2022 International Symposium on Computer Architecture and High Performance Computing Workshops (SBAC-PADW)

Towards a Federated Learning Framework on a Multi-Cloud Environment

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 Clícia 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*—This paper proposes Multi-FedLS, a Cross-silo Federated Learning (FL) framework for a multi-cloud environment aiming at minimizing financial cost as well as execution time. It comprises four modules: Pre-Scheduling, Initial Mapping, Fault Tolerance, and Dynamic Scheduler. Given an application and a multi-cloud environment, the Pre-Scheduling module runs experiments to obtain the expected execution times of the FL tasks and communication delays. The Initial Mapping module receives these computed values and provides a scheduling map for the server and clients' VMs. Finally, Multi-FedLS deploys the selected VMs, starts the FL application, and monitors it. The Fault Tolerance (FT) module includes fault tolerance strategies in the FL application, such as checkpoint and replication, and detects some anomalous behaviors. In case of an unexpected increase in the communication delay or a VM failure, the FT module triggers the Dynamic Scheduler Module in order to select a new VM and resume the concerned tasks of the FL application. Some preliminary experiments are presented, confirming that some proposed strategies are crucial to efficiently execute an FL application on a multi-cloud environment.**

***Index Terms*—Federated Learning Framework, Multi-Cloud Environment, Efficient Resource Allocation**

I. Introduction

Federated Learning (FL) is a recent type of distributed Machine Learning (ML) in which the participating clients do not share their private data [1]. The clients federation solves the learning task by having a coordinated central server that keeps the current global model. Each client trains its local model on its local data set and communicates only the model weights to the server, updating then the global model.

The server-clients architecture of FL, also called Model-Centric Federated Learning [2], is classified into Cross-Device or Cross-Silo Federated Learning, depending on the client's type. In Cross-Device Federated Learning, clients are low-powered devices, like mobile phones [3] or IoT devices [4] while in Cross-Silo Federated Learning, clients are companies or institutions (*e.g.*, hospitals [5]) with similar datasets willing to create a central model. In this second type of FL, the central server can assume that all clients are available during the whole execution, as they are powerful machines or even clusters.

Nowadays, training ML algorithms uses a huge amount of generated data. Data are often stored using cloud storage services [6] whose performance can vary. Besides the latter, cloud providers also offer Virtual Machines (VMs) with different accelerators in a service generically called Infrastructure-as-a-Service (IaaS). For example, the Amazon Web Services (AWS) offers GPUs attached to pre-defined VMs types, whose architectures vary from Kepler to Ampere [7], while Google Cloud Platform (GCP) allows the user to attach GPUs or TPUs to a pre-defined or custom instance type [8], [9].

Thus, in a multi-cloud environment, an FL framework should choose the cloud provider, VM types and regions, and the minimum required network bandwidth for each client and server. A wrong choice can result in a considerable increase in the execution cost of an FL application [10], [11]. In other words, the allocation of cloud resources is critical for an efficient execution of FL applications in clouds. Therefore, a multi-cloud environment should cope with the problem of configuring the clouds, FL execution monitoring, and selection of a new VM whenever one of the current allocated ones becomes unavailable.

Although there is a vast literature on scheduling and resource management of FL jobs [12]–[16], most of them focus on Cross-Device Federated Learning scenarios, where low-powered resources frequently become offline during the

This research is supported by Project Universal/CNPq n^o 404087/2021-3 and CNE/FAPERJ n^o E-26/201.012/2022(271103)

978-1-6654-5157-4/22/$31.00 ©2022 IEEE
DOI 10.1109/SBAC-PADW56527.2022.00016

training process. The scheduling and resource provisioning problems of Cross-Silo FL applications can be related to the scheduling of distributed Machine Learning (ML) applications [17]–[19] as there is the assumption that the workers are always available. However, there are two main differences between scheduling distributed ML and FL: (i) in ML, the scheduler assumes that all workers have the same amount of data, and (ii) data can be shared among workers.

In this work, we propose Multi-FedLS, an architecture of a Cross-silo FL framework. The framework comprises four modules: Pre-Scheduling, Initial Mapping, Fault Tolerance, and Dynamic Scheduler. The first module collects information about the FL application and the cloud environment when there is no previous knowledge about them, such as the expected execution time of the FL tasks and communication delays. Then, the Initial Mapping receives these computed values and provides, through a mathematical formulation, a scheduling map of the server and clients, aiming at minimizing financial costs and execution times. Finally, Multi-FedLS deploys the selected VMs, starts the FL application, and monitors it. The Fault Tolerance module is responsible for including fault tolerance strategies in the FL application, such as checkpoint and replication. In case of an unexpected increase in the communication delay or a VM failure, that module triggers the Dynamic Scheduler Module to select a new VM and resume the concerned tasks of the FL application.

The remaining of this paper is organized as follows. Section II presents the main available tools for federated learning. The proposed Multi-FedLS framework, introduced in Section III, is a Flower-based framework to execute Federated Learning Applications on Multi-clouds. Section IV describes the Pre-Scheduling components and presents some preliminary experiments. Finally, Section V, concludes the paper and present some future research directions.

II. Available Tools for Federated Learning

There are several tools for executing Federated Learning applications in the literature. TensorFlow Federated (TFF) [20] is a Google's library to execute Federated Learning applications in a simulation environment based on the well-known TensorFlow (TF) API. TFF gives the user two API layers: Federated Learning (TFF-FL) API for executing basic FL algorithms and Federated Core (TFF-FC) API for implementing new FL algorithms. The authors claim that TFF can simulate the FL environment in several machines with multiple GPUs[1]. However, we could not find any explanation on how to set up this multi-machine simulation in TFF tutorials[2].

PySyft and PyGrid are two components of a tool focused on Data-Centric Federated Learning [21], in which users compute data they cannot see. In this type of FL, the actors are Data Scientists, which do not have any data, while the Data Owner holds all data and receives computational requests from the Scientists.

Burlachenko *et al.* proposed FL_PyTorch, an FL simulation tool built on top of the PyTorch API [22], which helps researchers to develop new FL aggregation techniques. However, such a tool is focused only on simulation.

Federated AI Technology Enabler (FATE) [23] provides a framework to support not only the FL training step, but the whole process: data pre-processing, training, and inference steps. FATE has several modules to handle different features of the environment for: (i) describing FL algorithms, data structures, and communication channels; (ii) post-training inference and managing the whole FL pipeline; and (iii) helping users to explore the trained models through visualization tools.

Flower [24] is an open-source FL framework that deals with heterogeneous clients in simulated or real-world scenarios, allowing the use of any ML framework underneath it (TensorFlow, PyTorch, or a custom one). Besides, Flower can execute in several environments with different operating systems and hardware settings. Researchers can expand Flower's code in order to implement new FL algorithms. Due to its modular structure, the final user only needs to implement a few functions for transforming a regular ML application into a federated one.

We have adopted Flower in our proposal because of its simple architecture and the possibility of execution in real and heterogeneous environments.

III. Multi-FedLS: A Framework based on Flower to execute Federated Learning Applications on Multi-clouds

An FL application contains two types of tasks: server and clients. The clients train their local model with their private data while the server coordinates the learning by aggregating the clients' model weights.

A communication round is the processing unit of an FL application and it is composed by five steps: (i) the server sends the initial weights to clients; (ii) the clients train their local model with their training dataset and send the updated weights back to the server; (iii) the server receives all updates, aggregates them and sends back to the clients the new weights; (iv) the latter update their local weights with the received ones, evaluate the model with their test dataset and send the evaluation metrics to the server; and (v) the server receives all evaluation metrics, aggregates them and starts the next communication round. Note that there are two communication barriers in a single round. Thus, due to those synchronization barriers, any increase in the communication time has an impact in the total FL execution time. It is worth remarking that the clients' datasets may be spread in several storage systems of different cloud providers.

The proposed Multi-FedLS framework is composed by four modules: the Pre-scheduling, the Initial Mapping, the Fault Tolerance, and the Dynamic Scheduler modules.

Figure 1 shows the proposed architecture for the Multi-FedLS framework.

[1]https://www.tensorflow.org/federated/tutorials/simulations_with_accelerators, last accessed in January 2022

[2]https://www.tensorflow.org/federated/tutorials/simulations, last accessed in January 2022

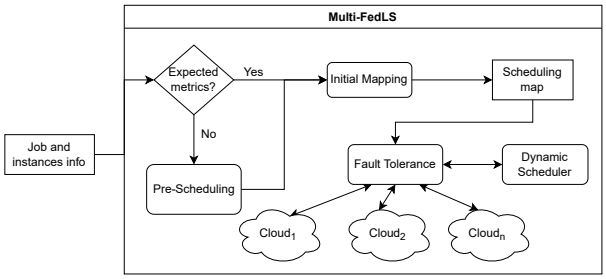

Fig. 1: Architecture of Multi-FedLS

A. Pre-Scheduling Module

The Multi-FedLS framework requires information from FL applications, the available VMs, and cloud providers to define the best mapping of clients and server to VMs in the multi-cloud environment. Some types of information, such as the number of clients, number of communication rounds, number of training epochs, location of each client's dataset, and VM prices, are easily obtained. On the other hand, many others, such as the communication delay between cloud regions or the execution time of clients in different VMs, require the execution of experiments to be known. We also point out that, due to the communication barriers, any increasing in the communication time has an impact in the total FL execution time. Consequently, it is crucial to evaluate the time spent sending messages between different cloud regions to define the best VMs locations for the clients and server processes in the multi-cloud environment. Also, the execution time of FL clients and server may vary according to the types and regions of the allocated VMs. Hence, such an information is also necessary when selecting the most suitable VMs for the FL application.

Thus, using dummy applications, the Pre-Scheduling module is responsible for executing some tests for obtaining all the above discussed values.

B. Initial Mapping

Receiving as input (i) the datasets locations, (ii) the machine learning procedures, and (iii) data provided by the Pre-Scheduling module, the Initial Mapping module generates an allocation map of the clients and server for a multi-cloud environment, aiming at minimizing both the total execution time and financial costs of the FL application. The mapping problem can be solved using either a mathematical formula or, in case the execution time is prohibitive, a heuristic.

Public cloud providers offer computing resources through a plethora of Virtual Machine (VM) instances with different capacities. They provide these instances under several contract models that differ according to the availability guarantees and prices. Under the classic *On-demand* pricing model, a user pays for VM instances per hour or second without any long-term commitment or upfront payment. In the other hand, many providers also offer their spare unused computing capacity, as *Spot VM Instances*, at significant discounts comparing to their standard *On-demand* prices. However, such price advantages come with the caveat of having no guarantees of VM availability. In order to reduce costs, Multi-FedLS can use Spot VMs, that, although cheaper than On-demand ones, can be revoked at any time by the cloud provider. Consequently, both a monitoring tool and a dynamic scheduler are necessary to reallocate the clients and server in case of such revocations.

C. Tolerating Faults and Dynamic Scheduler

Usually, Federated Learning tools can deal with clients' revocations along the execution. When Flower detects a client failure, the server ignores the faulty client by aggregating only the results from the other ones. Furthermore, it is possible to define a minimum number of available clients to start a new communication round. Since in cross-silo FL there are few clients, and the lost of one of them can compromise the learning outcomes, in our proposal, when a client is revoked, it is re-started from a checkpoint in a new VM, or in some cases, it is replicated to avoid new VM deploying and recovery overheads. The selection of a new VM should consider the characteristics of the used cloud providers. For example, if the Amazon Web Services stores a dataset only in one region, a VM belonging to this region is more suitable for executing the corresponding client than VMs from others regions.

Most FL tools do not handle faults on the server side. In Flower, when the server of an application fails, all clients finish their executions with an error. Thus, Multi-FedLS needs to handle possible server revocations by implementing fault-tolerant techniques on the server side. Although Flower does not have an automatic way to save the model updates on the server side, after each communication round, it allows the users to modify the server code to implement checkpointing [3]. In this case, Multi-FedLS has to deploy another VM to execute the server, considering all clients' positions, the communication times, and costs.

IV. Pre-Scheduling Components and Preliminary Experiments

In this section, we describe in more details the Pre-Scheduling module and introduce some results for a case study application. The module receives input information about the FL application and the environment. Concerning the application, it requires the number of clients, the datasets' location, the number of communication rounds, and the number of epochs trained in each round. Regarding the multi-cloud environment, it needs information about the cloud regions and the possible VMs types that can execute the FL tasks.

The Pre-Scheduling module automatically gets the following data of the multi-cloud environment: (i) regions of each cloud provider, (ii) number of Availability Zones (AZs, described later) at each region, and (iii) characteristics of the offered VMs, *e.g.* vCPUs, memory, GPU type, and GPU memory. Then, this module executes tests to compute the communication delay among all cloud regions and the expected execution time of each client at every instance type.

[3] https://flower.dev/docs/saving-progress.html

For the experiments, we have chosen a Federated Learning application for a Tumor Infiltrating Lymphocytes (TIL) classifier, described in [25], [26], and the Amazon Web Services (AWS) and Google Cloud Provider (GCP), as cloud providers.

A. Communication delay between cloud regions

Cloud providers divide their physical infrastructure into cloud regions, which are independent and isolated geographic areas [27], [28]. Those regions are also divided as isolated failure points, called Availability Zones (AZs), to improve the reliability of each region. If a user deploys VMs in two AZs of the same region and one of them fails, the other one may continue executing. An AZ is one or more data centers inside a building, being a region a collection of AZs inside a small geographical area (usually within one city). In February 2022, GCP had three or four zones at each region, and AWS contained three to six availability zones at each cloud region. Cloud providers identify each AZ by a letter at the end of the region's name, normally one between *a* and *f*.

In our experiments, the Pre-Scheduling module executes a dummy application with one server and single client to measure the message exchange time between them. That application used a floating point vector with the size of the VGG16 CNN model [29], which has more than 130 millions weights, to represent the FL model. The training message time is the time taken by the server to send the vector and receive it back, while the test message time is the time taken by the server to send the vector and receive a dictionary with 8 dummy keys and values, representing the possible machine learning metrics. In the presented experiments, the following regions and cloud providers were considered: *us-east-1* in AWS and *us-central1* (GCP) in GCP.

Tables I and II show the average times of 10 executions of the dummy application in several scenarios. The execution times were obtained in different days and times of the week during a period of one month.

Observing the main diagonal in both Tables I and II, we can see that all AZs in AWS have more homogeneous communication times than the GCP ones. We can also observe that GCP is overall faster than AWS, but the latter is more stable. We can confirm such differences by measuring the average time to exchange a training (or test) message, along with its standard deviation, within a cloud region. The average value and standard deviation of the message exchange in the training message (Table I) among all AZs from each region, in the AWS *us-east-1* (resp., GCP *us-central1*) is 30.73 (resp., 16.32) seconds with a standard deviation of 3.09% (resp., 18.38%). Moreover, the average time of a training message exchange among the cloud providers (AWS and GCP) is 145.33 seconds, with a standard deviation of 13.60%. The average values and standard deviation of the exchange time of a test message are analogous. In AWS region *us-east-1* (resp., GPC region *us-central1*) the average value is 16.48 (resp., 8.83) seconds with a standard deviation of 2.97% (resp., 18.91%). Finally, the average time for exchanging a test message between providers is 74.82 seconds with a standard deviation of 14.64%.

Note also that a misplaced task has a great impact on the total FL execution time. For example, if the scheduler allocates the FL server in the main AZ of the east region of AWS (*us-east-1a*) and the clients in the main AZ of the central region of GCP (*us-central1-a*), the communication time is delayed by 3.87 times.

B. Expected training and evaluation times

Running an application in all available GPU types of a multi-cloud environment to compute the expected training and evaluation times and financial costs is unrealistic. Malta *et al.* [30] proposed a simple equation to calculate the total execution time of a centralized Deep Learning training using only the first two epochs. They observed that all training epochs have similar execution times, except for the first one. We have verified similar behaviour in our previous experiments [25], [26]. Thus, the Pre-Scheduling module obtains only the times of the first two training and evaluation rounds.

The storage service's access time has also an impact in the total execution time because there are repetitive accesses to the dataset. In turn, the dataset cannot be moved, due to privacy issues. The region, where the VM is deployed, clearly influences the application execution time. In order to observe the impact of accessing the dataset from different regions, the pre-scheduling module obtains the execution times of the first two rounds in a scenario with four clients (with 948 training samples and 522 test samples each) using two different models: the Inception-ResNet v2 [31] and the VGG16 [29]. The Inception-ResNet (resp., VGG16) has 164 (resp., 16) layers with more than 54 (resp., 130) million adjustable weights.

In our experiments, we considered that the datasets were stored in both providers, using the same regions as the previous experiments, the *us-east-1* region of AWS and *us-central1* region of GCP as well as similar VMs to compare the different execution times. We chose the *g4dn.2xlarge* instance of AWS and the *n1-standard-8* instance of GCP. Both of them have eight vCPUs with 32 GB of RAM. The AWS' instance has an NVIDIA T4 GPU with 16 GB of memory and the same GPU model attached to the GCP's instance.

Tables III and IV respectively present the evaluated times for the Inception-ResNet v2 and VGG16 models. Column AWS represents the access time of a *g4dn.2xlarge* deployed in the *us-east-1a* AZ of AWS and column GCP represents the access time of a *n1-standard-8* instance deployed in the *us-central1-a* one. These tables also show the client ID (from 0 to 3), the round of the dummy application, and the step of an FL round (T for training and E for evaluate), each line represents.

We observe a small difference between VMs in AWS and GCP, when comparing the time taken to train and evaluate the datasets in the *us-east-1* region (DS in AWS). Sometimes GCP's VMs finished earlier than the AWS' VMs (*e.g.*, both training times of Client 1 in Table IV). On the other hand, we observed similar execution times when the dataset is in the *us-central1* region (in GCP). This preliminary experiment was conducted with only one instance type per cloud provider and GPU architecture. However, in AWS, there are more than

TABLE I: Required times to exchange a training message between Availability Zones of regions *us-east-1* (AWS) and *us-central1* (GCP). Times in seconds

		AWS *us-east-1*						GCP *us-central1*			
		1a	*1b*	*1c*	*1d*	*1e*	*1f*	*1-a*	*1-b*	*1-c*	*1-f*
AWS	*1a*	29.74±1.13	31.31±0.71	30.71±1.70	32.49±2.05	29.55±1.03	30.84±0.84	115.24±58.23	94.28±45.02	119.21±47.17	104.24±45.71
	1b	-	30.15±0.80	30.46±0.69	30.79±1.51	31.66±1.33	29.97±0.52	157.29±25.51	149.25±13.75	159.66±20.19	140.89±15.71
	1d	-	-	30.77±1.41	31.03±3.61	32.28±2.24	30.59±1.4	168.23±22.12	164.59±16.08	171.67±12.92	165.71±20.05
	1c	-	-	-	30.96±1.89	31.89±2.27	31.89±3.65	157.13±18.15	151.04±16.51	147.93±15.03	151.06±16.90
	1e	-	-	-	-	29.38±1.18	29.45±0.95	148.45±16.40	139.53±17.77	136.25±19.22	137.37±19.16
	1f	-	-	-	-	-	29.46±0.42	161.93±17.63	148.60±9.63	150.48±20.28	147.79±11.05
GCP	*1-a*	-	-	-	-	-	-	12.58±2.19	16.80±3.21	11.76±2.03	15.26±2.80
	1-b	-	-	-	-	-	-	-	20.49±1.54	17.61±2.79	17.59±2.02
	1-c	-	-	-	-	-	-	-	-	13.37±2.48	17.80±2.69
	1-f	-	-	-	-	-	-	-	-	-	19.97±0.94

TABLE II: Required times to exchange a test message between Availability Zones of regions *us-east-1* (AWS) and *us-central1* (GCP). Times in seconds

		AWS *us-east-1*						GCP *us-central1*			
		1a	*1b*	*1c*	*1d*	*1e*	*1f*	*1-a*	*1-b*	*1-c*	*1-f*
AWS	*1a*	15.92±0.59	16.61±0.56	16.43±0.94	17.41±0.92	15.96±0.36	16.53±0.43	54.73±27.20	44.47±19.35	61.72±27.19	57.10±30.56
	1b	-	16.33±0.80	16.48±0.35	16.29±0.54	16.79±0.80	16.39±0.46	77.02±10.93	72.72±13.48	82.52±16.95	74.25±14.61
	1d	-	-	16.68±1.67	16.13±0.74	17.26±1.42	16.22±0.63	85.55±32.48	83.43±25.42	90.66±27.30	86.51±26.30
	1c	-	-	-	17.01±1.74	16.95±0.86	17.23±2.04	84.45±19.70	80.26±17.63	76.58±14.73	77.55±17.90
	1e	-	-	-	-	15.8±0.58	15.83±0.48	77.42±19.03	72.98±19.13	68.85±19.6	71.06±18.81
	1f	-	-	-	-	-	15.92±0.19	84.32±20.26	75.87±17.21	77.01±19.37	78.73±16.90
GCP	*1-a*	-	-	-	-	-	-	6.73±1.29	8.96±1.66	6.26±1.15	8.21±1.59
	1-b	-	-	-	-	-	-	-	10.86±0.38	9.48±1.16	10.07±1.49
	1-c	-	-	-	-	-	-	-	-	7.15±1.43	9.68±1.37
	1-f	-	-	-	-	-	-	-	-	-	10.86±0.43

TABLE III: Times of the Inception-ResNet v2 model (four clients with 5 training epochs per comm. round)

Client	Round	Step	DS in AWS		DS in GCP	
			AWS	GCP	GCP	AWS
0	1^{st}	T	0:15:49	0:14:51	0:13:28	0:15:16
		E	0:01:04	0:01:09	0:00:41	0:00:52
	2^{nd}	T	0:05:23	0:06:09	0:03:02	0:03:37
		E	0:02:57	0:03:16	0:00:31	0:00:48
1	1^{st}	T	0:14:19	0:14:37	0:13:39	0:13:45
		E	0:01:14	0:01:18	0:00:42	0:00:51
	2^{nd}	T	0:05:00	0:06:05	0:03:06	0:03:13
		E	0:02:36	0:02:53	0:00:36	0:00:46
2	1^{st}	T	0:13:52	0:15:00	0:13:48	0:14:46
		E	0:01:21	0:01:23	0:00:41	0:00:48
	2^{nd}	T	0:05:48	0:06:05	0:03:17	0:03:33
		E	0:03:01	0:03:03	0:00:31	0:00:49
3	1^{st}	T	0:13:43	0:14:40	0:14:01	0:13:58
		E	0:01:05	0:01:45	0:00:42	0:00:50
	2^{nd}	T	0:05:00	0:06:26	0:02:57	0:03:16
		E	0:01:42	0:04:07	0:00:31	0:00:44

TABLE IV: Times of the VGG16 model (four clients with 5 training epochs per comm. round)

Client	Round	Step	DS in AWS		DS in GCP	
			AWS	GCP	GCP	AWS
0	1^{st}	T	0:03:58	0:03:17	0:02:43	0:04:41
		E	0:02:44	0:02:31	0:00:38	0:00:52
	2^{nd}	T	0:06:52	0:07:26	0:02:23	0:02:54
		E	0:02:59	0:03:38	0:00:31	0:00:52
1	1^{st}	T	0:03:10	0:03:06	0:02:40	0:02:46
		E	0:02:26	0:02:53	0:00:31	0:00:56
	2^{nd}	T	0:07:26	0:06:09	0:02:24	0:02:55
		E	0:02:08	0:03:08	0:00:26	0:00:52
2	1^{st}	T	0:02:49	0:03:30	0:02:49	0:04:38
		E	0:02:27	0:02:34	0:00:35	0:00:52
	2^{nd}	T	0:06:39	0:07:27	0:02:23	0:03:16
		E	0:02:42	0:03:26	0:00:30	0:00:46
3	1^{st}	T	0:02:56	0:03:16	0:02:42	0:02:30
		E	0:01:36	0:02:41	0:00:36	0:01:09
	2^{nd}	T	0:06:24	0:07:17	0:02:25	0:02:58
		E	0:02:49	0:02:33	0:00:31	0:00:53

30 instance types with NVIDIA GPUs [32], and in GCP, there are six different NVIDIA GPU architectures attachable to 27 instance types [33]. Therefore, in future experiments, the Pre-Scheduling module should repeat these tests for every instance considered in the framework environment.

V. Concluding Remarks

This paper proposed a framework to execute Federated Learning applications in a multi-cloud environment minimizing the total execution time and costs. The concept of the whole framework is introduced. The first module, the Pre-Scheduling, is more deeply described and some preliminary experiments were presented and discussed.

As future work, in short-term, we intend to evaluate different scenarios of communication delays and VM failures. Then, we will consider experiments with other FL applications, including known datasets, such as CIFAR-10 [34] and MNIST [35]. Finally, we will focus on the conception of the other modules of the Multi-FedLS framework.

ACKNOWLEDGMENT

This research is supported by project CNPq/AWS 440014/2020-4, Brazil, by *Programa Institucional de Internacionalização* (PrInt) from CAPES (process number 88887.310261/2018-00), UNIVERSAL CNPq (process number 145088/2019-7), and the *Coordenação de Aperfeiçoamento de Pessoal de Nível Superior* (CAPES - Finance Code 001).

REFERENCES

[1] 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*, vol. n/a, no. n/a, 2020. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.6002

[2] 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

[3] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, A. Singh and J. Zhu, Eds., vol. 54. Fort Lauderdale, FL, USA: PMLR, 4 2017, pp. 1273–1282. [Online]. Available: http://proceedings.mlr.press/v54/mcmahan17a.html

[4] V.-D. Nguyen, S. K. Sharma, T. X. Vu, S. Chatzinotas, and B. Ottersten, "Efficient federated learning algorithm for resource allocation in wireless iot networks," *IEEE Internet of Things Journal*, vol. 8, no. 5, pp. 3394–3409, 2021.

[5] S. Rajendran, J. S. Obeid, H. Binol, R. D'Agostino, K. Foley, W. Zhang, P. Austin, J. Brakefield, M. N. Gurcan, and U. Topaloglu, "Cloud-Based Federated Learning Implementation Across Medical Centers," *JCO Clinical Cancer Informatics*, no. 5, pp. 1–11, 2021, pMID: 33411624. [Online]. Available: https://doi.org/10.1200/CCI.20.00060

[6] A. Li, X. Yang, S. Kandula, and M. Zhang, "CloudCmp: Comparing Public Cloud Providers," in *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 1–14. [Online]. Available: https://doi.org/10.1145/1879141.1879143

[7] A. W. Services, "Recommended GPU Instances," https://docs.aws.amazon.com/dlami/latest/devguide/gpu.html, 2022, accessed 19 January 2022.

[8] P. Google Cloud, "GPUs on Compute Engine," https://cloud.google.com/compute/docs/gpus, 2022, accessed 19 January 2022.

[9] ——, "Cloud Tensor Processing Units (TPUs)," https://cloud.google.com/tpu/docs/tpus, 2022, accessed 19 January 2022.

[10] S. Ostermann, A. Iosup, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema, "A performance analysis of ec2 cloud computing services for scientific computing," in *Cloud Computing*, D. R. Avresky, M. Diaz, A. Bode, B. Ciciani, and E. Dekel, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 115–131.

[11] C. Kotas, T. Naughton, and N. Imam, "A comparison of amazon web services and microsoft azure cloud platforms for high performance computing," in *2018 IEEE International Conference on Consumer Electronics (ICCE)*, 2018, pp. 1–4.

[12] H. H. Yang, Z. Liu, T. Q. S. Quek, and H. V. Poor, "Scheduling policies for federated learning in wireless networks," *IEEE Transactions on Communications*, vol. 68, no. 1, pp. 317–333, 2020.

[13] W. Xia, T. Q. S. Quek, K. Guo, W. Wen, H. H. Yang, and H. Zhu, "Multi-armed bandit-based client scheduling for federated learning," *IEEE Transactions on Wireless Communications*, vol. 19, no. 11, pp. 7108–7123, 2020.

[14] 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 International Conference on Communications Workshops (ICC Workshops)*, 2021, pp. 1–6.

[15] B. Buyukates and S. Ulukus, "Timely communication in federated learning," in *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2021, pp. 1–6.

[16] L. Lima Pilla, "Optimal task assignment for heterogeneous federated learning devices," in *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2021, pp. 661–670.

[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, pp. 6270–6284, 2019.

[18] 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, p. 106002, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S095070512030304X

[19] 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*, pp. 1–1, 2021.

[20] A. Ingerman and K. Ostrowski, "TensorFlow Blog: Introducing TensorFlow Federated," https://blog.tensorflow.org/2019/03/introducing-tensorflow-federated.html, 2019, accessed 16 August 2021.

[21] T. Ryffel *et al.*, "A generic framework for privacy preserving deep learning," *ArXiv*, vol. abs/1811.04017, 2018.

[22] A. Paszke *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, 2019.

[23] FedAI, "FATE," https://fate.readthedocs.io/en/latest/, 2019, accessed 27 December 2021.

[24] 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.

[25] R. Brum, L. Drummond, M. C. Castro, and G. Teodoro, "Towards optimizing computational costs of federated learning in clouds," in *2021 International Symposium on Computer Architecture and High Performance Computing Workshops (SBAC-PADW)*, 2021, pp. 35–40.

[26] 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*. Porto Alegre, RS, Brasil: SBC, 2021, pp. 6–10. [Online]. Available: https://sol.sbc.org.br/index.php/eradrj/article/view/18558

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

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

[29] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd Int. Conf. on Learning Representations, ICLR 2015*, 2015. [Online]. Available: http://arxiv.org/abs/1409.1556

[30] E. M. Malta, S. Avila, and E. Borin, "Exploring the Cost-Benefit of AWS EC2 GPU Instances for Deep Learning Applications," in *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing*, ser. UCC'19. New York, NY, USA: Association for Computing Machinery, 2019, p. 21–29. [Online]. Available: https://doi.org/10.1145/3344341.3368814

[31] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning," in *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, ser. AAAI'17. AAAI Press, 2017, p. 4278–4284.

[32] A. W. Services, "Amazon EC2 Instance Types," https://aws.amazon.com/ec2/instance-types/, 2021, accessed 19 December 2021.

[33] P. Google Cloud, "Machine Families - Documentation," https://cloud.google.com/compute/docs/machine-types#predefined_machine_types, 2022, accessed 20 February 2022.

[34] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[35] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.