and GCP. To this end, we chose a use-case application, which searches for Tumor-Infiltrating Lymphocytes (TILs) in high-resolution scanned human tissue images aiming to understand the patient's cancer extension and predicting the most suitable treatments.

Experiments with different data placement scenarios of such an application have been conducted for evaluating the scalability of the model, its effectiveness when compared to a random user VM selection, and its accuracy when compared to a real multi-cloud environment with a client's silo in either Google Cloud Storage or AWS S3 of different regions.

The paper is organized as follows. Section II introduces related work. Section III presents our mathematical formulation to the problem of scheduling FL clients and the server to a multi-cloud environment. Section IV and Section V briefly describe the use-case application and Flower, a tool that creates FL systems. Section VI presents and discusses experimental evaluation results. Finally, Section VII concludes the paper and proposes some future work.

II. Related work

Most existing Federated Learning works tackle Cross-Device architectures, where the server samples a fraction of clients in each communication round, among hundreds or thousands of connected devices [10], [11], while few of them focus on Cross-Silo Federated Learning [4], [12], [13]. However, to the best of our knowledge, none of the latter consider the scheduling problem of FL clients and the server in a cloud or datacenter environment.

Regarding distributed machine learning in general, a majority of works focus on the job scheduling problem of multiple ML jobs that should be allocated among a number of limited resources [14]–[16]. Few of them handle the scheduling problem of a unique distributed ML application. In [17], Amiri *et al.* consider that some tasks (called workers) may present sporadic slowness due, for instance, to temporary network disconnection. To overcome such a problem, the central dataset is divided into small n chunks. Each worker should then compute a number of them sequentially. Thus, a server task sends chunks to multiple workers and waits for k results, $k \leq n$. The authors propose two scheduling schemes for chunks assignment to the workers aiming at minimizing the average completion time. However, the data-sharing approach to this problem is not suitable for Federated Learning scenarios where data privacy and heterogeneity are essential concerns.

III. Scheduling FL Clients and Server to a Multi-Cloud Environment

In this section, we present our application and multi-cloud models. Then, we describe our mathematical formulation to schedule FL clients and server into a multi-cloud environment.

A. FL application Model

An FL application is a distributed algorithm, composed of a set of clients, C, and a server, s, that uses communication barriers along execution. We consider in this work, that the FL application executes a set of rounds, each one divided into five steps. In the first one, the server sends the message s_msg_{train} containing the model weights to all clients. After receiving the weights, in Step 2, each client, $c_i \in C$, trains the neural network for a fixed number of epochs on the local training dataset and sends the updates back to the server, through the message c_msg_{train}. Then, in Step 3, the server receives all updates, aggregates them, and sends the final weights to the clients (s_msg_{aggreg} message). Upon reception, in Step 4, each client updates their respective weights, tests the model with the test dataset, and sends its evaluation metrics (for example, accuracy and precision) to the server in the message c_msg_{test}. Finally, in Step 5, the server aggregates the evaluation metrics of all clients to have the global metrics. Figure 1 shows the execution steps of a round.

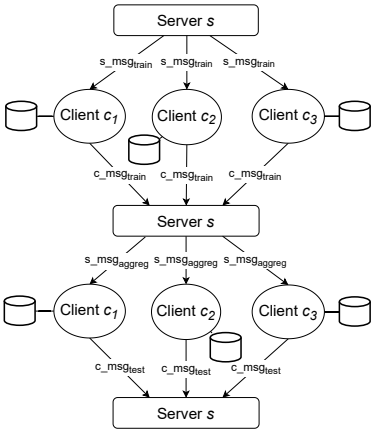

Fig. 1. Steps of one round of a Federated Learning Application

B. Multi-Cloud Model

Usually, commercial clouds have their physical infrastructures spread in different regions, which are independent and

isolated geographically. For instance, there are currently 26 regions in AWS [8] and 34 regions in GCP [9].

Each region offers computational resources packaged as Virtual Machines (VMs). Each VM has a number of virtual cores, named virtual CPUs (vCPUs), and may have one or more Graphics Processing Units (GPUs) connected to it. Thus, let P be a set of available cloud providers. A provider $p_j \in P$ has associated to it a set of regions R_j and a fixed cost $cost\_t_j$ (in $ per GB) to send any message from itself to any other VM, inside or outside the provider, as observed experimentally. A provider p_j usually offers a limited number of GPUs (N_GPU_j) and vCPUs (N_CPU_j). Moreover, in each region $r_{jk} \in R_j$, the number of available GPUs and vCPUs are bounded ($N\_L\_GPU_{jk}$ and $N\_L\_CPU_{jk}$, respectively).

Yet, each region $r_{jk} \in R_j$ has a set of available VM instance types, V_{jk}, that can be deployed in it, where each $vm_{jkl} \in V_{jk}$ contains a number of vCPUs, cpu_{jkl} and a number of GPUs, gpu_{jkl}, with a fixed hiring cost (in $ per second) $cost_{jkl}$.

C. Mathematical Formulation

The problem of scheduling tasks in distributed computing resources is an NP-complete one [18], even in simple scenarios. Furthermore, some features of multi-clouds render it more difficult. Thus, we model our scheduling problem as a Mixed-Integer Linear Programming problem. In our scheduling problem, we have two objectives: minimize the monetary cost and the total execution time (makespan) of the application. Here, we also consider that the obtained solution has to respect the constraints of a deadline, named T, and of a budget, called B, given by the user.

Considering that the FL application executes n_rounds, each one for a similar amount of time, we can divide B and T by n_rounds to obtain the maximum budget and deadline for a single FL round, named B_{round} and T_{round}, respectively. Hence, we formulate the scheduling problem for one round.

Table I presents the notation and variables used in the mathematical formulation. The proposed objective function (Equation 1) is a weighted function that minimizes the monetary cost, $total\_costs$, and the makespan, t_m, of a single FL round, where α, ranging from zero to one, is the weight given by the user for the objectives. Usually, once those objectives are conflicting, they cannot reach the optimal values simultaneously. For instance, for α values close to 1, the formulation prioritizes low costs solutions rather than small makespans. On the other hand, when $\alpha = 0.5$, both objectives receive the same importance.

$$\min \alpha \times total\_costs + (1 - \alpha) \times t_m \tag{1}$$

Let the binary variables x_{ijkl} indicate whether a client c_i will execute on VM vm_{jkl} of region r_{jk} of provider p_j ($x_{ijkl} = 1$), or not ($x_{ijkl} = 0$), and let y_{jkl} be the analogous representation for the server s. The variable $total\_costs$ includes the VMs' execution costs (vm_costs), computed as Equation 2, and the message transfer costs ($comm\_costs$), described in Equation 3, where $comm_{jm}$, presented in Equation 4, is the cost to exchange the FL messages between provider p_j and p_m (j can be equal to m). Therefore, $total\_costs$ is calculated as in Equation 5.

$$vm\_costs = \sum_{c_i \in C} \sum_{p_j \in P} \sum_{r_{jk} \in R_j} \sum_{vm_{jkl} \in V_{jk}} (x_{ijkl} \times cost_{jkl} \times t_m) + \sum_{p_j \in P} \sum_{r_{jk} \in R_j} \sum_{vm_{jkl} \in V_{jk}} (y_{jkl} \times cost_{jkl} \times t_m) \tag{2}$$

$$comm\_costs = \sum_{c_i \in C} \sum_{p_j \in P} \sum_{r_{jk} \in R_j} \sum_{vm_{jkl} \in V_{jk}} \sum_{p_m \in P} \sum_{r_{mn} \in R_m} \sum_{vm_{mno} \in V_{mn}} (x_{ijkl} \times y_{mno} \times comm_{jm}) \tag{3}$$

$$comm_{jm} = (size(s\_msg_{train}) + size(s\_msg_{aggreg})) \times cost\_t_m + (size(c\_msg_{train}) + size(c\_msg_{test})) \times cost\_t_j \tag{4}$$

$$total\_costs = vm\_costs + comm\_costs \tag{5}$$

Both the monetary cost and the makespan have to be first normalized. The normalization procedure updates the target values to share the same minimum and maximum values. Thus, $total\_costs$ is divided by the product of the monetary cost of hiring the most expensive VM, the maximum execution time of an FL round (T_{max}), and the number of tasks (clients and server); plus the product of the most expensive message exchange between providers ($comm_{jm}$) and the number of clients, as shown in Equation 6. Similarly, the makespan is divided by the maximum execution time of a FL round (T_{max}).

$$\max_{p_j \in P, r_{jk} \in R_j, vm_{jkl} \in V_{jk}} (cost_{jkl}) \times T_{max} \times (|C| + 1) + \max_{p_j, p_m \in P} (comm_{jm}) \times |C| \tag{6}$$

The objective function is subject to the following constraints. Constraints 7 and 8 guarantee that the budget and deadline for a FL application round are not violated, while constraints 9 and 10 ensure that every client and the server execute on a single VM $vm_{jkl} \in V_{jk}$ in a region $r_{jk} \in R_j$ of a provider $p_j \in P$.

$$total\_costs \leq B_{round} \tag{7}$$

$$t_m \leq T_{round} \tag{8}$$

$$\sum_{p_j \in P} \sum_{r_{jk} \in R_j} \sum_{vm_{jkl} \in V_{jk}} x_{ijkl} = 1, \forall c_i \in C \tag{9}$$

$$\sum_{p_j \in P} \sum_{r_{jk} \in R_j} \sum_{vm_{jkl} \in V_{jk}} y_{jkl} = 1 \tag{10}$$

Constraints 11 to 14 guarantee that the solution will not exceed the maximum number of GPUs and vCPUs of each provider. Constraints 11 ensure that the total number of available GPUs does not exceed the global limit, while constraints 12 limit the total number of available vCPUs in each provider. Constraint 13 has the same meaning as 11, but here the GPU limit is by region. Constraint 14 is the vCPUs limit constraints per region, similarly as Constraint 12.

$$\sum_{c_i \in C} \sum_{r_{jk} \in R_j} \sum_{vm_{jkl} \in V_{jk}} x_{ijkl} \times gpu_{jkl} + \sum_{r_{jk} \in R_j} \sum_{vm_{jkl} \in V_{jk}} (y_{jkl} \times gpu_{jkl}) \leq N\_GPU_j, \forall p_j \in P \tag{11}$$

TABLE I
NOTATION AND VARIABLES USED.

Name	Description
C	Set of clients in the FL application
P	Set of available cloud providers
p_j	A cloud provider
R_j	Set of regions available in provider p_j
r_{jk}	A region of provider p_j
V_{jk}	Set of instance types available in region r_{jk}
vm_{jkl}	A instance type of region r_{jk}
B_{round}	Maximum budget to a single FL round
T_{round}	Maximum deadline to a single FL round
T_{max}	Maximum time of a single FL round
N_GPU_j	Maximum number of available GPUs in the provider p_j
N_CPU_j	Maximum number of available vCPUs in the provider p_j
$cost\_t_j$	Cost (in $ per GB) of sending a message from provider p_j
$N\_L\_GPU_{jk}$	Maximum number of available GPUs in region r_{jk}
$N\_L\_CPU_{jk}$	Maximum number of available vCPUs in region r_{jk}
cpu_{jkl}	Number of vCPUs in the instance type vm_{jkl}
gpu_{jkl}	Number of GPUs in the instance type vm_{jkl}
$cost_{jkl}$	Cost (in $ per second) of instance type vm_{jkl}
c_i	A client of the FL application
$size(s\_msg_{train})$	Size of training message sent by the server to a client
$size(s\_msg_{aggreg})$	Size of test message sent by the server to a client
$size(c\_msg_{train})$	Size of training message sent by a client to the server
$size(c\_msg_{test})$	Size of test message sent by a client to the server
$total\_costs$	Total financial costs of a single FL round
t_m	Total execution time (makespan) of a FL round
t_exec_{ijkl}	Computational time (training and test) of client c in vm_{jkl}
t_comm_{jklm}	Communication time (training and test messages) between region r_{jk} and region r_{lm}
t_aggreg_{jkl}	Aggregation time of server in vm_{jkl}
vm_costs	Total financial cost of all VMs in a single FL round
$comm\_costs$	Total financial cost of message exchange within a FL round
x_{ijkl}	Binary variable which indicates if client c_i executes on vm_{jkl} or not
y_{jkl}	Binary variable which indicates if the server executes on vm_{jkl} or not
α	Weight given by the user for the objectives (ranging from 0 to 1)

$$\sum_{c_i \in C} \sum_{r_{jk} \in R_j} \sum_{vm_{jkl} \in V_{jk}} x_{ijkl} \times cpu_{jkl} + \sum_{r_{jk} \in R_j} \sum_{vm_{jkl} \in V_{jk}} (y_{jkl} \times cpu_{jkl}) \leq N\_CPU_j, \forall p_j \in P \quad (12)$$

$$\sum_{c_i \in C} \sum_{vm_{jkl} \in V_{jk}} x_{ijkl} \times gpu_{jkl} + \sum_{vm_{jkl} \in V_{jk}} y_{jkl} \times gpu_{jkl} \leq N\_L\_GPU_{jl}, \ \forall p_j \in P, \forall r_{jk} \in R_j \quad (13)$$

$$\sum_{c_i \in C} \sum_{vm_{jkl} \in V_{jk}} x_{ijkl} \times cpu_{jkl} + \sum_{vm_{jkl} \in V_{jk}} y_{jkl} \times cpu_{jkl} \leq N\_L\_CPU_{jl}, \ \forall p_j \in P, \forall r_{jk} \in R_j \quad (14)$$

Next, constraint 15 guarantees that the FL application will finish before t_m, which is considered as the makespan of the FL round. Finally, constraints 16 and 17 define the domain of the decision variables x_{ijkl} and y_{jkl}, respectively.

$$x_{ijkl} \times y_{mno} \times (t\_exec_{ijkl} + t\_comm_{jkmn} + t\_aggreg_{mno}) \leq t_m, \forall c_i \in C, \forall p_j, p_m \in P, \forall r_{jk} \in R_j, \forall vm_{jkl} \in V_{jk}, \forall r_{mn} \in R_m, \ \forall vm_{mno} \in V_{mn} \quad (15)$$

$$x_{ijkl} \in \{0,1\}, \forall c_i \in C, \forall p_j \in P, \forall r_{jk} \in R_j, \forall vm_{jkl} \in V_{jk} \quad (16)$$

$$y_{jkl} \in \{0,1\}, \forall p_j \in P, \forall r_{jk} \in R_j, \forall vm_{jkl} \in V_{jk} \quad (17)$$

IV. CASE STUDY: TUMOR-INFILTRATING LYMPHOCYTES CLASSIFICATION

The selected use-case application searches for Tumor-Infiltrating Lymphocytes (TILs) in high-resolution scanned human tissue images and creates maps representing their spatial distribution. However, the Whole Slide Tissue Images (WSIs) of scanned tissue can have up to 100K×100K pixels, making it difficult for an expert to extract information from thousands of WSIs available in research studies. Because of that, Saltz *et al.* [19] developed a deep learning application to classify and analyze TIL patterns from tissue images.

For the evaluation of our proposal, we focus on the CNN training, which is one step of the application. We assume that the WSIs are previously divided into patches and classified by an expert. The centralized version of the CNN training in this TIL classification problem implements the VGG16 model [20], which has an input layer of size $224 \times 224 \times 3$. After this layer, the model has several convolutional and fully connected layers, ending with an output layer that determines if the TIL is positive or negative. The centralized version was implemented with the TensorFlow API [21], which optimizes dataset readings by using multiple threads to load the images from the main memory and transfer them to the GPU memory.

V. Federated learning using the Flower Framework

We used the Flower framework, proposed by Beutel *et al.* [22], to build our federated learning system. Flower offers facilities to execute FL experiments, considering heterogeneous clients on simulation and real-world scenarios. It supports different ML frameworks underneath it (e.g., TensorFlow, PyTorch, or a custom one). The FL client implements the VGG16 model [20], using the TensorFlow API [21].

For the aggregation function to the server, we chose the FedAvg [1] since it is already implemented in Flower and gives good results on different unbalanced schemes, as shown in Li *et al.* [13]. FedAvg computes the weighted average from all clients using their respective dataset size as associated weight.

VI. Performance Evaluation

For our experiments, we considered Amazon Web Services (AWS) and Google Cloud Provider (GCP) as cloud providers with two regions at each of them: N. Virginia (*us-east-1*) and Oregon (*us-west-2*) regions in AWS; and Iowa (*us-central1*) and Oregon (*us-west1*) regions in GCP.

Initially, we planned to have a testbed composed of several VMs with GPU in both AWS and in GCP, varying from the Kepler GPU architecture to the Ampere one. However, the CUDA version 10.0, used by the use-case TILs application, is not compatible with either of these two architectures. Consequently, we had to discard such GPU architectures from our testbed. Furthermore, experiments showed that a VM with the Volta GPU in AWS had constantly higher execution times than with the Turing one, which led us to also discard the former from our testbed, due to its unexpected behaviour. Finally, in other experiments, we observed that it was not possible to allocate, in either of the two regions, some types of GPU architectures in GCP. Thus, such GPUs were not taken into account either.

Hence, considering all the above constraints, for each region in AWS, we selected two instance types with GPU, Maxwell and Turing, and one instance without GPU. For GCP, we chose instances from the general-purpose N1 family with the same number of vCPUs as the AWS counterparts. Each instance had one GPU, Pascal, Turing or Volta, attached to it. Additionally, we also selected an instance without GPU. Note that in GCP, some GPU architectures are not available in all regions, thus, the number of instance types varies for each region.

Table II summarizes our VM instance selection in both platforms, giving also the amount of memory and cost per hour (obtained in May 2022) of each instance, and IDs that allows us to identify each one. Message transfer cost in AWS is $0.09 per GB in the first 10 TB/month and in GCP is $0.12 per GB in the first 1 TB/month. For storing clients' datasets in AWS and GCP, we respectively chose the Amazon Simple Storage Service (S3) [23] in the N. Virginia region (*us-east-1*) and the Cloud Storage [24] in the Iowa region (*us-central1*).

A. Training and test times

In order to have the best scheduling map of a Federated Learning application, it is necessary to know clients' and server execution times with corresponding financial costs as well as the communication time between a client and a server, considering all available VMs. However, to run the whole application in order to obtain those execution times and financial costs would be unrealistic.

Execution times: In [25], Malta *et al.* observed that all training epochs of a centralized Deep Learning training have similar execution times, except for the first one. Consequently, to obtain the total execution time, only the time of the first two epochs is necessary. The authors proposed then the Equation 18 that renders $Total\_runtime$, the total time of the application, where $T(Ep_1)$ and $T(Ep_2)$ are the execution times of the first and second training epoch, respectively, and N_{ep} is the total number of training epochs.

$$Total\_runtime = T(Ep_1) + (N_{ep} - 1) \times T(Ep_2) \tag{18}$$

The same behaviour of the previous work was observed in the FL approaches presented in [26] and [27] where the first FL round has a different execution time from all the other ones, which present similar execution times. Therefore, in this work, to obtain the execution time of our use-case FL application, only its first two rounds were executed. Note that execution time includes both CPU and storage access times.

Let define the *slowdown* sl_inst_{jkl} as the ratio between the execution time of a dummy application in one of the virtual machines and the execution time in a chosen baseline virtual machine, in our experiments, the AWS instance *g4dn.2xlarge* executing in the N. Virginia region (*us-east-1* region), vm_{111}. Thus, the execution time of a round of a client c_i executing on a provider p_j, region r_{jk} and virtual machine vm_{jkl} is the sum of training and test steps of client c_i executed on the baseline virtual machine ($train\_bl_i$ and $test\_bl_i$ respectively) multiplied by the corresponding slowdown (sl_inst_{jkl}), as shown in Equation 19.

$$t\_exec_{ijkl} = (train\_bl_i + test\_bl_i) \times sl\_inst_{jkl} \tag{19}$$

Note that once having obtained slowdown values for one dataset size, we can execute the FL application with other dataset sizes, without the need of re-executing the first two rounds of the application in all virtual machine types, except for the baseline. We created a dummy application that simulates the execution of one client in the first two rounds of our use-case FL application to compute this slowdown. Three experiment sets were conducted.

In the first experiment set, client datasets were stored in AWS and, in each experiment, only one client was executed in a single instance vm_{jkl} of Table II with an attached GPU. The slowdown of a vm_{jkl} is computed by dividing the sum of the second round execution (training and test) times in this instance by the sum of the second round execution in vm_{111}, our baseline instance. Table III presents the average values of each step (training and test) of the first (1º r.) and second

TABLE II
INSTANCE TYPES SELECTED

Prov.	Region	VM	vCPUS	RAM (GB)	GPU	GPU memory (GB)	Costs per hour ($)	ID
AWS	N. Virginia (us-east-1)	*g4dn.2xlarge*	8	32	Nvidia Tesla T4 Tensor Core	16	0.7520	vm_{111}
		g3.4xlarge	16	122	Nvidia Tesla M60	8	1.1400	vm_{112}
		t2.xlarge	4	16	-	-	0.1856	vm_{113}
	Oregon (us-west-2)	*g4dn.2xlarge*	8	32	Nvidia Tesla T4 Tensor Core	16	0.7520	vm_{121}
		g3.4xlarge	16	122	Nvidia Tesla M60	8	1.1400	vm_{122}
		t2.xlarge	4	16	-	-	0.1856	vm_{123}
GCP	Iowa (us-central1)	*n1-standard-8* with Turing GPU	8	30	Nvidia Tesla T4 Tensor Core	16	0.7300	vm_{211}
		n1-standard-16 with Pascal GPU	16	60	Nvidia Testa P4	8	1.3600	vm_{212}
		n1-standard-8 with Volta GPU	8	30	Nvidia V100 Tensor Core	16	2.8600	vm_{213}
		e2-standard-4	4	16	-	-	0.1340	vm_{214}
	Oregon (us-west1)	*n1-standard-8* with Turing GPU	8	30	Nvidia Tesla T4 Tensor Core	16	0.7300	vm_{221}
		n1-standard-8 with Volta GPU	8	30	Nvidia V100 Tensor Core	16	2.8600	vm_{222}
		e2-standard-4	4	16	-	-	0.1340	vm_{223}

TABLE III
EXECUTION TIMES OF ONE CLIENT WITH FIVE LOCAL EPOCHS, RUN IN DIFFERENT INSTANCES OF AWS AND GCP AND DATASET STORED IN AMAZON S3 IN N. VIRGINIA REGION (*us-east-1*)

Prov.	Region	VM ID	Training time		Test time		Sl
			1º r.	2º r.	1º r.	2º r.	
AWS	N. Virginia (*us-east-1*)	vm_{111}	04:17	06:53	03:13	03:03	1.00
		vm_{112}	08:14	31:23	16:29	19:09	5.09
	Oregon (*us-west-2*)	vm_{121}	04:34	06:34	02:40	03:14	0.99
		vm_{122}	07:34	27:47	14:48	16:19	4.44
GCP	Iowa (*us-central1*)	vm_{211}	03:24	07:09	03:01	03:04	1.03
		vm_{212}	04:27	07:53	03:41	04:47	1.28
		vm_{213}	02:59	07:20	03:20	02:56	1.04
	Oregon (*us-west1*)	vm_{221}	03:51	07:24	02:40	03:11	1.07
		vm_{222}	03:11	07:23	03:12	03:31	1.10

TABLE IV
EXECUTION TIMES OF ONE CLIENT WITH FIVE LOCAL EPOCHS, RUN IN DIFFERENT INSTANCES OF AWS AND GCP AND DATASET STORED IN GCP CLOUD STORAGE IN IOWA REGION (*us-central1*)

Prov.	Region	VM ID	Training time		Test time		Sl
			1º r.	2º r.	1º r.	2º r.	
AWS	N. Virginia (*us-east-1*)	vm_{111}	04:21	03:04	00:54	00:49	1.00
		vm_{112}	06:09	04:45	01:06	01:09	1.52
	Oregon (*us-west-2*)	vm_{121}	05:20	04:01	01:15	01:15	1.36
		vm_{122}	07:27	05:48	01:42	01:40	1.92
GCP	Iowa (*us-central1*)	vm_{211}	03:05	02:44	00:48	00:32	0.84
		vm_{212}	03:39	02:57	00:41	00:30	0.89
		vm_{213}	01:36	01:11	00:36	00:26	0.42
	Oregon (*us-west1*)	vm_{221}	04:05	02:56	00:58	00:53	0.99
		vm_{222}	02:41	02:41	01:08	00:49	0.90

(2º r.) round, and the computed slowdown (Sl). We executed the client in each instance type three times and observed an average standard deviation of 10.96%. The second experiment set is similar to the previous one, but client datasets are stored in GCP. Results are presented in Table IV with average standard deviation of 11.93%. Finally, the aggregation task of the server was executed in all virtual machines of AWS and GCP, taking around 0.3 seconds in AWS and 0.2 seconds in GCP.

Communication times: Let define sl_comm_{xy}, the communication slowdown of the pair of regions r_x (with $x = jk$) and r_y (with $y = lm$), as the ratio between the communication time of FL messages between these regions and the communication time in a baseline pair of regions. Here, we define the baseline pair by considering the same region in both sides, the N. Virginia region of AWS (*us-east-1* region). The communication time between regions r_x and r_y is thus given by Equation 20, with $train\_comm\_bl$, the communication time of the training messages, and $test\_comm\_bl$, the communication time of the test messages in the baseline pair of regions.

$$t\_comm_{xy} = (train\_comm\_bl + test\_comm\_bl) \times sl\_comm_{xy} \quad (20)$$

This slowdown sl_comm_{xy} is computed as the sum of both communication times between client and server divided by the sum of both times in the baseline pair of regions. In order to obtain this communication slowdown, we considered a dummy application with a single client and the server and measured the total time taken by the client to send and receive messages from the server. Neither the server nor the client does any computation with the messages, they only receive them and send them back. Messages had 25,000,000 floating points (size of the dummy model) which correspond to messages of 1GB size. We collected communication slowdown times of 10 executions per pair of regions r_{jk} and r_{lm}, having 5 of them with the client in the region r_{jk} and the server in the region r_{lm} and 5 the other way round. Table V presents the average times in seconds in all possible pairs of regions and the computed slowdown (Sl). Most standard deviations were below 15%, with only three above.

The training communication time is the time taken by the server to send the message s_msg_{train} with the dummy model (message size of 1GB) and receive back the same model, c_msg_{train}, (message size of 1GB). The test communication time is the time taken by the server to send the message s_msg_{aggreg} with the model (message size of 1GB) and receive back 10 float points, message c_msg_{test}, representing the possible ML metrics that clients compute (translated to 1.8KB). See Figure 1.

TABLE V
COMMUNICATION TIMES BETWEEN EACH PAIR OF REGIONS. THE TRAINING PHASE EXCHANGES A TOTAL OF 2GB IN MESSAGES AND THE TEST PHASE EXCHANGES A LITTLE MORE THAN 1GB IN TOTAL

Pair of regions	Comm. times (s)		Sl
	Training	Test	
us-east-1 (AWS) & *us-east-1* (AWS)	6.68	3.59	1.00
us-east-1 (AWS) & *us-west-2* (AWS)	39.67	20.30	5.84
us-east-1 (AWS) & *us-central1* (GCP)	22.83	12.07	3.40
us-east-1 (AWS) & *us-west1* (GCP)	33.02	16.10	4.78
us-west-2 (AWS) & *us-west-2* (AWS)	6.56	3.41	0.97
us-west-2 (AWS) & *us-central1* (GCP)	33.25	14.53	4.65
us-west-2 (AWS) & *us-west1* (GCP)	20.42	10.83	3.04
us-central1 (GCP) & *us-central1* (GCP)	2.30	1.21	0.34
us-central1 (GCP) & *us-west1* (GCP)	7.35	3.86	1.09
us-west1 (GCP) & *us-west1* (GCP)	4.09	2.30	0.62

B. Analysis of the Scalability of the Proposed Mathematical Formulation

We used *Gurobi Optimizer* [28], a state-of-the-art solver for mathematical programming models, for solving the proposed formulation. In order to analyze its scalability, the number of clients varied from 2 to 50. Furthermore, aiming at increasing the search space, for each VM of Table II, we have created five synthetic VMs. Thus, we have a total of 78 VMs, 54 with GPUs and 24 without any GPU.

For the sake of making the problem solution feasible, we set the GPUs and vCPUs limits of all regions and all providers to the infinite constant of Python's math library. We also considered a deadline and a budget of 10,000 seconds and $30000 per FL round respectively. Figure 2 presents the relation between the number of clients and Gurobi's execution time.

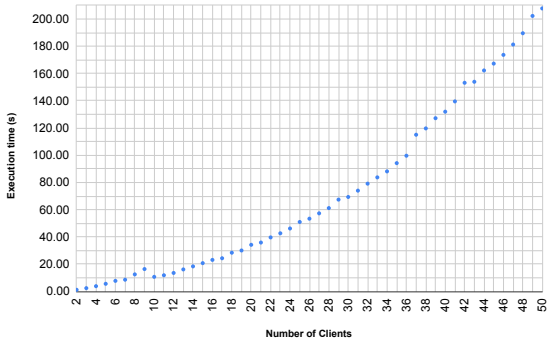

Fig. 2. Relation between number of clients and Gurobi's execution time.

We observe in Figure 2 that our mathematical formulation is robust and present the optimal solution, in a realistic time, with a large search space and a considerable number of clients.

C. Theoretical Analysis of the Scheduling Results against User Random Selection

In this section, we theoretically analyze the optimal setup results obtained with our model, comparing them with user random selection approaches. In order to obtain a solution that offers a balance between the execution time and financial cost, we set $\alpha = 0.5$.

Let's consider 50 homogeneous clients with 948 training samples and 522 test samples. As all clients datasets have the same size, we considered the same execution time as the baseline execution time for all the 50 clients. Therefore, for those clients that access their dataset in AWS (resp., GCP), the baseline training time ($train\_bl_i$) is 412.94 (resp., 183.53) seconds, and the baseline test time ($test\_bl_i$) is 182.77 (resp., 49.47) seconds. Regarding communication, messages of the server as well as the training message of clients have 0.54GB of size and the test message from clients has 1.81KB of size. The total value for the communication baseline time ($train\_comm\_bl + test\_comm\_bl$) is 27.26 seconds.

We consider two data placement scenarios: *GCP(50)* where all datasets are stored in GCP Cloud Storage and *GCP(25)-AWS(25)* where datasets of 25 clients are stored in GCP Cloud Storage and the other 25 in AWS S3. For each scenario, the respective optimal setup results of our model are compared to a given user selection approach, also described in the following. Note that all values are computed from the slowdowns presented in the previous sections.

- *GCP(50) scenario:* Our model proposes a optimal setup where the 50 clients and the server should be in the Iowa region of GCP (*us-central1* region), with all clients in different *n1-standard-8* VMs with a Volta GPU in each (vm_{213}) and the server in a *e2-standard-4* VM (vm_{214}). The vm_{213} is the most expensive VM in GCP but has the smallest execution time. On the other hand, we considered a random approach where all clients and the server are placed in the Oregon region of GCP (*us-west1* region), using the same VM types, assigning vm_{222} to clients and vm_{223} to the server. Compared to this approach, the optimal one reduces in 53.70% the execution time and in 25.92% the costs.

- *GCP(25)-AWS(25) scenario:* Instead of placing the clients near the datasets in each cloud provider, our mathematical model proposes to place all the 50 clients and the server in the Oregon region of AWS (*us-west-2*): each client in different *g4dn.2xlarge* VM (vm_{121}), which is the fastest VM in all regions of AWS, and the server in a *t2.xlarge* VM (vm_{123}). On the other hand, we consider a random approach where all clients are placed near the datasets in the fastest VMs of the chosen region and the server in one of the cheapest VMs. Thus, for such a configuration, we used the *g4dn.2xlarge* VM in the N. Virginia region (vm_{111}) for the 25 clients with dataset in AWS, the *n1-standard-8* VM with a Volta GPU in the Iowa region (vm_{213}) for the ones with dataset in GCP, and the *e-standard-4* VM in the Iowa region (vm_{214}) to the server. In this case, the FL round in the optimal setup reduces in 10.47% the execution time and in 48.34% the costs.

Table VI summarizes these two scenarios. It presents the location of the datasets (Scenario), the optimal setup, the execution time and costs computed by the mathematical formulation, along with the random approaches described above, and the difference between the latter and the optimal setup.

TABLE VI
THEORETICAL RESULTS OF A SINGLE FL ROUND AND 50 CLIENTS

Scenario	Optimal selection (model)			User random selection			Difference (%)	
	Setup	Exec. time	Costs ($)	Setup	Exec. time	Costs ($)	Exec. time	Costs
GCP(50)	clients in vm_{213} server in vm_{214}	0:01:45	13.84	clients in vm_{222} server in vm_{223}	0:03:47	18.69	53.70	25.92
GCP(25)-AWS(25)	clients in vm_{121} server in vm_{123}	0:10:16	13.72	half clients in vm_{111} and half in vm_{213}, server in vm_{214}	0:11:29	26.56	10.47	48.34

This difference is computed by $\frac{v_r - v_o}{v_r}$, where v_r is the random approach value and v_o is the optimal one.

D. Analysis of the Scheduling Results in a Multi-cloud Platform

For each of the current experiments, we have had to respect the maximum number of global and per region vCPUs and GPUs that a user can allocate in each cloud provider. In GCP, vCPUs (both global and per region one) and GPUs are respectively limited to 40 and 4. In AWS, the limit of the N. Virginia (*us-east-1*) region vCPUs is 52 and of the Oregon region (*us-west-2*) is 36. On the other hand, there is no restriction for the number of global vCPU and the GPU limit is included in the vCPU one. Thus, we have kept the infinity constant for these limits in AWS. We use the same input data (baseline execution and communication time) as the previous experiment for all clients $c \in C$.

We have reproduced the best FL scenario presented in [27] which consists of four clients with equally divided datasets, that execute 10 FL rounds with 5 local epochs each. Each client has 948 training samples and 522 test samples, with 10% of TIL-positive in each dataset. We have then considered three possible dataset placement scenarios: AWS(4), GCP(4) and AWS(2)-GCP(2). In AWS(4), all datasets are stored in AWS, while in GCP(4) they are placed in GCP. In the last scenario, AWS(2)-GCP(2), half of the datasets is stored in AWS, while the other half is stored in GCP.

Our model uses only three different VMs to place the clients in each of the above three scenarios. For sake of evaluation comparison, for each scenario, we also created two user random selection assignments. In the case of AWS(4) (resp., GCP(4)) scenarios, the first assignment allocates clients and the server in the cheapest VMs (with GPU, in case of clients) within the same cloud region in AWS (resp., GCP) where is located the dataset of the corresponding client, while the second assignment has a similar allocation but on the other cloud provider, i.e., GCP (resp., AWS). For the GCP(2)-AWS(2) scenario, the two user selection assignments consider that clients are allocated in cheapest VMs with GPU in the same region of their datasets but the server changes position, being in AWS in the first setup and in GCP in the second. All the scheduling assignments are described in Table VII, and the configuration of each VM can be found in Table II.

We first present the execution time and the cost for a single round of FL to all setups in Table VIII. The optimal values come from the mathematical formulation and the random scheduling setups are computed using the slowdowns from the previous subsections. We also present how much the optimal setup gains in terms of percentage from the random scheduling schemes, computed by $\frac{v_r - v_o}{v_r}$, where v_r is the user random selection approach value and v_o is the optimal one. Note that a negative percentage value means that the optimal value is bigger than the random one.

We can observe from Table VIII that our mathematical model presents better results in all scenarios with an average execution time reduction of 20.03% compared to the randomly selected scenarios Regarding the monetary costs, the average difference is -3.97%. This negative difference comes from scenario GCP(4), where the optimal setup is more expensive than both random scenarios (by 18.05% and 37.88%), but with a higher reduction in the execution time (47.69% and 58.81%).

The above comparisons show that placing the clients (and server) as close as possible to the dataset does not always provide the best execution time. For example, in the scenario AWS(4), where all datasets are in the N. Virginia region of AWS and our mathematical formulation places clients and the server in the Oregon region of AWS, the single FL round has presented a small reduction in both execution time and total costs when compared to the first user selection approach, where all allocated VMs are in the N. Virginia region. We should point out that a FL application usually executes for several rounds, which increases the absolute difference in time and costs between the optimal setup and the random ones.

Results from scenario AWS(2)-GCP(2) show that the misplacement of the server can increase both execution time and costs. Although the execution time of all clients is the same in both random setups, the communication time between the slowest client and the server varies. The clients whose datasets are in AWS take longer to execute than the other two, and communication time inside the same AWS region is much lower than between both providers. Moreover, the transfer costs change according to the server placement. If it is on AWS, the cost is less ($0.09 per GB) than when it is in GCP ($0.12 per GB). Therefore, the difference in execution time and costs between the two random setups for this scenario can be explained.

Finally, we did a real deployment and executed all setups in both cloud providers (AWS and GCP) with 10 FL epochs. Table IX summarizes the obtained results, showing that the optimal solution allows an average execution time reduction of 21.07%, with an average cost increase of only 4.30%.

We can observe in the table that the computed differences

TABLE VII
VMs SETUP FOR OPTIMAL AND RANDOM SCHEDULING SCHEMES FOR ALL SCENARIOS WITH 4 CLIENTS

Scenarios	Optimal selection (model)	1^{st} user random selection	2^{nd} user random selection
AWS(4)	c_1 in vm_{121}, c_2 in vm_{121}, c_3 in vm_{121}, c_4 in vm_{121}, s in vm_{123}	c_1 in vm_{111}, c_2 in vm_{111}, c_3 in vm_{111}, c_4 in vm_{111}, s in vm_{113}	c_1 in vm_{211}, c_2 in vm_{211}, c_3 in vm_{211}, c_4 in vm_{211}, s in vm_{214}
GCP(4)	c_1 in vm_{213}, c_2 in vm_{213}, c_3 in vm_{213}, c_4 in vm_{213}, s in vm_{214}	c_1 in vm_{211}, c_2 in vm_{211}, c_3 in vm_{211}, c_4 in vm_{211}, s in vm_{214}	c_1 in vm_{111}, c_2 in vm_{111}, c_3 in vm_{111}, c_4 in vm_{111}, s in vm_{113}
AWS(2)-GCP(2)	c_1 in vm_{121}, c_2 in vm_{121}, c_3 in vm_{121}, c_4 in vm_{111}, s in vm_{123}	c_1 in vm_{111}, c_2 in vm_{111}, c_3 in vm_{211}, c_4 in vm_{211}, s in vm_{113}	c_1 in vm_{111}, c_2 in vm_{111}, c_3 in vm_{211}, c_4 in vm_{211}, s in vm_{214}

TABLE VIII
THEORETICAL RESULTS WITH SINGLE FL ROUND

Scenarios	Optimal selection (model)		User random selection			Difference (%)	
	Exec. time	Costs ($)	#	Exec. time	Costs ($)	Exec. time	Costs
AWS(4)	0:10:16	1.13	1^{st}	0:10:23	1.13	1.09	0.53
			2^{nd}	0:10:23	1.30	1.05	13.44
GCP(4)	0:01:47	1.12	1^{st}	0:03:25	0.95	47.69	-18.05
			2^{nd}	0:04:21	0.81	58.81	-37.88
AWS(2)-GCP(2)	0:10:16	1.13	1^{st}	0:10:23	1.16	1.09	2.64
			2^{nd}	0:11:29	1.33	10.47	15.51

TABLE IX
REAL CLOUD EXECUTION WITH 10 FL ROUNDS

Scenarios	Optimal selection (model)		User random selection			Difference (%)	
	Exec. time	Costs ($)	#	Exec. time	Costs ($)	Exec. time	Costs
AWS(4)	1:31:18	10.66	1^{st}	1:35:14	10.87	4.12	1.92
			2^{nd}	1:55:03	13.59	20.64	21.56
GCP(4)	0:22:00	11.98	1^{st}	0:36:54	9.61	40.38	-24.61
			2^{nd}	0:51:22	8.53	57.18	-40.34
AWS(2)-GCP(2)	1:36:09	10.92	1^{st}	1:37:37	11.25	1.51	2.93
			2^{nd}	1:38:42	12.51	2.59	12.71

between the optimal values and the random ones are close to the theoretical results (Table VIII) but not equal. Besides, the real cloud differences in the second random setup of scenario AWS(4) are far from the theoretical ones. In [29], Leitner *et al.* have extensively evaluated the performance of different cloud platforms, showing that the performance of IO-bound applications in AWS varies a lot even in the same VM. Hence, since our application transfers data from memory to the GPU at least two times per round, the variation of IO performance has a negative impact on execution times, which explains the difference found between Table VIII and Table IX.

E. Discussion about the multi-cloud environment and results

Firstly, concerning the AWS-GCP environment of our experiments, all AWS EC2 instances were almost always available for deployment, although, they frequently presented performance variation. Particularly, the Volta GPU instance had poor unexpected performance, which led us to remove it from the model's search space. On the contrary, GCP instances presented a stable performance. We also observed that the time taken to access datasets allocated in AWS S3 (Table III) was higher than the time to access the corresponding datasets in the GCP Cloud Storage (Table IV). Such behavior is coherent with other ones from the related literature. In [29] Leitner *et al.* present extensive experiments regarding the performance and predictability of different VM instance types in many cloud providers. The authors concluded that, in general, AWS has worse performance and is more unpredictable than GCP.

We also observe that the α parameter of our model has a low impact. The results with α equal to 0.5, 0, or 1 produced the same execution times and costs, for most of the experiments.

Our theoretical analysis shows that the execution time given by the proposed model is reduced by up to 58.81% when compared to random scenarios with four FL clients in the cloud whose datasets are stored in GCP (scenario GCP(4)). Considering 50 clients, with half of them storing their dataset in AWS and the other half in GCP (scenario AWS(25)-GCP(25)), the monetary costs and execution time are reduced by up to 48.34% and 10.47% respectively.

In the experiments conducted in the AWS-GCP platform, scenario GCP(4) yielded a reduction of 40.38% on the execution time when placing clients in the same region of the dataset in the most expensive VM type with a monetary cost increase of 24.61%.

When comparing, for a single FL application round, the theoretical execution times and the ones obtained in the AWS-GCP platform, we observe a difference of 11.14% in scenario AWS(4), -22.96% in scenario GCP(4), and 6.42% in scenario AWS(2)-GCP(2). Although in scenario GCP(4), such a difference in percentage is higher than the other two,

the absolute value is smaller: only 25 seconds of difference compared to 1 minute and 10 seconds in AWS(4) and 40 seconds in AWS(2)-GCP(2). Moreover, Ward and Barker [30] have shown in 2014 that the same VM type could vary its performance by up to 29%. The authors associated this variation with the oversold physical machine underneath the VMs and other multi-tenanted phenomena. In our experiments, we observe that the variation among the same VM type is smaller nowadays, but still present.

VII. Conclusion and Future Work

This paper has presented a mathematical formulation for a Cross-Silo Federated Learning task scheduling problem in a multi-cloud scenario, aiming at minimizing its execution time and monetary cost. This formulation is part of a framework proposal to execute Federated Learning applications in a multi-cloud scenario. Theoretical results show that the proposed model is robust and scales with the growth of the number of clients and available virtual machines.

Furthermore, in a real scenario using Amazon-GCP multi-cloud platform with 4 clients and datasets stored in AWS S3 and GCP Cloud Storage, the optimal setup offered an improvement by up to 57.18% in the execution time and up to 21.56% in the monetary costs when compared to random selection approach.

In future work, we aim introducing heterogeneity in clients' datasets and models, to evaluate the proposed approach in terms of performance and quality results. Besides, we intend to use Spot instances, which are offered with a huge discount by cloud providers, but can be revoked. In this case, upon the revocation, our framework will need to to reschedule the interrupted tasks dynamically through a dynamic scheduler module. Moreover, we are currently working on a fault-tolerant module in our framework not to loose all computation when a revocation occurs.

Acknowledgment

This research is supported by *Programa Institucional de Internacionalização* (PrInt) from CAPES (process number 88887.310261/2018-00), by CNPq (process number 145088/2019-7), by the *Coordenação de Aperfeiçoamento de Pessoal de Nível Superior* (CAPES - Finance Code 001), by Project Universal/CNPq n^o 404087/2021-3 and CNE/FAPERJ n^o E-26/201.012/2022(271103).

References

[1] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," in *Proc. of the 20th Inter. Conf. on Artificial Intelligence and Statistics*, vol. 54, 2017, pp. 1273–1282.

[2] S. Shen, T. Zhu, D. Wu, W. Wang, and W. Zhou, "From distributed machine learning to federated learning: In the view of data privacy and security," *Concurrency and Computation: Practice and Experience*, 2020.

[3] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated Machine Learning: Concept and Applications," *ACM Trans. Intell. Syst. Technol.*, vol. 10, no. 2, Jan. 2019. [Online]. Available: https://doi.org/10.1145/3298981

[4] S. Rajendran *et al.*, "Cloud-Based Federated Learning Implementation Across Medical Centers," *JCO Clinical Cancer Informatics*, no. 5, 2021. [Online]. Available: https://doi.org/10.1200/CCI.20.00060

[5] R. L. Villars, C. W. Olofson, and M. Eastwood, "Big data: What it is and why you should care," *White paper, IDC*, vol. 14, pp. 1–14, 2011.

[6] H. Liu, "Big data drives cloud adoption in enterprise," *IEEE Internet Computing*, vol. 17, no. 4, pp. 68–71, 2013.

[7] N. Leavitt, "Storage challenge: Where will all that big data go?" *Computer*, vol. 46, no. 09, pp. 22–25, 2013.

[8] A. W. Services, "Region and Zones - Amazon Elastic Compute Cloud," https://aws.amazon.com/about-aws/global-infrastructure/, accessed 15 Feb. 2022.

[9] P. Google Cloud, "Geography and regions - Documentation," https://cloud.google.com/about/locations, 2022, accessed 15 Feb. 2022.

[10] J. Ren, J. Sun, H. Tian, W. Ni, G. Nie, and Y. Wang, "Joint resource allocation for efficient federated learning in internet of things supported by edge computing," in *2021 IEEE Inter. Conf. on Communications Workshops (ICC Workshops)*, 2021.

[11] B. Buyukates and S. Ulukus, "Timely communication in federated learning," in *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2021.

[12] Y. Huang, L. Chu, Z. Zhou, L. Wang, J. Liu, J. Pei, and Y. Zhang, "Personalized Cross-Silo Federated Learning on Non-IID Data," *Proc. of the AAAI Conf. on Artificial Intelligence*, vol. 35, no. 9, May 2021.

[13] Q. Li, Y. Diao, Q. Chen, and B. He, "Federated learning on non-iid data silos: An experimental study," *ArXiv*, vol. abs/2102.02079, 2021.

[14] Q. Zhang, R. Zhou, C. Wu, L. Jiao, and Z. Li, "Online scheduling of heterogeneous distributed machine learning jobs," in *Proc. of the Twenty-First Inter. Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing*, 2020.

[15] L. Liu, H. Yu, G. Sun, H. Zhou, Z. Li, and S. Luo, "Online job scheduling for distributed machine learning in optical circuit switch networks," *Knowledge-Based Systems*, vol. 201-202, 2020.

[16] M. Yu, J. Liu, C. Wu, B. Ji, and E. Bentley, "Toward efficient online scheduling for distributed machine learning systems," *IEEE Transactions on Network Science and Engineering*, 2021.

[17] M. Mohammadi Amiri and D. Gündüz, "Computation scheduling for distributed machine learning with straggling workers," *IEEE Transactions on Signal Processing*, vol. 67, no. 24, 2019.

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

[19] J. Saltz *et al.*, "Spatial Organization And Molecular Correlation Of Tumor-Infiltrating Lymphocytes Using Deep Learning On Pathology Images," *Cell reports*, vol. 23, no. 1, 2018.

[20] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd Int. Conf. on Learning Representations, ICLR 2015*, 2015.

[21] M. Abadi *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. [Online]. Available: https://www.tensorflow.org/

[22] D. J. Beutel, T. Topal, A. Mathur, X. Qiu, T. Parcollet, and N. Lane, "Flower: A friendly federated learning research framework," *ArXiv*, vol. abs/2007.14390, 2020.

[23] A. W. Services, "Amazon S3," https://aws.amazon.com/s3/, 2021, accessed 19 Dec. 2021.

[24] P. Google Cloud, "Cloud Storage," https://cloud.google.com/storage, 2021, accessed 19 Dec. 2021.

[25] E. M. Malta, S. Avila, and E. Borin, "Exploring the Cost-Benefit of AWS EC2 GPU Instances for Deep Learning Applications," in *Proc. of the 12th IEEE/ACM Inter. Conf. on Utility and Cloud Computing*, 2019.

[26] R. Brum, L. Drummond, M. C. Castro, and G. Teodoro, "Towards optimizing computational costs of federated learning in clouds," in *2021 Inter. Symposium on Computer Architecture and High Performance Computing Workshops (SBAC-PADW)*, 2021.

[27] R. Brum, G. Teodoro, L. Drummond, L. Arantes, M. Castro, and P. Sens, "Evaluating federated learning scenarios in a tumor classification application," in *Anais da VII Escola Regional de Alto Desempenho do Rio de Janeiro*. SBC, 2021.

[28] Gurobi. (2022) Gurobi optimizer. [Online]. Available: https://www.gurobi.com/products/gurobi-optimizer/

[29] P. Leitner and J. Cito, "Patterns in the chaos—a study of performance variation and predictability in public iaas clouds," *ACM Trans. Internet Technol.*, vol. 16, no. 3, apr 2016.

[30] J. S. Ward and A. Barker, "Observing the clouds: a survey and taxonomy of cloud monitoring," *Journal of Cloud Computing*, vol. 3, no. 1, Dec 2014.