Multi-FedLS : A Scheduler of Federated Learning Applications in a Multi-Cloud Environment

Rafaela Correia Brum

Universidade Federal Fluminense
Instituto de Computação
Sorbonne Université
École doctorale Informatique,Télécommunications et Électronique (Paris)
Laboratoire d'informatique de Paris 6 (LIP6)

RAFAELA CORREIA BRUM

Multi-FedLS: A Scheduler of Federated Learning Applications in a Multi-Cloud Environment

NITERÓI

2023

Universidade Federal Fluminense
Sorbonné Université

RAFAELA CORREIA BRUM

Multi-FedLS: A Scheduler of Federated Learning Applications in a Multi-Cloud Environment

Thesis presented in joint supervision (cotutelle) between Universidade Federal Fluminense and Sorbonne Université. Thesis presented to the Computing Graduate Program of the Universidade Federal Fluminense and the École doctorale Informatique,Télécommunications et Électronique Sorbonne Université in partial fulfilment of the requirements for the degree of Doctor of Science and the *diplôme national de doctorat*. Topic Area: Computer Science

Advisor:
LÚCIA MARIA DE ASSUMPÇÃO DRUMMOND (UFF)

Co-advisor:
PIERRE SENS (Sorbonne Université)

NITERÓI

2023

Ficha catalográfica automática - SDC/BEE
Gerada com informações fornecidas pelo autor

B893m Brum, Rafaela Correia
Multi-FedLS: A Scheduler of Federated Learning Applications in a Multi-Cloud Environment / Rafaela Correia Brum. - 2023.
116 f.: il.

Orientador: Lúcia Maria de Assumpção Drummond.
Coorientador: Pierre Sens; Maria Clicia Stelling De Castro.
Tese (doutorado)-Universidade Federal Fluminense, Instituto de Computação, Niterói, 2023.

1. Escalonamento de tarefas. 2. Computação em nuvem. 3. Ambiente com múltiplas nuvens. 4. Aprendizado Federado. 5. Produção intelectual. I. Drummond, Lúcia Maria de Assumpção, orientadora. II. Sens, Pierre, coorientador. III. De Castro, Maria Clicia Stelling, coorientadora. IV. Universidade Federal Fluminense. Instituto de Computação.V. Título.

CDD - XXX

Bibliotecário responsável: Debora do Nascimento - CRB7/6368

RAFAELA CORREIA BRUM

Multi-FedLS: A Scheduler of Federated Learning Applications in a Multi-Cloud Environment

Approved on November 29th, 2023 by:

Dr. Lúcia M. A. Drummond, D.Sc. / IC / Universidade Federal Fluminense (President)

Dr. Pierre Sens, Ph.D. / LIP6 / Sorbonne Université (Co-advisor)

Dr. Luciana Arantes, Ph.D. / LIP6 / Sorbonne Université

Dr. Maria Clicia Stelling de Castro , D.Sc. / IME / Universidade do Estado do Rio de Janeiro

Dr. Guillaume Pierre, Ph.D. / IRISA / Université de Rennes (reviewer)

Dr. Christophe Cérin, Ph.D. / LIPN / Université de Paris Nord (reviewer)

Dr. Maria Cristina Silva Boeres, Ph.D. / IC / Universidade Federal Fluminense

Dr. Aline Marins Paes Carvalho, D.Sc. / IC / Universidade Federal Fluminense

Niterói
2023

Agradecimentos

Gostaria de agradecer primeiramente a Deus, por toda força me dada para continuar nessa difícil empreitada. Também agradeço a Nossa Senhora, por todos os momentos que eu senti sua proteção e intercessão.

Às professoras Lúcia Drummond, Maria Clicia Stelling e Luciana Arantes, por suas orientações, apoio e carinho. Grata por trabalhar com pessoas tão competentes que buscam sempre o crescimento dos seus alunos.

Au professeur Pierre Sens, pour toute son orientation au cours de ces dernières années. Reconnaissante d'être arrivée jusqu'ici en travaillant avec un professeur aussi dévoué.

Aos meus pais, Antonino e Mônica Brum, por me permitirem seguir meus sonhos até agora. Às minhas irmãs, Renata e Roberta, por todo o apoio dado. E aos meus sobrinhos, Lucas, Maria Alice e Pedro, por serem tão puros e trazerem uma alegria renovadora nesses anos árduos de trabalho.

À Irina Aibara, por todo apoio me dado no meu ano de doutorado sanduíche. Aos meus colegas de doutorado, Luan Teylo, Fernando Chagas, Eder Medeiros, Leonardo Vasconcelos e Mônica da Silva, pelas conversas, ajudas e risadas feitas nos corredores da UFF, e pelo WhatsApp.

Aos meus amigos Jéssyca Torres, Vinicius Mattos, Talita Alburquerque, Maria Carolina Fernandes, Giovanna de Andrade e Caroline Passos pelas orações feitas durante esses anos e pelas noites de conversas e jogos nos finais de semana da pandemia. Aos meus amigos Matheus Carneiro, Matheus Borges e Gabriel Franco pelas noites e tardes de distração que se fortaleceram na pandemia.

Ao meu noivo, Raul de Queiroz, que foi um dos primeiros a acreditar nos meus sonhos.

Ao CNPq pela bolsa concedida nos quatro anos de doutorado. À CAPES pela bolsa de doutorado-sanduíche, concedida por meio do seu Programa Institucional de Internacionalização (CAPES/PrInt). Ao CNPq e AWS pelo projeto BioCloud, que disponibilizou verbas para a utilização de recursos de nuvem.

Resumo

Aprendizado Federado (AF) é uma nova área de Aprendizado de Máquina (AM) distribuído, onde o aprendizado garante a privacidade de dados. Cada cliente tem acesso somente ao seu conjunto de dados local e privativo. Essa abordagem é atrativa em vários domínios do conhecimento porque permite que diferentes instituições colaborem entre si sem compartilhar seus dados confidenciais. Além disso, como a quantidade de dados necessários para o treinamento tem crescido muito nos últimos anos, a maioria das instituições não pode pagar por *data centers* físicos para armazenar e manipular todos os seus dados. Uma opção viável é utilizar serviços de armazenamento na nuvem oferecidos por provedores com diferentes garantias de privacidade e disponibilidade dos dados. O usuário é o responsável pela escolha das regiões onde armazena seus dados e pelo controle de acesso a eles.

Adicionalmente, os provedores de nuvem oferecem vários serviços para executar uma aplicação. Eles oferecem aos usuários a possibilidade de criar Máquinas Virtuais (MV) com diferentes configurações, onde os usuários têm o total controle sobre elas. Este tipo de serviço é denominado Infraestrutura-como-um-Serviço (*Infrastructure-as-a-Service* - IaaS). Sendo assim, o ambiente de nuvem é propício para a colaboração de diferentes instituições na criação de um modelo de Aprendizado de Máquina através do Aprendizado Federado, principalmente considerando diversas nuvens.

Nesta tese, nós propomos o *Multi-FedLS*, uma estrutura robusta criada para executar aplicações de AF em um ambiente *multi-cloud*. O *framework* considera a localização atual dos conjuntos de dados de cada cliente, o atraso de comunicação e o custo de utilização nas nuvens, focando na redução de custo e tempo de execução. Além disso, *Multi-FedLS* utiliza, sempre que possível, instâncias mais baratas visando a redução de custo, mas que pode ser revogada a qualquer momento pelo provedor de nuvem. Assim, para garantir a execução completa das aplicações de AF, o *framework* utiliza técnicas de tolerância a falhas, como *checkpoints* e migração do trabalho para retomar o treinamento em outra MV a partir de uma revogação. O *Multi-FedLS* é composto por quatro módulos: *Pre-Scheduling*, *Initial Maping*, *Fault Tolerance* e *Dynamic Scheduler*. Os resultados obtidos mostram que é viável executar aplicações em ambientes *multi-cloud* com MVs de baixo custo, usando formulação matemática, técnicas de tolerância a falhas e heurística simples para escolha de novas MVs. O *framework* obteve uma redução de custo de 56.92% comparado ao tempo de execução da aplicação sem o *framework*, com um aumento no tempo de execução de somente 5.44% em provedores de nuvem comerciais.

Palavras-chave: Aprendizado Federado, Escalonamento de tarefas, Ambiente com múltiplas nuvens.

Abstract

Federated Learning (FL) is a new area of distributed Machine Learning (ML) where learning ensures data privacy. Each client has access only to its own local and private dataset. This approach is attractive in various domains of knowledge because it allows different institutions to collaborate without sharing their confidential data. Besides, as the amount of data required for training has grown significantly in recent years, most institutions cannot afford physical data centers to store and manipulate all their data. A viable option is to utilize cloud storage services offered by providers with different data privacy and availability guarantees. The user is responsible for choosing the regions where their data is stored and controlling access to it.

Additionally, cloud providers offer various services to execute an application. They provide users with the ability to create Virtual Machines (VMs) with different configurations, where users have full control over them. This type of service is known as Infrastructure-as-a-Service (IaaS). Thus, a cloud environment is conducive to the collaboration of different institutions in creating a Machine Learning model through Federated Learning, mainly when considering multi-cloud environments.

In this thesis, we propose *Multi-FedLS*, a robust framework designed to execute FL applications in a multi-cloud environment. The framework considers the current location of each client's datasets, communication delay, and cost of utilization in the clouds, focusing on cost and runtime reduction. Moreover, *Multi-FedLS* utilizes cheaper instances whenever possible to reduce costs, even though they may be revoked at any time by the cloud provider. Thus, to ensure the successful execution of FL applications, the framework employs fault-tolerance techniques such as checkpoints and work migration to resume training on another VM after a revocation. *Multi-FedLS* comprises four modules: *Pre-Scheduling*, *Initial Mapping*, *Fault Tolerance*, and *Dynamic Scheduler*. The obtained results demonstrate the feasibility of executing applications in multi-cloud environments using low-cost VMs, employing mathematical formulation, fault-tolerance techniques, and simple heuristics for selecting new VMs. The framework achieved a cost reduction of 56.92% compared to application runtime without using it, with only a 5.44% increase in runtime on commercial cloud providers.

Keywords: Federated Learning, Task Scheduling, Multi-Cloud environment.

Resumé

Titre: Multi-FedLS : Un Ordonnanceur d'Applications d'Apprentissage Fédéré dans un Environnement Multi-Cloud

L'apprentissage fédéré (AF) est un nouveau domaine de l'apprentissage machine distribué où l'apprentissage garantit la confidentialité des données. Chaque client a accès uniquement à son propre ensemble de données local et privé. Cette approche est attrayante dans divers domaines du savoir car elle permet à différentes institutions de collaborer sans partager leurs données confidentielles. En plus, comme la quantité de données requises pour la formation a considérablement augmenté ces dernières années, la plupart des institutions ne peuvent pas se permettre des centres de données physiques pour stocker et manipuler l'ensemble de leurs données. Une option viable consiste à utiliser des services de stockage en nuage proposés par des fournisseurs offrant différentes garanties de confidentialité et de disponibilité des données. L'utilisateur est responsable du choix des régions où ses données sont stockées et du contrôle de leur accès.

De plus, les fournisseurs de services en nuage offrent divers services pour exécuter une application. Ils permettent aux utilisateurs de créer des machines virtuelles (MV) avec différentes configurations, où les utilisateurs ont un contrôle total sur celles-ci. Ce type de service est appelé Infrastructure en tant que Service (IaaS). Ainsi, un environnement cloud est propice à la collaboration de différentes institutions dans la création d'un modèle d'apprentissage machine grâce à l'apprentissage fédéré, en particulier lorsqu'on envisage des environnements multi-cloud.

Dans cette thèse, nous proposons *Multi-FedLS*, un *framework* robuste conçu pour exécuter des applications AF dans un environnement multi-cloud. Le *framework* prend en compte l'emplacement actuel des ensembles de données de chaque client, le délai de communication et le coût d'utilisation dans les nuages, en se concentrant sur la réduction des coûts et du temps d'exécution. De plus, *Multi-FedLS* utilise des instances moins chères chaque fois que possible pour réduire les coûts, même si elles peuvent être révoquées à tout moment par le fournisseur de services en nuage. Ainsi, pour assurer l'exécution réussie des applications AF, le cadre utilise des techniques de tolérance aux pannes telles que les points de contrôle et la migration des tâches pour reprendre la formation sur une autre MV après une révocation. *Multi-FedLS* comprend quatre modules: *Pre-Scheduling*, *Initial Mapping*, *Fault Tolerance* et *Dynamic Scheduler*. Les résultats obtenus démontrent la faisabilité de l'exécution d'applications dans des environnements multi-cloud en utilisant des MV peu coûteuses, en utilisant une formulation mathématique, des techniques de tolérance aux pannes et des heuristiques simples pour la sélection de nouvelles MV. Le *framework* a obtenu une réduction des coûts de 56,92% par rapport au temps d'exécution de l'application sans son utilisation, avec seulement une augmentation de 5,44% du temps d'exécution sur les fournisseurs de services en nuage commerciaux.

Mots-clés: Apprentissage Fédéré, Ordonnancement des tâches, Environnement Multi-Cloud

List of Figures

List of Tables

Acronyms and Abreviations

AWS	:	Amazon Web Services
CNN	:	Convolutional Neural Network
DNN	:	Deep Neural Network
FL	:	Federated Learning
GCP	:	Google Cloud Provider
GPU	:	Graphic Processing Unit
ML	:	Machine Learning
RNN	:	Recurrent Neural Network
S3	:	Simple Storage Service
TIL	:	Tumor-Infiltrating Lymphocyte
TPU	:	Tensor Processing Unit
VM	:	Virtual Machine

Contents

Chapter 1

Introduction

The rise of data protection laws (*e.g.*, GDPR[1] in Europe and LGPD[2] in Brazil) led researchers to concern about sharing sensitive data, especially in the field of distributed Machine Learning (ML). The traditional distributed ML usually exchange data among clients to balance their execution times [125].

McMahan *et al.* [92] proposed the term Federated Learning in 2017 as a learning technique that allows users to collectively benefit from shared models trained from distributed data without centrally storing them.

This way, Federated Learning (FL) is a recent type of distributed ML in which the participating clients do not share their private data [125]. The client federation solves the learning task coordinated by a central server without sharing the data. Instead, each client computes and communicates only the model weights to update the current global model kept by the server.

The server-clients architecture of FL, also called Model-Centric Federated Learning [145], is classified into Cross-Device or Cross-Silo Federated Learning, depending on the connected client's type. If the clients are low-powered devices, like mobile phones [92] or edge devices [97], it is a Cross-Device Federated Learning. If the clients are companies or institutions (*e.g.*, hospitals [102]) with similar datasets willing to create a central model, it is a Cross-Silo Federated Learning. In this second type of FL, the central server can assume that all clients are available during the whole process, and there are usually fewer clients than in Cross-Device FL (less than 10 [84]).

Moreover, the digital data created by an institution and used to train ML algorithms

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

[2] http://www.planalto.gov.br/ccivil_03/_ato2015-2018/2018/lei/L13709compilado.htm

increases rapidly [136]. Most institutions cannot upgrade their data centers due to high financial costs. One viable option is the use of cloud storage services, in which the user only pays for the amount of stored data [81, 88]. Many cloud providers offer these storage services with different privacy guarantees and data availability. For instance, in the Google Cloud provider, the user can store the data in a multi-region configuration. The data is available in different regions of the same country, and the user defines who can access it.

In this scenario, each participating institution of the FL environment can choose the best storage option among the different cloud providers. According to Li *et al.* [83], the response time to access a file in one cloud provider can be twice the time to access it in another provider. Besides, they show that the maximum throughput of one cloud provider can vary up to 57%, and in another cloud provider varies less than 2%. Moreover, inside the same cloud provider, diverse storage options have different response times and throughput, as presented by Teylo *et al.* in [130]. Each institution chooses the best cloud provider and storage service based on its needs. However, FL clients might have their data in different cloud providers and, to preserve privacy, it is prohibitive to transfer all data to a single cloud provider. Thus, in this case the FL application needs to execute in a multi-cloud environment. Since the basis of FL algorithms is the message exchange between the server and clients, communication time and costs within a cloud provider or several cloud providers can have a meaningful impact on application performance.

Besides the cloud storage services, cloud providers offer Virtual Machines (VMs) with different accelerators in a service generically called Infrastructure-as-a-Service (IaaS). These accelerators can reduce the execution time of Deep Neural Networks (DNNs), which are a ML algorithm type consisting of many layers, represented by weights' matrices. It is necessary to execute several matrix multiplications to obtain the DNN's answer. As the DNN training consists of passing all training samples multiple times through the model, huge computational power to train them is usually required. Matrix multiplication is an example of a single instruction multiple data (SIMD) processing [48]. Nowadays, there are two accelerators specialized in these instructions: the Graphics Processing Units (GPUs)[3] and Google's Tensor Processing Units (TPUs)[4]. Amazon Web Services (AWS) offers GPUs attached to pre-defined VMs types, and the available GPU architectures vary from Kepler to Ampere [121]. Google Cloud Platform (GCP) allows the user to attach GPUs or TPUs to a pre-defined or custom VM type [60, 57].

[3]https://blogs.nvidia.com/blog/2009/12/16/whats-the-difference-between-a-cpu-and-a-gpu/

[4]https://cloud.google.com/blog/products/ai-machine-learning/an-in-depth-look-at-googles-first-tensor-processing-unit-tpu

It is also important to consider the costs of public clouds. Usually, there are two main markets to deploy VMs with different availability guarantees and costs: the on-demand and the preemptible (spot) market. The on-demand market allocates VMs for a fixed cost per time unit, ensuring their availability during the whole execution. On the other hand, the preemptible (spot) market offers a high discount, but the provider can terminate the VM at any time.

In a multi-cloud environment, a FL framework must identify the cloud provider assigned to each client. It must also define the instance and the region assigned to each of them. In addition, it may be necessary to specify the minimum required network bandwidth. Consequently, the task of determining which platform to use for each client is complex, and a wrong choice can involve a considerable additional cost [79, 100]. Furthermore, the transfer rate between cloud providers and between different regions inside the same cloud provider can increase the time of the whole FL execution.

As previously pointed out, the main concern in FL is data privacy. Data leakage episodes occurred in 2017 despite cloud providers guaranteeing privacy[5,6]. These events involved Amazon Web Services, a telecommunication US company[7], and the US Government[8]. Therefore, the problem of efficient resource allocation and security management is critical to the success of FL in clouds. Also, activities such as the configuration of the environment, monitoring the execution, and choosing a new instance, if the current selection is unavailable, should be taken into account.

1.1 Objective

The main objective of this thesis is to explore the multi-cloud environment to execute Federated Learning efficiently while preserving data privacy and allowing the collaboration of different institutions in the creation of new knowledge. It also focuses on possible savings in time and costs of the necessary infrastructure to execute FL applications. We propose *Multi-FedLS* to achieve these objectives through a robust, adaptative, and flexible framework. Its robustness guarantees the complete execution of the applications aiming at reducing time and costs. Its adaptativeness provides application execution in different environments, whether it is simulation environments, single clouds, new cloud providers, or new regions. Finally, its flexibility allows the easily inclusion of new modules.

[5]https://aws.amazon.com/compliance/data-privacy-faq/
[6]https://cloud.google.com/security
[7]https://www.upguard.com/breaches/verizon-cloud-leak
[8]https://www.upguard.com/breaches/spy-games-booz-allen-hamilton-pentagon

In this thesis, we concentrate on the problem of scheduling Cross-Silo FL applications in a multi-cloud scenario considering the clients' data location to preserve privacy. Our proposal, *Multi-FedLS*, is a framework to manage multi-cloud resources reducing the execution time and financial costs of these applications. This framework comprises four modules, and each of them issues on one or more of the desired characteristics. They are the Pre-Scheduling, Initial Mapping, Fault Tolerance, and Dynamic Scheduler modules. Our 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). The first module, Pre-Scheduling, is responsible for collecting specific information about the selected environment and the FL application when there is no previous knowledge about them, such as the communication delays and expected execution time of the FL tasks. This module focuses on the adaptative and flexible features of our framework, as it automatically collects new environmental data when there is the addition of new VMs, regions, or even cloud providers without recollecting the ones that already have information. 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. As this module uses a mathematical formulation to provide the best scheduling map, it concentrates on the robustness of our framework focusing on finishing the application execution in cheap VMs and within a given deadline and budget. Finally, Multi-FedLS deploys the selected VMs, starts the FL application, and monitors it. The Fault Tolerance module is responsible for including checkpoint strategies in the FL application, on both server and client sides, concentrating again on the robustness of our framework. In case of an unexpected error or a VM failure, that module triggers the Dynamic Scheduler Module to select a new VM and resume the tasks of the FL application. Our last module emphasizes the robustness and adaptation of our framework to eventual revocations during the application execution. We should also emphasize that experiments with Multi-FedLS were conducted with both applications from benchmarks and a real-world application, showing how flexible our framework can be.

1.2 Contributions

The main contributions of this work are the following:

I The proposal of the Multi-FedLS framework that explores preemptible (spot) VMs to

minimize the monetary cost of the execution in a multi-cloud scenario while ensuring the complete execution of the application;

II The design of the Pre-Scheduling module that collects unknown information about the FL application and environment making the framework adaptative to new environments;

III The design of the Initial Mapping module that defines the mathematical formulation used in the initial scheduling map choosing the optimal VM allocation considering both monetary costs and execution time;

IV The design of the Fault Tolerance module that includes checkpoint techniques on both server and client sides ensuring the complete execution even in case of revocations;

V The design of the Dynamic Scheduler module that chooses a new VM in case of eventual revocations;

VI Empirical evaluations of the proposed framework in both emulated and real multi-cloud environments; and

VII The proposal and evaluation of an FL approach to a use-case application in clouds.

This work was conducted in collaboration with the LIP6 laboratory from Sorbonne Université (SU) as part of the ReMatCH project[9]. That collaboration started in 2020 and yielded a total of three conference papers [14, 16, 17], a journal paper submitted to JPDC [15], a one-year stay in the LIP6 laboratory as part of a double PhD degree by Federal Fluminense University and SU, and several events participation. Moreover, in 2021 we start a project called BioCloud, in partnership with CNPq and AWS[10]. From this project, we transformed a centralized Tumor-Infiltrating Lymphocytes (TIL) application that predicts a patient's survival rate to use an FL approach, published in [12, 13]. This application uses patient image data, which can be sensitive being a good candidate to benefit from Federated Learning.

1.3 Thesis Outline

The remainder of this thesis is organized as follows. Chapter 2 introduces some background concepts about Federated Learning and Cloud Computing. Chapter 3 presents

[9]More information about the project can be found in `http://cloud.ic.uff.br/index.php/pt/capes-print/`

[10]More information about the project can be found in `http://cloud.ic.uff.br/index.php/project-cnpq-aws/`

the related work and compares our proposal's contributions with existing ones. Chapter 4 describes the tumor-infiltrating lymphocytes application and our Federated Learning approach to solve it. Chapter 5 presents the models for a FL application and the environment considered in the Multi-FedLS proposal along with the framework's general architecture while Chapter 6 describes each module of Multi-FedLS in details. Chapter 7 describes and discusses the experiments regarding our framework. Finally, Chapter 8 concludes the thesis and introduce some future directions.

Chapter 2

Background

This chapter introduces background concepts used in the current work. Section 2.1 explains some basic concepts of Machine Learning and Deep Learning research fields. Section 2.2 presents concepts of Distributed Machine Learning, and Section 2.3 introduces Federated Learning concepts. Section 2.4 describes some FL tools. Finally, Section 2.5 explains Cloud Computing concepts, including multi-cloud environments.

2.1 Machine Learning

Machine Learning (ML) is a field in the Artificial Intelligence area that learns to perform a task without being explicitly programmed [94]. Each experience collected from the real world is called a sample or example. In this work, we focus on supervised learning classification algorithms, which expect as input a training dataset and outputs an estimator of the class for a new example. Each example is modelled as a set of features, which describes its main aspects. A training dataset S is a set of N classified objects $\{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N)\}$, chosen from a domain X with an arbitrary, fixed, and unknown distribution $\mathcal{D}$. These classification values are given by some unknown function $y = f(\mathbf{x})$. The $\mathbf{x}_i$ objects are typically vectors of the form $(x_{i1}, x_{i2}, ..., x_{im})$, whose values can be discrete or real. Each value x_{ij} denotes the value of the j-th feature X_j of the object $\mathbf{x}_i$. For classification purposes, y_i values typically belong to a discrete set of L labels, *i.e.* $y_i \in \{l_1, l_2, ..., l_L\}$; for regression purposes, y_i values belong to the (sub)set of real values. Here, we focus on classification problems.

2.1.1 Deep Learning

Deep Neural Networks (DNNs) are the most common algorithms in Deep Learning [49], which can handle datasets with hundreds or even thousands of features. DNNs consist of many connected layers with many neurons in each layer. Each neuron consists of a linear combination of activation functions and receives the results of the previous layer multiplied by weights, which are arbitrary float-pointing values, as input. The goal of DNNs is to find near-optimal weights to understand the concepts behind the dataset extracting patterns from it to classify each sample into one of the defined categories. Figure 2.1 illustrates a DNN mapping four input features (X_1, X_2, X_3, X_4) to two output classes (y_1, y_2) with three hidden layers and three neurons in each hidden layer.

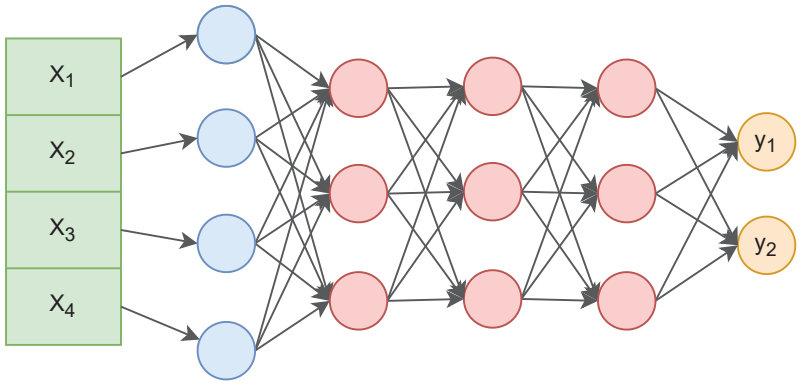


Figure 2.1: Simple DNN with four input features and two output classes. The blue layer represents the input layer, the red ones are the hidden layers, and the yellow one is the output layer.

The blue layer in Figure 2.1 represents the input layer, and it maps each input feature to a single value used as input to the next layer. Then, the red layers represent the three hidden layers of this DNN. Each hidden layer receives the values from the previous layer, computes a linear combination of weights and the input values in each neuron, and sends the results to the next layer. These layers usually receive the results of all neurons of the previous layer and send the computed value to all neurons of the next layer. For this reason, they are also called fully-connected layers. Finally, the yellow layer represents the output layer, which receives the values from the last hidden layer and usually presents the probability of the current sample belonging to each output class.

Usually, DNNs have several fully-connected layers with several neurons in each one. Also, Goodfellow *et al.* [49] explain different types of DNNs focused on distinct input data types. Two of them are:

- Convolutional Neural Networks (CNNs) are mostly used with image inputs. Instead of mapping each pixel of an image to a single neuron in the input layer, CNNs use convolution layers before the fully-connected layers to process the image and reduce the input layer size.

- Recurrent Neural Networks (RNNs), a type of DNN focused on sequential data inputs. One word in a text can have a different meaning depending on its context. For example, the word 'give' in "I give you a present" means the action of delivering something, and "I give up" means to surrender. With that in mind, researchers connect the last hidden layers of an RNN to the first ones. This recurrent connection saves the result of one sample and influences the results of the following samples.

Our framework supports these two and any other type of DNNs. In each learning step, called epoch, the DNN computes the class probabilities of all training samples. To calculate them, the DNN executes sequential matrix multiplications. These multiplications are called the feedforward step as the information goes from the input to the output layer [49]. After finishing this step, the DNN calculates an error function, and the error propagates through the DNN from the output layer to the input one. This process is called the backpropagation step, in which the model computes new weights to each connection of the DNN. Both the feedforward and the backpropagation steps are executed at least one time each in a single epoch, and, usually, a DNN needs several steps to reach good metrics in the evaluation. Therefore, training a single DNN requires a huge amount of computational power. One option to achieve the performance is using accelerators, such as Graphics Processing Units (GPUs). GPUs focus on a single instruction execution on multiple data, which matches the description of matrix multiplications.

Nowadays, the DNN's complexity and available training data increase continuously. These factors lead to efficient and parallel training of the models. Distributed Machine Learning, also named Distributed Deep Learning, rises to tackle these needs [7].

2.2 Distributed Machine Learning

Researchers in the Distributed Machine Learning area present three different forms to parallelize the execution of a single ML algorithm, frequently a DNN, on multiple devices. These forms are: (i) data parallelism, which replicates the model in all devices and splits the dataset among them; (ii) model parallelism, which replicates the dataset in all devices and splits the model among them; and (iii) pipelining, which divides the steps among

devices to overlap communications with computations. Data parallelism is the most used form as it is simpler to implement than the other two. Figure 2.2 shows an example of data parallelism architecture of Distributed Machine Learning, with one parameter server and four workers.

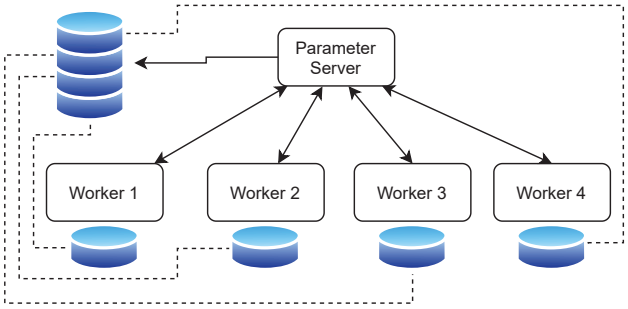


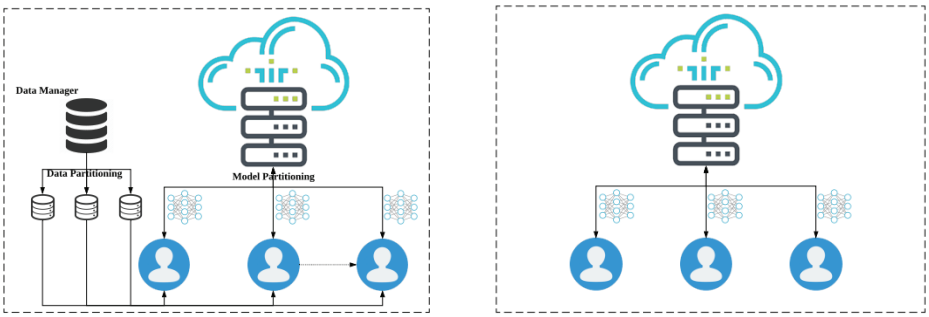
Figure 2.2: Architecture of a Distributed ML scenario using Data Partition and 4 workers.

We can observe from Figure 2.2 that the parameter server is responsible for managing the whole Distributed Machine Learning process, from the division of the dataset among the clients to the aggregation of the weights after each training epoch. Each worker receives the model and the partial dataset from the parameter server and starts the training. The parameter server is responsible for the load balancing among the workers, so they have similar execution times and do not waste time waiting for the synchronization.

The premise of Distributed Machine Learning is the execution of a single dataset and model on multiple devices. Thus, the parameter server needs to access all datasets to distribute them among the workers. Moreover, the parameter server also considers the class distribution of the whole dataset when distributing it to the workers, *i.e.* if the total dataset has more samples of one class than the other, the parameter server will respect this proportion in all partial datasets.

Figure 2.3 presents the differences between a traditional distributed Machine Learning system and a Federated Learning one, the focus of this thesis.

A traditional distributed ML architecture (Figure 2.3a) usually has a central dataset that is distributed among workers by the parameter server (PS), using a Data Manager (Data Partitioning). After that, the PS sends the model to the workers to start the training (Model Partitioning). One worker can communicate with another if it finishes the training epochs faster to balance their load. On the other hand, a FL architecture (Figure 2.3b) does not have a centralized dataset, and the clients cannot communicate among them.



(a) Distributed ML (b) Federated Learning

Figure 2.3: Architecture of (a) Distributed Machine Learning and (b) Federated Learning. Figures from [125].

2.3 Federated Learning

In the past few years, we observed an increasing concern around personal data which led to data protection laws in many countries. Nowadays, institutions need special permission to store and share users' data with others. For example, a medical research institution needs the patients' consent to use their data in future research. This concern leads to a scenario where collecting different datasets and storing them in a unique place is not viable. Each institution owns a dataset with different sizes and classes distribution. Federated Learning (FL) [92] emerged as a category of Distributed Machine Learning in which a central server does not have access to any data and only aggregates the weights received from different clients. Also, each institution can have a unique class distribution, which the central server needs to consider to create a global model.

McMahan *et al.* [92] present the concept of FL as a framework with two main parties: the clients and a central server. Each participating client trains the model with its local dataset. After that, it sends the model updates to the server. The central server coordinates the learning of all clients by only aggregating their model updates. The data is only accessed by the owner, preserving thus privacy.

The authors of this paper use Google Keyboard, the GBoard, as a use-case of Federated Learning, and the clients are Android phones[1]. As mobiles can discharge and disconnect at any time, the FL algorithm created by McMahan *et al.* [92] samples a fraction of all connected clients to participate in the learning in each communication round. One round is composed of four steps, represented in Figure 2.4:

1. the central server chooses a fraction of the participant clients and sends the current

[1]https://ai.googleblog.com/2017/04/federated-learning-collaborative.html

model weights to them;

2. each client trains for several epochs on the local dataset and sends the updates to the server;

3. the central server receives and aggregates all updates, and sends the final weights back to the clients;

4. each client updates their weights and tests the model with a testing dataset , sending back to the server the metrics to be aggregated and start the next communication round.

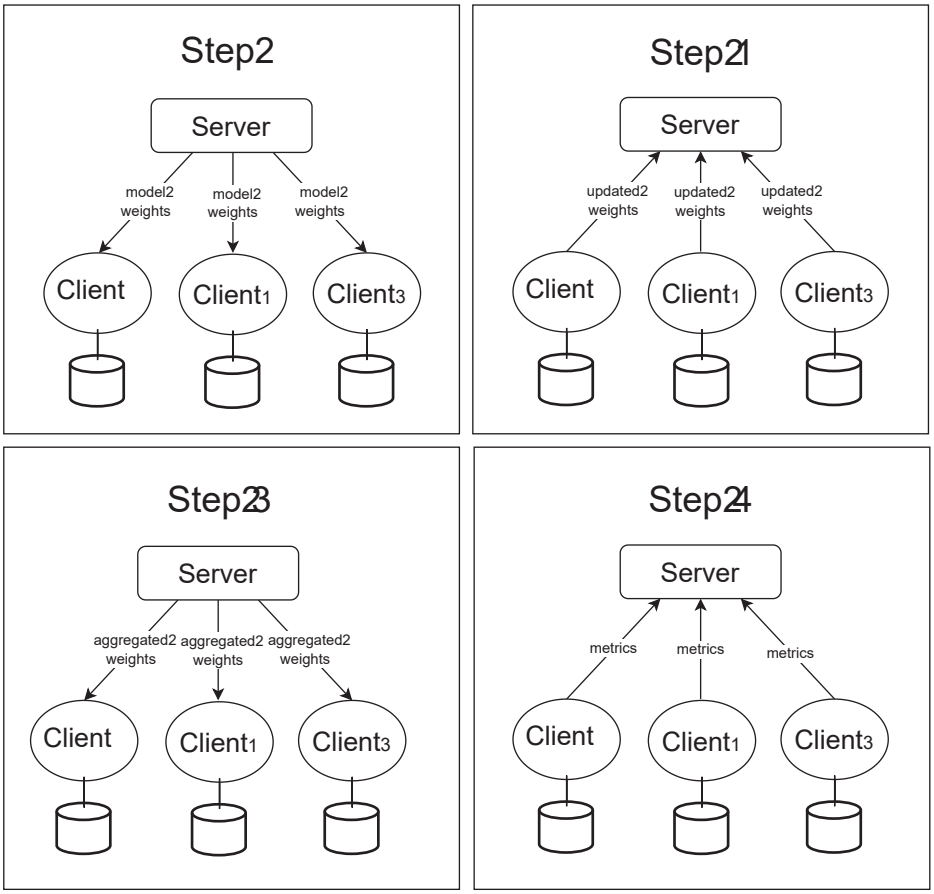
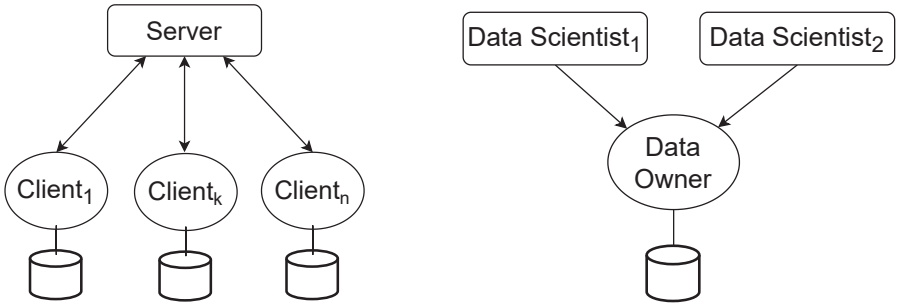


Figure 2.4: Steps of a communication round in Federated Learning.

2.3.1 Federated Learning Classification

There are two possible architectures for Federated Learning in the literature: Model-Centric Federated Learning [145] and Data-Centric Federated Learning [69]. The Model-Centric FL is the first architecture presented to the area, with a central model to coordinate the learning process among all clients. On the other hand, in the Data-Centric FL,

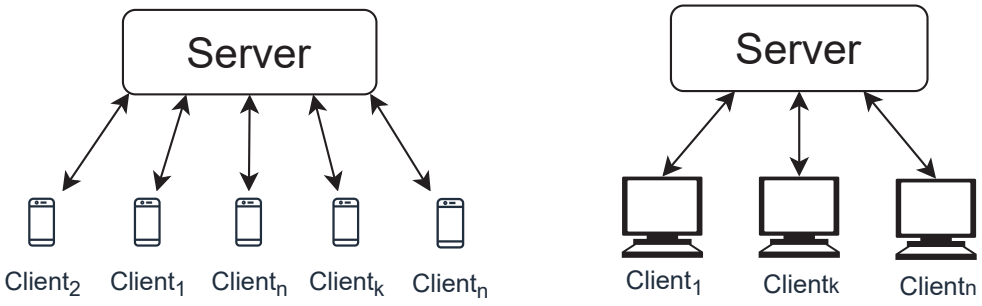
the enterprise owns the data and stores it with private access, and third parties ask for training or inference on that data without seeing it. In this type of FL, the enterprise that stores the dataset is called Data Owner, and each third party is a Data Scientist. Figure 2.5 shows the difference between the Model-Centric and the Data-Centric Federated Learning.



(a) Model-Centric FL (b) Data-Centric FL

Figure 2.5: Example architecture of (a) Model-Centric FL and (b) Data-Centric FL.

Most research in FL focuses on Model-Centric architecture, which admits classification into Cross-Device or Cross-Silo FL. This classification is relative to the connected client type, as shown in Figure 2.6. If the clients are low-powered devices, like mobile phones [92] or edge devices [103], we name it a Cross-Device Federated Learning (Figure 2.6a). There are some challenges concerning the power consumption and the connection in this type of FL. On the other hand, if the clients are companies (*e.g.*, hospitals [102]) with similar datasets willing to create a central model, it is a Cross-Silo Federated Learning (Figure 2.6b).



(a) Cross-Device FL (b) Cross-Silo FL

Figure 2.6: Type of clients in (a) Cross-Device FL and (b) Cross-Silo FL.

Another classification considers the distributed dataset features and the samples. If all clients have the same feature space but different samples, it is a Horizontal Federated Learning [145]. In this scenario, all clients use the same features as input to the model. The collaboration among clients is straightforward, with a centralized server to aggregate

the training weights. For example, two banks can collaborate to create a centralized fraud detector model on credit card transactions. Both have the same feature space, with possible features being the current income, savings, occupation, credit card limit, the average purchase price.

On the other hand, if the clients do not have the same feature space but have the same samples, they can collaborate to create a more sophisticated model. For example, a fitness application can collaborate with a hospital to create a model to understand the relationship between exercise and health. This type of FL is called Vertical Federated Learning [145]. The training is more challenging as the parties need to send and receive intermediate training results to insert them in its local train.

In this work, we focus on Cross-Silo and Horizontal Federated Learning applications.

2.4 Available Tools for Federated Learning

There are several tools for executing Federated Learning applications in the literature. Table 2.1 presents an overview of them. This table describes the designed environment (Environment), the supported FL type (FL type), and the ML frameworks (ML framework) supported by each tool.

Table 2.1: Available tools for Federated Learning applications

Name	Environment	FL type	ML framework
TensorFlow Federated [74]	Simulation	Model-centric	TensorFlow
PySyft/PyGrid [105]	Simulation/Real	Data-centric	PyTorch
FL_PyTorch [19]	Simulation	Model-centric	PyTorch
FATE [37]	Simulation/Real	Model-centric	Custom
Flower [8]	Simulation/Real	Model-centric	Any

We will discuss each of them in the following sections.

2.4.1 TensorFlow Federated

TensorFlow Federated (TFF) [74] is Google's library for executing Federated Learning applications in a simulation environment based on the well-known TensorFlow (TF) API [1]. TFF gives the user two API layers: Federated Learning (TFF-FL) API, for executing basic FL algorithms such as FedAvg, and Federated Core (TFF-FC) API, for implementing new FL algorithms.

TFF-FL API presents wrapper functions on TF Models to handle the communication in the TFF environment [45]. It also implements a class that wraps the dataset distribution among clients in the simulation environment. This class contains two datasets for FL environments: a distributed version of the MNIST dataset for image recognition [26] and a distribution version of a text dataset where each client has lines of a single character in a Shakespeare's play [46].

On the other hand, TFF-FC API is a programming environment to implement new Federated Learning algorithms [44]. TFF-FC is designed to implement the distributed functions from a global perspective instead of the client's or server's perspective. Thus, this API includes pre-defined distributed functions, such as *federated_sum* and *federated_reduce*, to be used by the server.

The authors of TFF claim that it can simulate the FL environment in several machines with multiple GPUs[2]. However, we could not find any explanation on how to set up this multi-machine simulation in its tutorials[3], and we were not able to execute this multi-machine simulation on our own.

2.4.2 PySyft/PyGrid

PySyft and PyGrid are two parts of the available tool focused on Data-Centric Federated Learning [105], in which the users compute from data they cannot see. In this type of FL, the actors are Data Scientists, which do not have any data, and the Data Owner, which holds all data and receives the computational requests from the Scientists.

PySyft is the library API of this tool that defines which type of data Scientists can reference in the Data Owner dataset and the FL algorithm used in the collaboration. PyGrid is the environment platform that stores the Data Owner dataset using containers.

Until June 2021, PySyft[4] and PyGrid[5] were separate projects. Now, the researchers integrated the PyGrid project under the PySyft one. They are currently working on creating tutorials to explain the creation of the data storage using PyGrid and connecting to it using PySyft.

[2]https://www.tensorflow.org/federated/tutorials/simulations_with_accelerators
[3]https://www.tensorflow.org/federated/tutorials/simulations
[4]https://github.com/OpenMined/PySyft
[5]https://github.com/OpenMined/PyGrid

2.4.3 FL_PyTorch

Burlachenko *et al.* propose FL_PyTorch, a FL simulation tool built from the PyTorch API [101]. The authors propose a simulator to help the researchers develop new FL aggregation techniques. The authors focus on four objectives to their tool: simplicity, extensibility, hardware utilization, and easy debugging.

The authors implemented FL_PyTorch with a generalized FL algorithm to help researchers build new FL algorithms. They divided FL_PyTorch into modules mapped to different PyTorch data types and functions, which gives researchers a simple way to modify this tool. Moreover, FL_PyTorch exploit the clients' independent execution to parallelize them by assigning each client to one single CPU thread. FL_PyTorch maps each CPU thread to a different GPU connection if available, so each client sends the work to the GPU independently.

Finally, FL_PyTorch presents a Graphic User Interface (GUI) so FL practitioners can straightforwardly execute their simulations and visualize the measured metrics in each communication round in graphics. This GUI also monitors the whole FL simulation to help researchers debug their novel FL algorithms. Although the simple way to execute FL applications in FL_PyTorch, this tool is focused on simulation and cannot run in a real-world scenario.

2.4.4 Federated AI Technology Enabler (FATE)

Federated AI Technology Enabler (FATE) [37] provides a secure framework to support the whole FL environment. It presents six separated submodules to handle different aspects of the environment: FederatedML, Federated Network, FATE Serving, FATEFlow, FATEBoard, and KubeFATE.

FATE does not support any known ML API as it provides its API through the FederatedML submodule [42]. This submodule presents FL algorithms and data definitions to be used among clients and the server. Federated Network [41] is the FATE submodule to implement the communication tunnels among all participants in FL. It offers a metadata manager and a proxy to implement the message exchange infrastructure.

FATE Serving [38] is the submodule focused on post-training inference. After the FL model finishes training, FATE Serving handles the income of new data and uses trained models to present its classification. FATEFlow [40] manages the whole FL pipeline, from

data processing to serving inference. A Direct Acyclic Graphic (DAG) represents this pipeline using a declarative human-readable language. Each node of this DAG can be a general FL process, like FL training, or one of the other submodules, like FATE Serving.

The last two submodules of FATE are FATEBoard [39] and KubeFATE [43]. FATEBoard is a visualization tool to help models exploration and understanding. It communicates with FATEFlow to get the FL job pipeline and present it to the user visualize. It also monitors the execution and reports the model metrics and charts to compare different FL models. Finally, KubeFATE [43] manages the FL workload using Docker containers and the container orchestrator Kubernetes and deploys the application in a real scenario. It uses public container images of each FATE submodule and can deploy FATE in a single-machine or a multi-party scenario. Although following the instructions, we could not make FATE work in any supported scenario.

2.4.5 Flower

Most tools presented in Table 2.1 focus on FL simulation with homogeneous clients. Flower is a FL framework proposed by Beutel *et al.* [8] that focuses on the FL execution with heterogeneous clients on simulation and real-world scenarios. The authors developed Flower aiming at five objectives, with Flower being: ML framework-agnostic, client-agnostic, expandable, accessible, and scalable.

Flower allows a FL practitioner to use any ML framework underneath it (TensorFlow, PyTorch, or a custom one) and execute the FL application in several client environments, with different operating systems or hardware settings. Besides, Flower is open-sourced, allowing researchers to expand its code to implement new FL algorithms and architectures. Beutel *et al.* also created Flower in a modular way, as showed in Figure 2.7, which helps the accessibility of this tool. With Flower, a final user needs only to implement a few functions to transform a regular ML application into a federated one. Finally, the authors reached clients' scalability using the gRPC protocol [67], a high-performance implementation of the Remote Procedure Call (RPC) protocol that supports the communication among several tasks with minimum overhead, as presented by Beutel *et al.* with 1000 clients in Flower [8].

Figure 2.7 presents the modular design of Flower showing which functions are already implemented by Flower in white (Framework Code) and which the user needs to implement to create a FL application in blue (User Code).

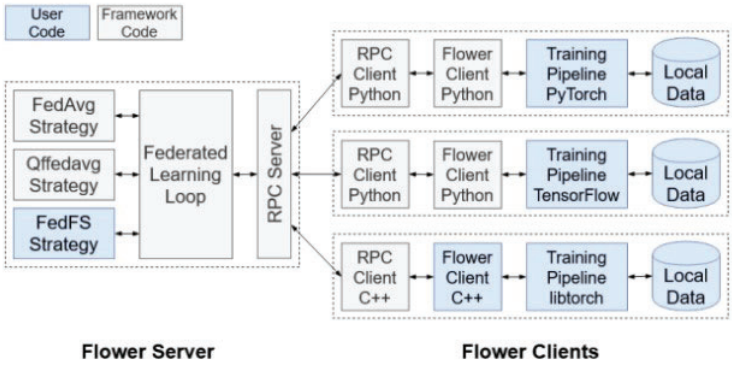


Figure 2.7: Architecture of Flower. Figure from [8].

Flower implements some FL algorithms on the server, like FedAvg [92], FedProx [85], and Q-FedAvg [86]. So, a final user can use any of these to translate his/her ML application to a federated scenario. However, if researchers want to develop a new FL algorithm, they can create a new Strategy for Flower with their proposal.

On the client side, Beutel *et al.* used the Python programming language to implement the client module. So, a FL practitioner using Python needs only to implement the training and evaluation functions of the clients through a ML API (TensorFlow, PyTorch, or a custom one) to execute his/her application in a federated scenario. If the practitioner uses another programming language, he/she needs to create the client module using the existing communication module as gRPC supports different programming languages (*e.g.*, C++, Go, Java, PHP, and others).

We consider Flower as our FL tool due to its simple architecture to execute FL in real and heterogeneous environments.

2.5 Cloud Computing

Cloud Computing is a computational paradigm that provides resources and services over the Internet with low cost, high availability, and in a fast, flexible, and scalable way [25]. Over the years, many authors defined Cloud Computing:

- According to Armbrust *et al.* [4] and Dikaiakos *et al.* [28], Cloud Computing represents a two-part paradigm: the applications offered as services through the Internet to a final user and the hardware and software located in physical data centers managed by a provider;
- Foster *et al.* [47] define Cloud Computing as a highly distributed computational

paradigm in which multiple resources are offered on-demand to external users by virtualization through the internet.

We can misunderstand Cloud Computing with a computational grid, especially with the first definition [134]. However, the main difference is the on-demand offering presented in the second definition. It means that the user will have as many resources as he/she wants and only pays for the resources used (pay-as-you-go scheme) [76]. Moreover, the United States National Institute of Standards and Technology (NIST) presented five essential characteristics to define Cloud Computing [9], and this is the mainly used definition of Cloud Computing. These characteristics are:

- On-demand self-service: users request the resources when needed to each cloud provider and without human interaction;
- Broad network access: users can access their allocated resources through a variety of platforms, such as mobile phones and computers;
- Resource pooling: providers pool their resources to allocate multiple consumers requests. Users do not know the exact location of allocated resources but can have control of a higher level of it when choosing the region in which they are;
- Rapid elasticity: Allocated resources can upscale or downscale automatically and quickly. It translates as a sense of infinite resource capability to the final user; and
- Measured service: Cloud systems automatically control their resources using metrics accordingly to the type of service provided. These metrics help report the resource usage to the cloud provider and the user.

One of the technologies behind the success of clouds is virtualization. It is the process of sharing computational resources (such as CPU, storage, and network) isolated from the physical hardware. This virtualization process reduces the inefficiency in the allocation and distribution of resources [70]. Moreover, there are several technical and economic advantages in using clouds over other distributed platforms, such as grids and clusters, as they combine the virtualization and scalability in a service model viable for both client and provider [76]. Some advantages of cloud computing are availability, monetary cost savings, reliability, and service integration [70]. According to Hashem *et al.* [70], in the cloud, the cost savings are related to automation and data processing and to the whole process of acquiring and maintaining the infrastructure and managing it.

Regarding the implementation of a cloud, there are four possible types of cloud environments [93]:

- Private cloud: a single organization manages this cloud environment and uses its data centers to provide resources through the Internet only to its employees and partners;
- Community cloud: one or more organizations in a community that shares some common interest (*e.g.*, economic issues or mission) provide this environment accessed only by a part of consumers from these organizations;
- Public cloud: a large company or organization offers computational resources to the general public in this cloud environment. There are several examples of public cloud providers, Amazon Web Services [108], Google Cloud Provider [51] or Microsoft Azure [5]; and
- Hybrid cloud: this environment combines the above cloud environments (private, community, and public). An organization can use a private cloud to execute its everyday tasks but allocates additional resources from a public cloud when necessary to handle any workload peaks.

In this thesis, we focus on public clouds. These cloud providers offer services with different levels of abstraction, which falls into three categories [70, 9]:

- Software as a Service (SaaS) is the level where applications are offered to the final user through the Internet. The user uses any web browser to access these applications free of charge or charged for their use. Some examples are Office 365 [99], Gmail [139], and Google Docs [140].
- Platform as a Service (PaaS) offers a development environment to implement, test and deploy user applications. This service deploys different resources to help the final user monitor and manage the development of user-made applications. AWS Elastic Beanstalk [116], Google App Engine [54], CloudFoundry [24], and Heroku [71] are examples of PaaS.
- Infrastructure as a Service (IaaS) consists of virtual hardware acquisition. The user can acquire from storage or processing capacity to full virtual machines in the IaaS model, and he/she has total control of software configuration and installation. AWS Elastic Compute Cloud (EC2) [111], AWS Simple Storage Service (S3) [107],

Google Compute Engine (GCE) [58], and Google Cloud Storage (GCS) [52] are some examples of IaaS.

In this work, the FL participating institutions store their data in one of the most used cloud providers: Amazon Web Services [108] or Google Cloud Provider [51], due to scope restrictions.

2.5.1 VM allocation

Cloud providers offer different markets to deploy their VMs. These markets have different prices, availabilities and guarantees. The main markets are the *on-demand* and the *preemptible/spot* ones. The *on-demand* market guarantees the VM availability from the request moment up to the user termination. However, it is usually more expensive. On the other hand, cloud providers make their spare capacity available with a huge discount so users with fault-tolerant applications can execute them paying less for the VMs[6,7]. These discounts can be up to 90% the on-demand price. However, the spare capacity of a provider is flexible and the VMs can be revoked by the cloud provider at any time.

2.5.2 Storage Services

Cloud providers have several storage services available to rent. Until June 2023, AWS offers a total of 12 storage services divided into eight categories [120]: (i) object storage with Amazon Simple Storage Service (S3) [107]; (ii) file storage through Amazon Elastic File System (EFS) [112] and Amazon FSx [113]; (iii) block storage with Amazon Elastic Block Store (EBS) [110]; (iv) high-speed cache with Amazon File Cache [122] (v) data migration using AWS DataSync [109] and AWS Snow Family [114]; (vi) hybrid cloud storage and edge computing with AWS Storage Gateway [118] and AWS Snow Family [114]; (vii) managed file transfer through AWS Transfer Family [119]; and (viii) disaster recovery and backup using AWS Elastic Disaster Recovery [117] and AWS Backup [115].

GCP offers ten different storage services divided into eight categories [56]: (i) object storage with Cloud storage [52]; (ii) block storage through Persistent Disk [62] and Local SSD [61]; (iii) archival storage using Cloud Storage [52]; (iv) file storage with Filestore [59]; (iv) data transfer through Data Transfer Services [63] and Transfer Appliance [64]; (v) backup and disaster recovery with Google Cloud Backup and DR [65]; (vi) mobile appli-

[6]https://repost.aws/knowledge-center/ec2-spot-instance-insufficient-capacity
[7]https://cloud.google.com/compute/docs/instances/preemptible

cation services using Cloud Storage for Firebase [27]; (vii) collaboration, communication, and file storage through Google Workspace [50]; and (viii) building artifacts with Artifact Registry [55].

Each storage service focuses on different needs in a company workflow. In this work, the FL clients need to store datasets that contain any file type (*e.g.*, text or image files). Besides, we assume that companies access the datasets frequently. So, we assume the companies use the object storage service from each provider.

Amazon S3 and Cloud Storage are the object storage services from AWS and GCP, respectively. They similarly represent the objects, using a two-level organization [107, 52]. At the superior level, they use buckets, structures similar to folders having a unique global name. These buckets help organize the data of different users, identifying and billing them accordingly. S3 restricts each bucket to a single region, and each account can associate up to 100 buckets. In Cloud Storage, the user can configure the bucket availability to a single cloud region, in two close regions (dual-region), or several regions spread in a larger area (multi-region). There is no limit in GCP associated with the number of buckets in a single account, but there are limitations regarding the bucket's name and creation rate [53].

Objects are the lower level of these two storage services. They contain the user stored data represented by a name and unique key used to access the object[8,9]. Both services have an upper limit to a single object size of 5TB [107, 53] and allow the user to create, change and read objects from a bucket using a single operation. However, if the user wants to rename or move the object to another place, it takes at least two, downloading the object to a local system and uploading it with the new name or in the new location.

Both cloud providers allow the user to choose the privacy level for each object. By default, AWS makes all objects stored in S3 private, allowing only access from the resource owner and account administrator[10]. If the user wants to let others see their data, he/she needs to grant access to each object explicitly. On the other hand, GCP does not assume any privacy level but requests the user to set the requested level of external access when uploading new files to Cloud Storage[11].

[8]https://docs.aws.amazon.com/AmazonS3/latest/userguide/UsingObjects.html
[9]https://cloud.google.com/storage/docs/naming-objects
[10]https://aws.amazon.com/s3/security/?pg=ln&sec=be#Access_management_and_security
[11]https://cloud.google.com/storage/docs/access-control

2.5.3 Multi-Cloud Environment

The use of multiple clouds by a single application defines a multi-cloud environment [131, 34, 72]. These clouds can be private or public, and they can communicate with each other or not, as presented in Figure 2.8.

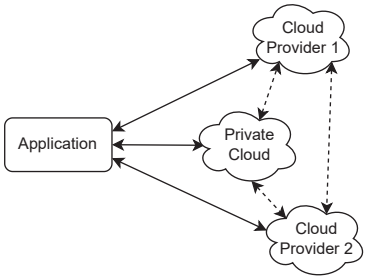


Figure 2.8: Multi-cloud environment using a private cloud and 2 public cloud providers.

There are different definitions of multi-cloud environments regarding the use of each independent cloud. Hong *et al.* [72] define that, in a multi-cloud environment, each task uses one cloud within the user application workflow. For istance, the user could use one of the Google Cloud Provider storage services and execute their application using the services of Amazon Web Services. On the other hand, Elkhatib [34] has a more general definition of a multi-cloud environment where the application uses a cloud manager that abstracts the specifics of each cloud, and the application does not need to know which cloud is used.

This second definition is more similar to a cloud federation. However, Toosi *et al.* [131] differ multi-cloud environments from a cloud federation, considering how much one provider is aware of the other providers. In a cloud federation, the cloud providers communicate among them to fulfill the application's needs. In a multi-cloud environment defined by Elkhatib [34], each provider does not know the others, and the application is responsible for managing the use of different cloud providers.

Federated Learning applications consist of independent institutions (or companies) collaborating to reach a globally unique model. We also assume that each company stores its private dataset in a single cloud provider, such as Amazon Web Services (AWS) [108] or Google Cloud Provider (GCP) [51]. Thus, one possible real collaboration scenario is described in Figure 2.9, in which there are 3 institutions (Inst. 1, Inst. 2, and Inst. 3) with their private datasets (D_{Inst1}, D_{Inst2}, D_{Inst3}) stored in different cloud providers.

We can map each company to a single client task in the whole FL application and, thus, our application fits the most restrictive definition of a multi-cloud environment by Hong

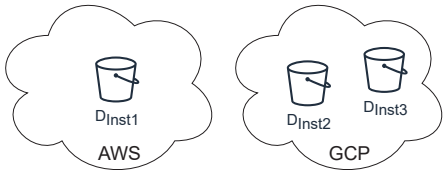


Figure 2.9: Example of a scenario with 3 companies using the Amazon Web Services (AWS) or the Google Cloud Provider (GCP) to store their private dataset.

et al. [72]. This multi-cloud environment presents challenges for executing FL regarding communication time and costs and FL server bottlenecks. The basis of Federated Learning applications is message exchange. Depending on which storage services each company chooses, we can have several options to allocate the related FL client. Besides, the location of the FL server is a concern when executing FL in multi-cloud as all clients need to send their updates to one single machine.

Chapter 3

Related work

There are two main scheduling perspectives related to Federated Learning (FL): one considers Cross-Device FL while the other deals with Cross-Silo FL. In the related literature, most papers treat Cross-Device FL. The main challenge of the Cross-Device FL architecture is handling low-powered devices that can disconnect at any time due to unstable network connections. Thus, most recent papers focus on either the proposal of new algorithms for server aggregation [85, 75, 142, 22] or the scheduling of clients to be sampled in a Cross-Device FL application [144, 141, 104, 20, 87].

Our work focuses on Cross-Silo FL architectures, where the clients are institutions with more computational power and stable network connection than Cross-Device clients. We assume that clients will always be available and all of them participate in each communication round. The scheduling problem of Cross-Silo FL applications consists of finding the optimal placement of each participating client, aiming at minimizing the execution and financial costs. Section 3.1 presents current work focused on Cross-Silo FL applications. The scheduling of Cross-Silo FL applications can be related to scheduling distributed Machine Learning (ML) applications as there is the assumption that workers are always available. Thus, Section 3.2 presents recent work on scheduling distributed ML applications. Section 3.3 introduces papers that consider distributed ML and FL executed on cloud environments.

Another important topic in the literature is resource management in single- and multi-clouds. Section 3.4 presents papers that executes different applications in the clouds.

3.1 Cross-Silo Federated Learning

Few papers focus on Cross-Silo Federated Learning to the best of our knowledge.

Rajendran *et al.* [102] present a FL approach to predict the disease risk related to tobacco and radon. They use health care electronic record data as input and implement their approach using two different ML models, a neural network (NN) and a logistic regression (LR). They asynchronously trained both models, in which one institution trains the model at a time. The authors show the results in a simulated environment and a real-world scenario.

Huang *et al.* [73] propose a method for Federated Learning focusing on training personalized models in each participating client. Instead of one unique global model, the FL server keeps a single model for each client and aggregates the weights of those participants whose models' weights are similar. The authors prove that their method improves the mean accuracy of all clients when there is no equal class distribution among all clients. Huang *et al.* [73] conducted their experiments in a single machine and used four benchmark datasets.

Li *et al.* [84] present an experimental study investigating the impact of different class distribution among clients in three different FL algorithms. In this study, they propose several data partitioning strategies. The authors created a benchmark dataset and concluded that there is no FL algorithm better in all scenarios.

3.2 Scheduling Distributed Machine Learning Jobs

The majority of works focus on determining the optimal number of parameter servers and workers per job when scheduling multiple ML jobs in limited resources [148, 6, 149]. They assume that all workers will have the same amount of data and allocate them as close as possible. These two assumptions are the main difference from our work, where each client stores their heterogeneous private datasets in different cloud regions or even cloud providers. However, some recent papers consider stragglers [95], the network performance [89], or locality awareness [147] being closer to ours.

Amiri *et al.* [95] consider transient stragglers among workers during the distributed ML training. Transient stragglers are workers with sporadic slowness due to temporary network disconnection, as an example. To handle them, the authors divide the central dataset into n small chunks assuming that each worker will compute over a number of them

sequentially. Then, the parameter server sends each piece to multiple workers and waits for k results from different chunks, $k \leq n$. The authors present the proposed system model and theoretical analysis for the minimum average completion time. Finally, they explain two scheduling schemes for chunks assignment to the workers, and their experimental results present better completion time than other algorithms in the literature. One aspect of this work is the data exchange between the tasks, prohibitive in a Federated Learning scenario. Thus, the whole FL execution needs to consider stragglers' execution time.

Liu *et al.* [89] focus on the impact of Optical Circuit Switches (OCSes) on multiple distributed ML training jobs execution in a data center or cluster. OCSes are energy-saving and high-performance switches that use light properties to make light passively flow in the desired direction. Hence, each port of an OCS switch only supports the communication to another port in a single way. If the application needs to send any message in another direction to another port, the OCS switch needs time to reconfigure itself. Thus, Liu *et al.* proposed two scheduling policies that consider the reconfiguration time. The Heaviest-Load-First (HLF) policy to schedule tasks within the same distributed ML job, and Shortest Weighted Remaining Time First (SWRT) to schedule different training interleaving their computation and communication phases. Besides handling possible network contention, the authors assume that all workers will have the same dataset size, class distribution and computational power, which is not always valid in a Federated Learning scenario due to privacy concerns.

Yu *et al.* [147] formulate an optimization model to allocate tasks from different distributed ML jobs into a set of physical machines regarding resource and locality-aware constraints, named the offline Distributed Machine Learning Resource Scheduling problem (DMLRS). DMLRS needs to know when each ML training job arrives and determines the best allocation scheme to all workers and parameter servers to each ML training job minimizing the communication time between them. They assume that the parameter server and the workers can co-locate in the same physical machine. They also consider that communication happens after training a few samples instead only at the end of a whole epoch. Besides, the authors propose an online scheduling algorithm based on a reformulation of DMLRS to a primal-dual and online problem. Despite presenting a locality-aware scheduling problem, this work has assumptions that cannot be valid in a Federated Learning scenario. First, they assume a central dataset, which is impossible in FL. Second, Yu *et al.* assume that the communication between workers and parameter server happens more frequently than in a FL scenario, where clients and the server communicate only after a few training epochs. Finally, the co-location of clients and servers

in Federated Learning can help malicious clients discover other clients' information using reverse engineering on the weights updates [132]. Thus, it is safer to allocate tasks to different resources in FL.

3.3 Distributed Machine Learning and Federated Learning on Clouds

Regarding the papers handling distributed Machine Learning on Clouds, most of them use cloud services as a tool to evaluate their scheduling proposal [95, 147] or to verify the central dataset integrity [151]. We found few research papers that consider aspects of Cloud Computing to propose their solution [150, 31, 30, 137]. However, the authors assume the equal division of the central dataset among workers, and they usually use a single region to allocate all cloud resources. These facts reinforce that distributed ML do not concern about data privacy and heterogeneity, which are the essence of Federated Learning.

Zhang *et al.* [150] consider Spot (or preemptible) cloud instances to train distributed ML tasks. Cloud providers offer their spare capacity as preemptible instances with a huge discount, but they can revoke them at any time. As there is no guarantee that all workers from a distributed ML job will be available throughout the execution, Zhang *et al.*study the convergence of classical algorithms to distributed ML training when the number of workers varies. They present a formula to obtain the optimal number of workers to deploy so that the algorithm does not diverge. They also validate their results using Amazon EC2 [111] Spot instances.

Duong and Quang [31, 30] present a system to execute distributed ML jobs into on-demand instances of a cloud provider called FC^2. They implement a simple web interface so that ML practitioners add their ML model characteristics and available budget, and FC^2 calculates the total number of workers to deploy into different instance types from a pool of available ones. The authors also implement two heuristics to recommend the instances, one considering the execution time per instance and the other the instances' bandwidth. Doung and Quang execute all results using Amazon EC2 [111] instances.

Wagenländer *et al.* [137] propose Spotnik to deal with instance revocations due to preemptible cloud instances in distributed ML training. They assume a decentralized training approach in which the workers communicate with each other in a pre-defined ring pattern to aggregate their results. Thus, in the communication phase, if a worker

w_i cannot reach its neighbor w_j, it assumes a revocation occurred, ignores the changes in the current iteration, removes w_j from the known topology, and propagates this change. Spotnik also adapts its training method accordingly to the current number of workers. If the number of workers is low, Spotnik uses synchronous strategies to calculate the aggregated results and, if the number of workers is high, Spotnik uses asynchronous ones.

To the best of our knowledge, few works tackle the problem of federated learning on clouds in the related literature.

Liu *et al.* [90] present a hierarchical FL architecture with mobile devices as clients. Their principle idea is the addition of an edge layer of servers. These edge servers aggregate the results of a fraction of the clients and send their aggregation to a central cloud server. Then, this central server aggregates the results of all edge servers and, consequently, all clients. The authors present the algorithm to this architecture and its convergence analysis in the paper. However, they only show simulation results to prove their convergence analysis.

Fang *et al.* [36] propose an architecture for Federated Learning in the cloud focusing on privacy. Their architecture consists of a central server on the cloud with clients that uses ElGamal encryption [33] to encrypt all messages. They present how the server and clients exchange their encryption keys and how the framework works. The authors compare their solution theoretically with other privacy-concerning deep learning methods. Besides, they simulate the FL environment in a local machine and present the accuracy, execution times, and messages size of each one.

The only found work that runs a FL approach in a cloud environment is the work of Rajendran *et al.* [102]. They presented the results of both implemented models in a scenario with two clients in a simulated environment and on the Microsoft Azure Cloud Databricks [29], an open-source tool for data engineering and collaborative data science, to exchange the ML model between two institutions.

In this work, we focus on synchronous FL training on VMs in the cloud and how we can schedule each actor (the server and clients) to minimize the total execution costs respecting the location of each client's dataset.

3.4 Resource management in clouds

There are several works approaching resource management on clouds (using single or multi-clouds) [124, 135, 35, 23, 146, 152, 128, 77, 11]. Some propose using Spot (preemptible) VMs to reduce costs and handle possible revocations [124, 135, 35, 152, 128]. However, most of them focus on Bag-of-Tasks applications, and others consider communication between tasks in their solution [124, 152].

Before November 2017, AWS used a per hour spot cost that fluctuates with the supply and demand of each VM type. Nowadays, AWS charges per second with much more stabler costs[1]. Despite the assumption of possible communications in the applications, Shastri and Irwin [124] proposed an cloud index price that aggregated all current Spot VMs prices to try to reduce the execution costs. They created a migration policy that was triggered whenever the current Spot VMs costs surpasses significantly this cloud index price.

Zhou *et al.* [152] proposed FarSpot, a framework to execute long-running HPC applications in AWS Spot market reducing application costs while guaranteeing the executing within a user-defined deadline. They proposed an ensemble spot price predictor, using Random Forest [10] and LightGBM [78] with dynamic weights to understand the behaviour of the new spot market price in AWS. They also defined a cost-aware deadline to distribute over the tasks and created an migration strategy that changes the Spot VMs reducing the task execution cost within the deadline. They evaluated FarSpot using the NPB benchmark[2] reducing the monetary cost by 32% compared to the state-of-the-art algorithms. However, they use only VMs without GPUs.

Besides the communication among tasks, these works uses only a single cloud and do not assume any synchronization barrier.

3.5 Summary

As presented in this chapter, most work on Federated Learning in the literature tackles the clients' unstable network from the Cross-Device architecture. In this scenario, the server needs to sample a fraction of clients in each communication round, as there are usually hundreds or thousands of devices connected. We focus on Cross-Silo FL, which assumes fewer clients and a better network connection in the present work. The clients

[1] https://aws.amazon.com/pt/blogs/aws/new-per-second-billing-for-ec2-instances-and-ebs-volumes/
[2] https://www.nas.nasa.gov/software/npb.html

are dedicated machines or clusters, and the server usually samples all to participate in each round as they are available throughout the entire execution. Moreover, we intend to create a generic Cross-Silo FL scheduler and, thus, we cannot take advantage of algorithms that focus on CNNs or RNNs.

Thus, Table 3.1 shows all previously described works regarding Federated Learning compared to ours. We describe the ML algorithm that each work implements (ML model), and whether the authors evaluated their proposal using simulation or real-world schemes (Environment). For the papers proposing scheduling solutions, we also present the minimization objective and the criteria used in their solution. If the authors do not mention which ML model they focus on, we use the N/S value.

Table 3.1: Recent work on Cross-Silo Federated Learning research area

Paper	**ML model**	**Environment**	**Minimization Objective**	**Criteria**
Huang *et al.* [73]	DNNs	Simulation	-	-
Li *et al.* [84]	CNNs	Simulation	-	-
Rajendran *et al.* [102]	NN and LR	Single-cloud	-	-
Our work	DNNs	Multi-cloud	Execution time and costs	Clients' dataset locality, communication and training time

Regarding the works on scheduling distributed Machine Learning (DML) jobs, they focus on three aspects that our scheduler could consider: non-persistent stragglers [95], network performance [89], and locality of the dataset and tasks [147]. Table 3.2 compares these works presenting their main characteristics. We show the minimization objective of each paper, how the authors solved their optimization problem (Solution), in which environment they tested the solution, and if the authors considered an online or offline problem. An online optimization problem is when new tasks can arrive during the execution, and, in the offline approach, all tasks' information is already available before the scheduling.

Table 3.2: Recent work on scheduling Distributed Machine Learning (DML) jobs

Paper	**Minimization Objective**	**Solution**	**Environment**	**Problem**
Amiri *et al.* [95]	Total delay in a single DML job	Two exact task scheduling schemes	AWS (Amazon EC2)	Offline
Liu *et al.* [89]	Total weighted job computation time of multiple DML jobs	Two heuristics (one focus on intra-job and another on inter-job scheduling)	Physical cluster	Online
Yu *et al.* [147]	Maximize fairness of multiple DML jobs	Approximation algorithm based on randomized rounding	Physical cluster	Online

Chapter 4

Case Study: Federated Learning in a biomedical application

In this chapter, we present our adaptation of a Bioinformatics application to a Federated Learning scenario. The purpose of this chapter and this application is to understand the FL execution in clouds and to validate our framework. Thus, the tumor-infiltrating lymphocyte classification problem is described in Section 4.1. Then, Section 4.2 presents the adaptation for the Federated Learning scenario. Section 4.3 discusses the results comparing the centralized and the FL approach. Finally, Section 4.4 discusses the execution of this application in a real multi-cloud scenario.

4.1 Tumor-Infiltrating Lymphocytes Classification

The use-case application analyzes high-resolution scanned tissue images to detect and classify spatial structures in the human tissue. It quantifies Tumor-Infiltrating Lymphocytes (TILs) that are a type of white blood cells in the immune system. Recent studies have shown that strong correlations exist between TILs density and spatial distribution with the overall patient survival in several cancer types [3]. Consequently, a quantitative analysis of tissue TILs is an important tool to describe tumor micro-environment and to understand the cancer-immune system in order to predict treatment response and prognosis.

The Whole Slide Tissue Images (WSIs) of scanned tissue may have in the order of up to 100K×100K pixels, making it costly, error-prone, and challenging for an expert (typically a pathologist) to extract quantitative information from hundreds to thousands of WSIs as used in research studies and clinical settings. The automated analysis of WSIs

captured by high-resolution tissue scanners is a rapidly growing area in image analysis and may enable extracting quantitative TIL characterization. These aspects have motivated the development of a deep learning application [106] built using Convolutional Neural Networks (CNNs) to classify and analyze TIL patterns on tissue images stained with hematoxylin and eosin (H&E). The application pipeline enables TIL classification and quantification with a posteriori identification spatial patterns.

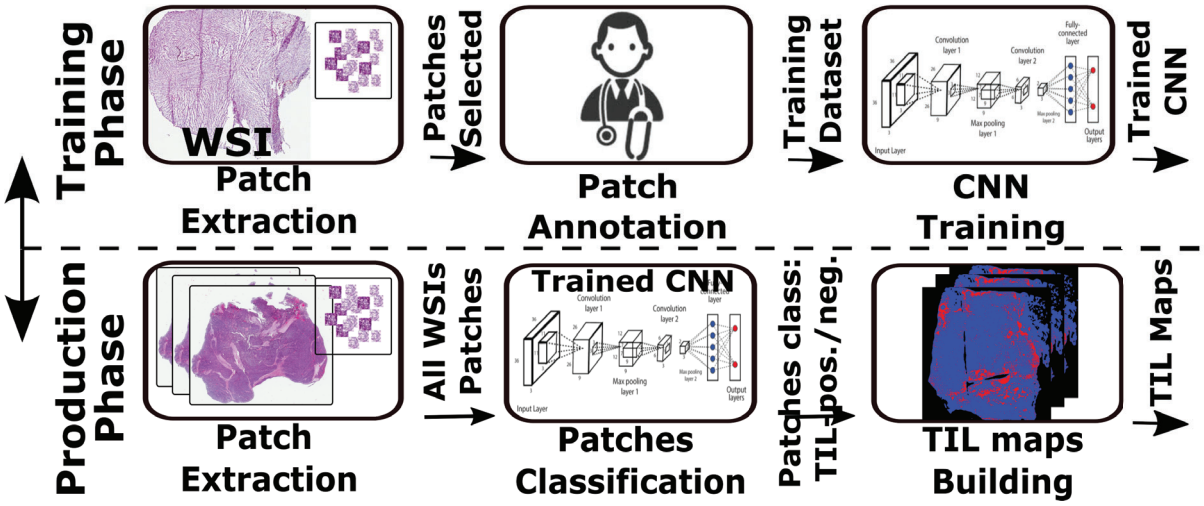


Figure 4.1: Use-case TIL analysis workflow. CNN is trained to identify TIL-positive/-negative patches annotated by a pathologist (top). The CNN model is then used to classify input WSI on a patch basis. The result is a TIL map presenting TIL-rich regions (red) in the input tissue. Source: [12].

The pipeline of the motivating TIL analysis application is presented in Figure 4.1. The CNN model training is showed in the top part of the figure. The process starts by extracting patches (50×50 square micron patches) from the input WSIs, which are then reviewed and classified by an expert pathologist as TIL-positive/-negative, and those patches are then used to train the CNN model. Further (bottom part), the trained model is applied to new unseen input WSIs to compute the desired TIL maps for a large number of WSIs in the production phase. After the TIL maps are computed (not shown in the image) different TIL characterizations may be carried out to perform the actual correlations with information of interest (e.g., treatment expected response and survival).

The model developed in the original application [106] was performed using a small-scale crowd-sourcing with several pathologists volunteering their time to create the training data and reviewing the generated TIL maps. This process was limited to a small group of experts and only publicly available WSIs were used because of the privacy issues involved in sharing those data across a larger group of experts, for instance, from different institutions. The use of Federated Learning improves this process by enabling

(i) an increase in the number of pathologists that could collaborate with annotation by restricting individuals to partitions of the data with a systematic data access control and (ii) the use of private datasets from multiple institutions to train the model without the need of sharing the data.

The centralized version of TIL classification implements different Convolutional Neural Network models, as described in [80]. In this thesis, we implement the two best models: the Inception-ResNet V2 CNN [127] and the VGG16 one [126]. The former model has an input layer of size 240×240×3 and the latter has an input layer of size $224 \times 224 \times 3$. After the input layers, both models has several convolutional and fully connected layers, ending with a output layer that determines if the TIL is positive or negative.

The centralized algorithm also splits the training dataset into two parts, using a user-defined percentage. The first part is used to train the model parameters and the second part validates the model and updates the hyper-parameters. Both training and validation steps are executed for each epoch. After finishing all epochs, the algorithm runs the model with the testing dataset to calculate the final model accuracy and other relevant ML metrics.

4.2 Tumor-Infiltrating Lymphocytes Classification with Federated Learning

In the proposed Federated Learning approach for TIL classification, we focus only on the CNN training step (third step of training phase in Figure 4.1). We assume that the patches were already extracted from the WSIs and classified by an expert. Also, we partition the centralized training and test datasets among all clients and assume that each client has access to two exclusive datasets, one used for training and the other for testing.

We use Flower to implement the FL approach as it has a simple design. Flower has a small limitation: the FL server needs to know the number of clients before starting its computation and waits for the availability of all of them before starting the communication rounds. If a client goes offline in the middle of any communication round, the server waits for the same or another client to be online to continue the execution. As we considered a Cross-Silo FL, the server has to sample all clients in each communication round. The FL client implements the same CNN model as the centralized version, either the Inception-ResNet V2 [127] or the VGG16 [126]. It also splits the training dataset into two parts, named training and validation datasets.

Each communication round is composed of training and test phases. In the training phase, each client trains and validates the local model for some epochs and sends the updated weights to the server. In the test phase, each client receives the aggregated weights, updates its model, and tests it against the testing dataset.

Note that the current implementation of Flower does not support stopping the FL training before a user-given number of rounds, which does not allow the execution of the early stopping feature of machine learning training, in which the model stops the training when the test accuracy starts to diverge even though the training accuracy continues to converge [49]. Thus, we did not use that feature in the centralized training as well.

4.3 Experimental Results

We present the results comparing the centralized application to Federated Learning Scenarios. We executed all tests in AWS, using the *g4dn.2xlarge* instance type. This instance is an interesting choice in those tests because it belongs to the accelerated computing family, as it has an Nvidia T4 Tensor Core with 16GB of memory, has a moderate price, its on-demand price is less than one dollar, and it is also available in the spot market.

Regarding the centralized approach, we selected 3793 patches as the training dataset and 2090 patches as the test dataset. They were obtained from the TCGA repository [96]. Both datasets are unbalanced, with 10% of TIL-positive samples in the training dataset and 20% of positive samples in the test dataset. We selected 10% of the training dataset to be the validation one. We used the best hyperparameters from the original experiments [80].

The Federated Learning approach divides the datasets among all clients homogeneously and each client has the same proportion of TIL-positive and TIL-negative samples in all datasets. Moreover, all clients select randomly 10% of their training dataset to be their validation one. We executed the clients in different *g4dn.2xlarge* instances and the server in a *t2.xlarge* one. We chose a cheaper instance for the server as it does not require a GPU. It only receives the weights from all clients and aggregates them. We deployed each task into a different VM to avoid unintended data access from other clients and reinforce the privacy issues.

We first present the results using the Inception-ResNet V2 (Section 4.3.1) as the CNN model and then we show the results using the VGG16 (Section 4.3.2). Finally, we present the results comparing the execution of the FL approach on on-demand and Spot VMs of

AWS (Section 4.3.3).

4.3.1 Varying the number of clients using Inception-ResNet V2 Model

The used metrics to evaluate our model in these tests are the loss and test accuracy, which are the standard metrics the server in Flower computes. The loss represents the distance between the predicted label and the correct one. The accuracy measures how many correct predictions are made by the model. Note that both training and test datasets have each a different accuracy, the validation accuracy for the former and the test accuracy for the latter.

We executed the centralized version of our algorithm for 25 and 50 training epochs. Table 4.1 shows the highest test accuracies found among 5 executions, the training execution times (hour:min:sec) and the total costs on an on-demand *g4dn.2xlarge* instance.

Table 4.1: Test Accuracy, Execution time, and Financial Cost for the centralized training on an on-demand *g4dn.2xlarge* instance

# epochs	**Acc.**	**Time**	**Cost**
25	79%	1:44:15	$1.29
50	32%	3:19:49	$2.48

In these two executions, the validation accuracy has increased until the last epoch, which configures the overfitting problem [49], where the model can only predict the training dataset and loses the capacity to generalize and predict unseen samples. This problem is avoided with the early stopping feature, which we did not implement to compare fairly with the Flower implementation, which does not have this feature.

Next, we present the results of the Federated Learning strategy in three different scenarios, with two, three, and four clients. In all of them, we attribute the centralized dataset equally divided to all clients. We executed the Federated Learning for five rounds with ten training epochs each in all scenarios, so each client executes for 50 epochs in total. We present the highest test accuracy found in our tests.

Table 4.2 shows the highest accuracy execution and the corresponding loss for the two clients scenario. Each client has 1896 training samples and 1045 test samples. The loss and test accuracy are presented for each communication round (Comm. round). Note that the server aggregates the loss and accuracy of all clients.

In this table, we can observe a decrease in the accuracy in the last communication

Table 4.2: Highest test accuracy found with two clients

Comm. round	Client 1		Client 2		Server	
	Loss	Acc.	Loss	Acc.	Loss	Acc.
1	0.3273	90.33%	0.2991	92.25%	0.3132	91.29%
2	0.2758	90.33%	0.2447	92.25%	0.2603	91.29%
3	0.3238	94.74%	0.3218	93.30%	0.3228	94.02%
4	0.4282	93.78%	0.4301	94.64%	0.4291	94.21%
5	0.5598	79.81%	0.5805	75.41%	0.5702	77.61%
Final	0.5598	79.81%	0.5805	75.41%	0.5702	77.61%

round. Our dataset is unbalanced as 90% of the samples are TIL-negative against only 10% of TIL-positive ones. As the clients choose randomly the samples of the training and the validation datasets, some TIL-positive patterns may be excluded from the training portion. We can avoid this problem by increasing the training datasets, using some image augmentation techniques, like scaling, cropping, rotation, and others [143].

Table 4.3 shows the highest test accuracy found in the execution of the three clients scenario. Each client has 1264 training samples and 696 test samples and we present the metrics found in each round.

Table 4.3: Highest test accuracy found with three clients

Comm. round	Client 1		Client 2		Client 3		Server	
	Loss	Acc.	Loss	Acc.	Loss	Acc.	Loss	Acc.
1	0.3233	89.80%	0.2719	92.10%	0.2739	91.98%	0.2897	91.29%
2	0.3493	89.80%	0.2775	92.10%	0.2808	91.98%	0.3025	91.29%
3	0.3800	89.80%	0.2942	92.10%	0.3005	91.98%	0.3249	91.29%
4	0.3280	89.80%	0.2646	92.10%	0.2715	91.98%	0.2880	91.29%
5	0.3040	93.82%	0.2867	95.11%	0.2903	95.13%	0.2937	94.69%
Final	0.3040	93.82%	0.2867	95.11%	0.2903	95.13%	0.2937	94.69%

In this case, we observe better final loss and accuracy than in the two clients scenario (Table 4.2). It may happen due to the way the centralized dataset was divided among the clients. Thus, the local datasets in this scenario can be more representative of the centralized dataset than the local datasets of the two clients scenario. In Table 4.3, we observe that the accuracy does not change in the first four rounds. However, the aggregated loss increases in the first three rounds and then decreases in the next one. The fact shows that the model is still learning from the training dataset and is not diverging from the test dataset.

Finally, Table 4.4 presents the highest test accuracy of the federated learning approach with four clients, where each client has 948 training samples and 522 test samples.

Table 4.4: Highest test accuracy found with four clients

Comm.	**Client 1**		**Client 2**		**Client 3**		**Client 4**	
round	**Loss**	**Acc.**	**Loss**	**Acc.**	**Loss**	**Acc.**	**Loss**	**Acc.**
1	0.3609	90.42%	0.3619	90.25%	0.3328	92.72%	0.3430	91.78%
2	0.3250	90.42%	0.3241	90.25%	0.2907	92.72%	0.3009	91.78%
3	0.2990	90.42%	0.2962	90.25%	0.2523	92.72%	0.2649	91.78%
4	0.3033	90.42%	0.2986	90.25%	0.2458	92.72%	0.2611	91.78%
5	0.2755	91.57%	0.2487	92.93%	0.2363	93.30%	0.2444	93.50%
Final	0.2755	91.57%	0.2487	92.93%	0.2363	93.30%	0.2444	93.50%

Comm.	**Server**	
round	**Loss**	**Acc.**
1	0.3497	91.29%
2	0.3102	91.29%
3	0.2781	91.29%
4	0.2772	91.29%
5	0.2512	92.82%
Final	0.2512	92.82%

We can see that the accuracy in this scenario does not change until the last communication round, which is the same behaviour that occurred in the three clients' scenario (Table 4.3). However, in the four clients scenario (Table 4.4), the aggregated loss is smaller than in the previous case, although clients 1 and 2 have increased their local losses in the fourth round. This occurs due to the collaboration among clients of a federated strategy, which is one of the advantages of using federated learning. The collaboration leads to an indirect influence of all clients' data in each client's model.

When compared to centralized training, all federated learning scenarios yielded better accuracy. The 50 epochs' centralized training obtained a final accuracy of 32% (Table 4.1). While the federated learning approach with 5 communication rounds of 10 epochs each, 50 epochs in total, presented a final accuracy of 77.61% with 2 clients (Table 4.2), 95.13% with 3 clients (Table 4.3) and 92.82% with 4 clients (Table 4.4). In FL, the server aggregates the weights of all clients at each communication round. Thus, the training samples from all clients influence the local model of each other client. This fact avoids the overfitting problem found in the centralized training approach, once samples from one client may only influence the models of other clients without participating in a global training as in the centralized approach.

To sum up the previous results, Table 4.5 presents the final test accuracy and loss found in each client and the server in all three scenarios.

Table 4.5: Final test accuracy found with all scenarios

Scenario	Client 1		Client 2		Client 3		Client 4	
	Loss	Acc.	Loss	Acc.	Loss	Acc.	Loss	Acc.
2 clients	0.5598	79.81%	0.5805	75.41%	-	-	-	-
3 clients	0.3040	93.82%	0.2867	95.11%	0.2903	95.13%	-	-
4 clients	0.2755	91.57%	0.2487	92.93%	0.2363	93.30%	0.2444	93.50%

Scenario	Server	
	Loss	Acc.
2 clients	0.5702	77.61%
3 clients	0.2937	94.69%
4 clients	0.2512	92.82%

Concerning the execution times, the federated learning implementations reduced them significantly when compared to the centralized approach. While the two clients scenario took 2 hours and 34 minutes (2:33:41), the three clients one executed in 1 hour and 42 minutes (1:42:00), and the four clients scenario lasted only 1 hour and 14 minutes (1:13:39). Compared to the centralized training with 50 epochs (Table 4.1), the two clients scenario reduced the execution time by 23% and the four client one by 63%.

4.3.2 Varying number of communication rounds using VGG16 Model

Now, we use the VGG16 model to compare the centralized approach executing 50 training epochs with 4 different scenarios of Federated Learning. All scenarios have 4 clients, as it was the best configuration in the previous experiments, but we varied the number of communication rounds within 50 training epochs in total. The first scenario has 25 communication rounds with 2 local training epochs each (25 comm. rounds). The second one has 10 communication rounds with five local epochs each (10 comm. rounds) while the third one has five communication rounds with 10 local epochs each (5 comm. rounds). Finally, the last scenario has only two communication rounds with 25 local training epochs each (2 comm. rounds). We also added more ML metrics in our FL approach to understand it better.

Table 4.6 compares the results of the centralized approach and all scenarios. The columns show the model accuracy, precision, specificity, and total execution time and

cost. The model accuracy is the percentage of well-predicted test samples, regardless of their true class. The precision is the percentage of true positives in all samples predicted as positive. The specificity shows the opposite, how many true negative samples were correctly predicted within all samples predicted as negative. The total time considers the moment from the request of the first VM up to the moment the last VM is terminated. Finally, the total cost includes the cost regarding all clients and the server for each FL scenario.

Table 4.6: Comparison among centralized approach and different number of communication rounds with 4 clients in Federated Learning

Scenario	Accuracy (%)	Precision (%)	Specificity (%)	Total exec. time	Exec. costs
Centralized	79.50	50.50	81.00	3:24:36	$2.55
25 comm. rounds	91.29	-	91.29	2:31:13	$7.92
10 comm. rounds	91.41	87.51	91.44	1:32:38	$4.80
5 comm. rounds	91.20	49.31	91.63	1:09:13	$3.53
2 comm. rounds	91.29	-	91.29	0:50:56	$2.59

We can observe that the accuracy of all Federated Learning scenarios is better than the centralized one. As our dataset is unbalanced, when the FL approaches predict all samples as negative, they classify correctly most of the samples. For the second and third FL scenarios, we observe that the model yields a better precision and specificity than the centralized approach, which explains the better accuracy. We can observe that in the first and the last scenario the model did not predict any samples as positive. In the scenario with 25 communication rounds, each client trains for only two epochs per round, and, thus, the local models do not learn much from the data before communicating. So, at each communication round, the global model update does not result in significant improvements in the model. Consequently, in this first FL experiment, we obtained the worst results.

When we compare the centralized approach with the best Federated Learning scenario (10 communication rounds with 5 training epochs each), we observe a decrease in the execution time by 55% with a corresponding cost increasing by 46%. Moreover, all metrics presented better results. Note that, although we used four GPU instances in FL, instead of only one, as in the centralized case, the costs increased only 46%. This happened because each client had a reduced dataset and spent much less time to finish. On the other hand, we did not reach a linear speedup due to the communication overhead.

4.3.3 Federated Learning on On-demand and Spot Instances

In this test, we compared the execution of all federated learning scenarios using on-demand and spot instances. Here, we used the Inception-ResNet v2 model to compare the scenarios with different number of clients. So, we executed the FL approach in three different scenarios: with two, three, and four clients. We executed the FL application for five rounds with ten training epochs each.

Regarding the execution time, in the two clients scenario, the on-demand execution was almost 2 hours and 34 minutes (2:33:41), while the spot execution was only 2 hours and 10 minutes (2:10:09). In the three clients scenario, the on-demand instances executed for 1 hour and 42 minutes (1:42:00) and the spot ones executed for 1 hour and 38 minutes (1:38:24). For the four clients scenario, both executions were around 1 hour and 14 minutes, with the on-demand one finishing faster (1:13:39) than the spot execution (1:14:43).

We can observe that the difference in the execution times between both markets varies according to the scenario. This difference is around 24 minutes in the two clients scenario and is around 3 minutes in the three clients scenario. The basis of the Federated Learning paradigm is messages exchange between clients and the server. If the server is in a physical cluster far from a single client, their communication can delay the whole training process.

Table 4.7 shows the costs for each client, for the server and the total execution cost (Total).

Table 4.7: Execution time and costs using spot and on-demand instances

Scenario	**On-demand**				**Spot**			
	Exec. time	**Costs**			**Exec. time**	Costs		
		Client	**Server**	**Total**		**Client**	**Server**	**Total**
2 client	2:33:41	$1.89	$0.47	$4.26	2:10:09	$0.59	$0.15	$1.32
3 client	1:42:00	$1.25	$0.31	$4.06	1:38:24	$0.36	$0.09	$1.17
4 client	1:13:39	$0.89	$0.22	$3.79	1:14:43	$0.27	$0.07	$1.15

The use of Spot VMs can decrease significantly the total execution cost. This reduction is 68.96% for the two clients scenario, 71.10% for the three clients scenario, and 69.66% for the four clients scenario. The reduction is the main advantage of using spot instances. However, in our tests, the VMs were often revoked by the cloud provider, and the *g4dn.2xlarge* was difficult to allocate as most of the time there was no spot capacity

available. This happens when all available VMs of this type are allocated to other users, and there is no spare capacity for spot instances.

4.4 Execution in a Multi-Cloud Environment

The experiments in the previous section were conducted to understand the behaviour of our use-case application when using Federated Learning. From these experiments, we observed that the best configuration is using 4 clients with 10 communication rounds and 5 epochs in each round. From now on, we assume this configuration for all tests using this application.

In this section, we analyze the execution of our application in a multi-cloud scenario, where each client stores their dataset in different cloud providers to understand the impact of different providers in the execution time and costs in our use-case application. To this matter, we executed both server and clients into different VMs of AWS and GCP.

Regarding the application execution parameters, we executed the FL application with 4 clients and 10 communication rounds using 5 training local epochs in each. We added log messages into the FL framework to get the time the server and clients send and receive each message. From these log messages, we computed the FL synchronization time and the computation time of each client. The FL synchronization time is the sum of three times per round: (1) the initial sync time, (2) the aggregation sync time, and (3) the test sync time. The initial sync time is the time between the server sending the first message with the current weights and the last client receiving it. The aggregation sync time is the time between the first client sending the updated weights back to the server and the last client receiving the final weights, which includes the server aggregation in step 3 (Section 2.3). The test sync time is the time between the first client sending the evaluation metrics and the server presenting the global metrics. The computation time of each client is the sum of the training and the testing steps in each client, including the time to access the dataset. We show here only the longest computation time.

Concerning the cloud environments, we selected the main region of each cloud provider to store the datasets and deploy the VMs. We chose North Virginia for AWS (*us-east-1* region) and Iowa for GCP (*us-central1* region). We executed each client in a VM with 8 vCPUs, 32GB of memory, and an Nvidia T4 Tensor Core GPU. We used Amazon S3 to store the datasets in AWS and Cloud Storage to store them in GCP. We chose the Premium Network Tier of GCP to use the high-speed internal network as much as

possible. The cloud region, type of VMs (client and server), VM price (client and server), and network costs are summarized in Table 4.8 for the two cloud providers.

Table 4.8: Cloud region, VM and cost for both client and server and network transfer costs in each cloud provider used

	Region	Client		Server		Network Costs ($ per GB)
		VM	Cost ($ per hour)	VM	Cost ($ per hour)	
AWS	N. Virginia (*us-east-1*)	*g4dn.2xlarge*	0.752	*t2.xlarge*	0.1856	0.090
GCP	Iowa (*us-central1*)	*n1-standard-8*	0.730	*e2-standard-4*	0.13402	0.120

4.4.1 Execution times and Financial Costs

Table 4.9 presents the averages of three executions in different scenarios, where the costs are presented in $, and the time in hours, minutes and seconds. Our initial tests aimed at evaluating the execution times and financial costs when the complete FL application was executed on a unique cloud provider. As can be seen in S2, GCP presented better results than the AWS ones (seen in S1). Even if more expensive than the standard one, the choice of the Premium network showed to be a good option when compared with the AWS results. The second set of tests evaluated two cases: (i) the server, three clients, and their corresponding data sets on GCP, and one client and the corresponding data set on AWS, case S3; and (ii) all clients and the server on GCP, but one data set allocated on AWS, case S4. This last test showed that the allocation of the client in the same cloud provider of the dataset is the best option, in this case, even considering that GCP has presented better results than AWS in the other cases.

Table 4.9: Average times and costs of 4 scenarios: S1- all FL application on AWS, S2- all FL application on GCP, S3 - 3 clients and 3 data sets on GCP, 1 client and 1 data set on AWS, S4- 4 clients and 3 data sets on GCP and one data set on AWS.

Scenarios	Total time	Total cost $	Sync. time	Message exchange cost $	Computing time	VM cost $
S1 (AWS)	1:28:32	10.51	0:34:47	5.80	1:17:55	4.71
S2 (GCP)	0:36:54	9.61	0:04:35	7.73	0:29:44	1.88
S3 (3 GCP,1 AWS)	1:25:22	12.04	0:56:58	7.57	1:15:42	4.47
S4 (4 GCP,1 DS AWS)	1:48:21	13.25	1:18:26	7.73	1:45:20	5.52

Finally, we considered two scenarios where two datasets and two clients were allocated to each cloud provider. In the first case, the server was allocated on AWS, and in the second case on GCP. We observed similar execution times in the two cases, despite the

use of the Premium Tier of GCP, with a 12% increase in the total costs using the server in GCP as the Premium Tier of GCP are more expensive than AWS' network.

From the experiments in this chapter, we can observe that the misplacement of the server or the clients can lead to higher execution times, higher costs, or both. Thus, in this thesis, a framework to optimally select the VMs in which all tasks (server and clients) execute while preserving the whole and correct execution of the Federated Learning even in the presence of revocations is proposed.

Chapter 5

Proposed Models and Framework Architecture

In this chapter, we present the system and application models considered in this thesis and the general implementation architecture adopted by Multi-FedLS. Thus, the chapter is divided into two sections. Section 5.1 describes all variables regarding the FL application and the Multi-cloud environment, while Section 5.2 describes the overall architecture details of Multi-FedLS.

5.1 FL Application and Multi-cloud Environment Models

As described in Chapter 2, FL applications are distributed algorithms, composed of a set of clients and a server with communication barriers along the execution. Let C be the set of clients and s be the server of an FL application.

In this thesis, an FL application executes a set of rounds, each one divided into five steps. In the first step, the server sends a message containing the model weights, named s_msg_{train}, to all participating 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). Next, in Step 4, each client updates their 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 present the global metrics. Figure 5.1 illustrates those execution steps in a single round.

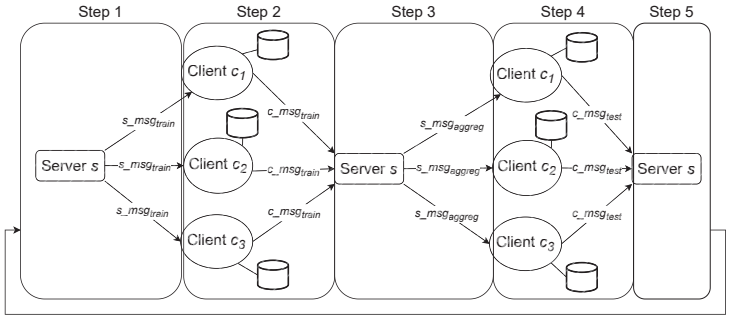


Figure 5.1: Steps of one round of a Federated Learning Application.

Regarding the environment, commercial clouds usually have their physical infrastructures spread in different regions, which are independent and isolated geographic. There are currently 31 regions among the globe in AWS [123] and 37 regions in GCP [66].

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 with it, a set of regions R_j and a fixed monetary cost $cost\_transfer_j$ (in \$ per GB) to send any message from any of its regions 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) and each region $r_{jk} \in R_j$ has its regional limits of available GPUs ($N\_L\_GPU_{jk}$) and vCPUs ($N\_L\_CPU_{jk}$).

Each region $r_{jk} \in R_j$ has a set of available instance types, V_{jk}, that can be deployed within the region, 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) $cost_{jkl}$. Figure 5.2 presents an example of two providers p_i with two regions (r_{ij} and r_{im}), and p_p with one region (r_{pq}) and each region with a different set of VMs.

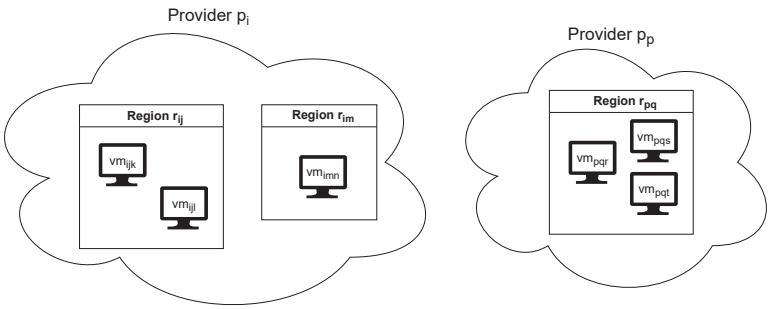


Figure 5.2: Two providers with regions, each one with a set of virtual machines.

Table 5.1 summarizes all variables of both models.

Table 5.1: Notation and variables of application and environment models.

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}
N_GPU_j	Number of available GPUs in the provider p_j
N_CPU_j	Number of available vCPUs in the provider p_j
$cost\_t_j$	Cost (in $ per GB) of sending a message from provider p_j
$N\_L\_GPU_{jk}$	Number of available GPUs in region r_{jk}
$N\_L\_CPU_{jk}$	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

5.2 Multi-FedLS architecture

Our framework takes advantage of the communication rounds of an FL application. From the experiments in Chapter 4, we observed that every round, except the first one, has similar execution times. Thus, we assume that all rounds have similar execution times reducing our scheduling problem and managing the resources for the whole FL application to scheduling and managing resources for a single FL round.

Figure 5.3 shows the proposed architecture for the *Multi-FedLS* framework. It consists of four main modules: Pre-Scheduling, Initial Mapping, Fault Tolerance with monitoring, and Dynamic Scheduler.

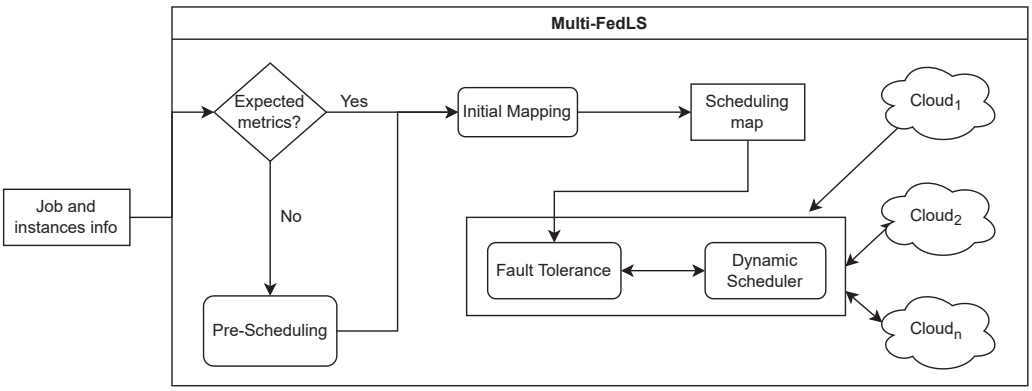


Figure 5.3: Architecture of Multi-FedLS.

The Multi-FedLS framework requires information from the FL application, the avail-

able VMs, and cloud providers to define the best mapping of clients and the server to VMs in the multi-cloud environment. It receives as input the number of clients, communication rounds, training epochs, location of each client's dataset, and the VM prices in each region of each cloud provider. Multi-FedLS expects these pieces of information inside two JSON files, one for the FL application information and the other for the environment information. Then, the Pre-Scheduling module uses dummy applications to obtain the communication delay between cloud regions and the execution time of clients in different VMs, if needed.

Receiving as input (i) the datasets locations, and (ii) the 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 using a mathematical formulation.

In order to reduce costs, Multi-FedLS uses preemptible VMs, that can be revoked at any time by the cloud provider. Consequently, our Fault Tolerance module adds checkpoint procedures to both the server and client sides. It also monitors the execution to trigger the Dynamic Scheduler module in case of an eventual revocation. Finally, the Dynamic Scheduler uses the same metrics as the Initial Mapping to choose the new VM to restart the affected server or client.

We implement Multi-FedLS using a master-worker architecture standard. The master is responsible for executing most modules of Multi-FedLS, such as the Pre-Scheduling, Initial Mapping and Dynamic Scheduler. On the other hand, the workers operate as asynchronous daemon applications running in the background on each deployed VM, executing the Fault Tolerance module and monitoring the VMs and tasks to communicate all status changes to the master. Workers are also an interface between the master and the deployed VMs. For example, when the master needs to configure a storage service in one specific VM, it sends a command to the corresponding worker, which executes all the configuration procedures.

The next chapter describes each of our four modules in details, while Chapter 7 describes the experimental results for each module and the whole framework.

Chapter 6

Multi-FedLS modules

This chapter describes the details of the Multi-FedLS modules. Section 6.1 presents the Pre-Scheduling module, and Section 6.2 presents the mathematical formulation used by our Initial Mapping module. The Fault Tolerance module is described in Section 6.3 while the algorithms to choose the new VM in our Dynamic Scheduler are described in Section 6.4.

6.1 Pre-Scheduling module

The Pre-Scheduling module executes several tests to obtain two slowdown metrics. Used as an input for the Initial Mapping, the metrics are; (1) the communication slowdown sl_comm_{jklm} between every pair of regions r_{jk} and r_{lm} and (2) the execution slowdown sl_inst_{jkl} in each VM vm_{jkl} in our environment.

Let the execution slowdown sl_inst_{jkl} be 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. Let sl_comm_{jklm}, the communication slowdown of the pair of regions r_{jk} and r_{lm}, be the ratio between the communication time of FL messages between these regions and the communication time in a baseline pair of regions.

To compute the communication slowdown, this module executes a dummy application with one server and a single client to measure the message exchange time between them. That application uses a floating point vector with the size of the VGG16 CNN model [126], which has more than 130 million 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 eight dummy keys and values, representing the possible machine learning metrics.

Regarding the execution slowdown, the Pre-Scheduling module executes a dummy application that simulates the execution of one client in the first two rounds of our use-case FL application in every VM of the environment. GPU instances are usually expensive, with several GPU architectures and machine configurations in cloud environments. Running all epochs of our FL application in all of them to calculate the average round execution time is costly. Malta *et al.* [91] propose 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 observed a similar effect in our experiments from Chapter 4, the average round execution time (sum of training and test execution time) had a standard deviation varying from 2.89% up to 13.43%. We considered this range an acceptable error margin in the expected times to reduce our Pre-Scheduling costs.

To send any computation to an NVIDIA GPU, we need to use the CUDA toolkit [98] to manage the communication with the accelerator. Any ML API (*e.g.*, TensorFlow, PyTorch, or a custom one) needs to locate, connect to and configure the GPU through the CUDA library to send the data and instructions to execute in the GPU. This configuration time can significantly impact the first round execution time depending on the size of the DNN model. Thus, we need the first two training and evaluation rounds of our application to compute the relative slowdown.

Once the two metrics (all communication slowdowns and all execution slowdowns) have been computed, we do not need to re-execute the dummy application in every *Multi-FedLS* execution, but only when there is a change in the regions or the VMs in the environment.

The baseline values for the current FL job are also computed by the Pre-Scheduling module. These values are (1) the execution time of each client in the baseline VM and (2) the message exchange time of the job model in the baseline pair of regions. The execution time of a client c_i is divided into the training execution time $train\_bl_i$ and test execution time $test\_bl_i$. The message exchange time is also divided into training and test, where $train\_comm\_bl$ and $test\_comm\_bl$ are the time spent sending messages during the training and the test respectively. We empirically collect these baseline values for the current FL job.

6.2 Initial Mapping module

The Initial Mapping module starts by calculating the expected communication and computation times of the current FL job in all regions and VMs. The expected communication time between regions r_{jk} and r_{lm} is given by Equation 6.1, 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 of the current FL job.

$$t\_comm_{jklm} = (train\_comm\_bl + test\_comm\_bl) \times sl\_comm_{jklm} \tag{6.1}$$

The expected execution time of a round of a client c_i executing on a provider p_j, region r_{jk} and VM 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 6.2.

$$t\_exec_{ijkl} = (train\_bl_i + test\_bl_i) \times sl\_inst_{jkl} \tag{6.2}$$

The complexity of scheduling tasks in distributed computing resources is proven to be NP-complete [133], which makes it challenging even in simple scenarios. Furthermore, the features of multi-clouds add to the difficulty. As a result, we have modeled our scheduling problem as a Mixed-Integer Linear Programming problem with two objectives: to minimize the total execution time (makespan) and the monetary cost. Additionally, the scheduling solution should comply with two constraints: the deadline (T) and the budget (B), as specified by the user. As we assume that the FL application executes through n_{rounds} with similar execution times, we obtain the maximum budget (B_{round}) and deadline (T_{round}) for an individual round by dividing B and T by n_{rounds}. Consequently, the scheduling problem can be formulated for a single round. Table 6.1 summarizes the variables defined in this chapter.

The proposed objective function (Equation 6.3) 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.

$$\min \alpha \times total\_costs + (1 - \alpha) \times t_m \tag{6.3}$$

Table 6.1: Notation and variables used in our framework.

Name	Description
B_{round}	Budget to a single FL round
T_{round}	Deadline to a single FL round
T_{max}	Time of a single FL round
$total\_costs$	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) between regions r_{jk} and 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)
T_{max}	Maximum possible makespan regarding all clients and possible VMs
$cost_{max}$	Maximum total cost considering T_{max}, all possible locations and VMs

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 6.4, and the message transfer costs ($comm\_costs$), described in Equation 6.5, where $comm_{jm}$, presented in Equation 6.6, 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 6.7.

$$vm\_costs = \sum_{c_i \in C} \sum_{p_j \in P} \sum_{r_{jk} \in R_j} \sum_{vm_{jkl} \in V_{jk}} (x_{ijkl} \times cost_{jkl} \times t_m) + \sum_{p_j \in P} \sum_{r_{jk} \in R_j} \sum_{vm_{jkl} \in V_{jk}} (y_{jkl} \times cost_{jkl} \times t_m) \quad (6.4)$$

$$comm\_costs = \sum_{c_i \in C} \sum_{p_j \in P} \sum_{r_{jk} \in R_j} \sum_{vm_{jkl} \in V_{jk}} \sum_{p_m \in P} \sum_{r_{mn} \in R_m} \sum_{vm_{mno} \in V_{mn}} (x_{ijkl} \times y_{mno} \times comm_{jm}) \quad (6.5)$$

$$comm_{jm} = (size(s\_msg_{train}) + size(s\_msg_{aggreg})) \times cost\_t_m + (size(c\_msg_{train}) + size(c\_msg_{test})) \times cost\_t_j \quad (6.6)$$

$$total\_costs = vm\_costs + comm\_costs \quad (6.7)$$

The monetary cost and the makespan in Equation 6.3 can have different minimum and maximum values. Thus, we need to normalize both objectives to have the same range of 0-1. To this end, we use T_{max} as the maximum possible makespan of a single FL round for the current application in the current sets of providers, regions, and VMs as well as $cost_{max}$ as the maximum cost in a single FL round, computed as Equation 6.8. We obtain $cost_{max}$ by the sum of two products: (i) the multiplication of the cost of hiring the most expensive VM by T_{max} and by the number of tasks; and (ii) the multiplication of the most expensive communication costs between providers ($comm_{jm}$) by the number of clients.

$$cost_{max} = \max_{p_j \in P, r_{jk} \in R_j, vm_{jkl} \in V_{jk}} (cost_{jkl}) \times T_{max} \times (|C| + 1) + \max_{p_j, p_m \in P} (comm_{jm}) \times |C| \quad (6.8)$$

Our mathematical formulation must respect a set of constraints (6.9 to 6.17). Constraints 6.9 and 6.10 ensure that our budget and deadline for an FL round are not violated. Constraints 6.11 and 6.12 guarantee that each task executes on a single VM $vm_{jkl} \in V_{jk}$ in region $r_{jk} \in R_j$ of provider $p_j \in P$.

$$total\_costs \leq B_{round} \quad (6.9)$$

$$t_m \leq T_{round} \quad (6.10)$$

$$\sum_{p_j \in P} \sum_{r_{jk} \in R_j} \sum_{vm_{jkl} \in V_{jk}} x_{ijkl} = 1, \forall c_i \in C \quad (6.11)$$

$$\sum_{p_j \in P} \sum_{r_{jk} \in R_j} \sum_{vm_{jkl} \in V_{jk}} y_{jkl} = 1 \quad (6.12)$$

Constraints 6.13 to 6.16 guarantee that the solution will not exceed the maximum number of GPUs and vCPUs of each provider. Constraints 6.13 ensure that the total number of available GPUs does not exceed the global limit, while constraints 6.14 limit the total number of available vCPUs in each provider. Constraint 6.15 has the same meaning as 6.13, but here the GPU limit is by region. Constraint 6.16 is the vCPUs limit constraints per region, similar to Constraint 6.14.

$$\sum_{c_i \in C} \sum_{r_{jk} \in R_j} \sum_{vm_{jkl} \in V_{jk}} x_{ijkl} \times gpu_{jkl} + \sum_{r_{jk} \in R_j} \sum_{vm_{jkl} \in V_{jk}} (y_{jkl} \times gpu_{jkl}) \leq N\_GPU_j, \forall p_j \in P \quad (6.13)$$

$$\sum_{c_i \in C} \sum_{r_{jk} \in R_j} \sum_{vm_{jkl} \in V_{jk}} x_{ijkl} \times cpu_{jkl} + \sum_{r_{jk} \in R_j} \sum_{vm_{jkl} \in V_{jk}} (y_{jkl} \times cpu_{jkl}) \leq N\_CPU_j, \forall p_j \in P \tag{6.14}$$

$$\sum_{c_i \in C} \sum_{vm_{jkl} \in V_{jk}} x_{ijkl} \times gpu_{jkl} + \sum_{vm_{jkl} \in V_{jk}} y_{jkl} \times gpu_{jkl} \leq N\_L\_GPU_{jl}, \ \forall p_j \in P, \forall r_{jk} \in R_j \tag{6.15}$$

$$\sum_{c_i \in C} \sum_{vm_{jkl} \in V_{jk}} x_{ijkl} \times cpu_{jkl} + \sum_{vm_{jkl} \in V_{jk}} y_{jkl} \times cpu_{jkl} \leq N\_L\_CPU_{jl}, \ \forall p_j \in P, \forall r_{jk} \in R_j \tag{6.16}$$

Constraint 6.17 ensures that t_m, the makespan of the FL round, will be at least equal to the total execution time of each client, which in turn is equal to the sum of the computational time t_exec_{ijkl} of the client in question, the communication time t_comm_{jklm} between this client's region and the server's one, and the aggregation time t_aggreg_{mno} of the server. The domain of the decision variables x_{ijkl} and y_{jkl} are determined in constraints 6.18 and 6.19.

$$x_{ijkl} \times y_{mno} \times (t\_exec_{ijkl} + t\_comm_{jkmn} + t\_aggreg_{mno}) \leq t_m, \forall c_i \in C, \\ \forall p_j, p_m \in P, \forall r_{jk} \in R_j, \forall vm_{jkl} \in V_{jk}, \forall r_{mn} \in R_m, \ \forall vm_{mno} \in V_{mn} \tag{6.17}$$

$$x_{ijkl} \in \{0, 1\}, \forall c_i \in C, \forall p_j \in P, \forall r_{jk} \in R_j, \forall vm_{jkl} \in V_{jk} \tag{6.18}$$

$$y_{jkl} \in \{0, 1\}, \forall p_j \in P, \forall r_{jk} \in R_j, \forall vm_{jkl} \in V_{jk} \tag{6.19}$$

6.3 Fault Tolerance module

As previously explained, *Multi-FedLS* takes advantage of Spot (or preemptible) VMs to reduce costs. The main drawback of these VMs is the revocation possibility at any time. AWS usually sends a two-minute notification to the selected VM to notify about its revocation while Google Cloud, after only 30 seconds, terminates the VM. Thus, the application must handle possible VM revocations and internal errors during the execution, which lead to faulty FL tasks. Note that the failure of a client has a different impact in the FL execution than the failure of the server.

During the FL execution, the Fault Tolerance module monitors all tasks and handles

eventual VMs revocations and runtime errors. When it detects any problem, it triggers the Dynamic Scheduler module (described in the next section) to select a new VM for the faulty task which was running in the revoked VM, informing if the task is the server or a client. After receiving from the Dynamic Scheduler the chosen VM to restart the task, the Fault Tolerance module launches it and continues to monitor all tasks within the FL application. When all tasks finish, the module stops monitoring the tasks and starts the VMs termination process.

Flower supports client failures during a FL round by making the server aggregate only the results received by the non-faulty clients. However, to start a new round, the FL server waits for a defined minimum number of available clients, which can be smaller than the total number of clients. In Cross-Silo FL, there are usually few clients and not considering one of them every round can compromise the learning outcomes. Thus, the FL server always waits for all clients to start the next round. Note that there is no need to manually restart the model at the clients because in the beginning of each round, the server sends the current weights to all clients (Section 5.1).

While Flower handles a client's error smoothly, it does not tolerate the failure of the server. Furthermore, when it happens, all clients stop their execution after receiving an error. *Multi-FedLS* handles possible server faults, using fault-tolerant techniques on both server and client sides.

Although Flower does not have an automatic way to save the model updates on the server side, it allows users to modify the server code to implement checkpointing during the server's aggregation phase[1]. Our framework takes advantage of this feature by asking the server to checkpoint every X rounds. Whenever a new checkpoint is stored on the server's VM local disk, it is transferred to another location asynchronously, which can be either a storage service or an extra VM.

On the client side, the aggregated weights received from the server in each round are stored in the VM's local disk, without sending it to another location. The clients' checkpoint is only used when the server's checkpoint is older than the clients' one. Thus, to restart the server, it is necessary to verify if the server or the clients has the latest checkpoint. If it is the server checkpoint, the saved weights are sent to the new VM before restarting the server task and the FL server just reads the weights. If it is the client one, the FL server restarts in the new VM and waits for any client to send its weights before starting the first round.

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

6.4 Dynamic Scheduler module

We denoted the faulty task as the task that was running in the revoked VM instance. The Dynamic Scheduler module adapts the objective function of the mathematical formulation of the Initial Mapping module and uses its value in a greedy heuristic to choose the new VM for the faulty task. It also takes advantage of the metrics computed by previous modules. Depending on the type of the faulty task, the Dynamic Scheduler computes differently the expected makespan and expected monetary costs, using a new instance to execute the faulty task. Algorithm 1 computes the new makespan using t as the faulty task (which can be s for the server or client c_i) and vm_{jkl} as the instance to be allocated to task t.

Algorithm 1 Makespan Re-calculation

Input: t, vm_{jkl}
  $max\_makespan \leftarrow -\infty$
  if $t = s$ **then**{Faulty task is the server}
    for $c_i \in C$ **do**
      $current\_vm_{mno} \leftarrow current\_map_{c_i}$
      $total\_time \leftarrow t\_exec_{imno} + t\_comm_{mnjk} + t\_aggreg_{jkl}$
      if $total\_time > max\_makespan$ **then**
        $max\_makespan \leftarrow total\_time$
      end if
    end for
  else {Faulty task is client c_t}
    $current\_server\_vm_{mno} \leftarrow current\_map_s$
    $max\_makespan \leftarrow t\_exec_{tjkl} + t\_comm_{jkmn} + t\_aggreg_{mno}$
    for $c_i \in C \backslash \{c_t\}$ **do**
      $current\_vm_{pqr} \leftarrow current\_map_{c_i}$
      $total\_time \leftarrow t\_exec_{ipqr} + t\_comm_{pqmn} + t\_aggreg_{mno}$
      if $total\_time > max\_makespan$ **then**
        $max\_makespan \leftarrow total\_time$
      end if
    end for
  end if
  return $max\_makespan$

The FL round makespan is defined by the client which takes longer to send its weights to the server. The delay is related to a longer training time and/or communication time. Thus, the makespan is obtained by computing the higher time a client needs to train and communicate with the server along with the server's aggregation time. If the faulty task is the server, then vm_{jkl} is the new server instance and the loop executes through all clients $c_i \in C$ to compute the total expected execution time of each client and store

the highest one as the new makespan. If the faulty task is a client, the current server instance is available in variable $current\_server\_vm_{mno}$ and the makespan of the faulty task is computed by using vm_{jkl} as the client's instance. Then, the loop executes through all clients $c_i \in C$, excluding the current task to check if any of the current clients have a higher makespan.

Algorithm 2 shows how to calculate the new cost, which receives the same inputs as the previous algorithm, i.e., the faulty task t and the new instance vm_{jkl}, and also the makespan in ms.

Algorithm 2 Financial Cost Re-calculation

Input: $t, vm_{jkl}, makespan$
  $total\_cost \leftarrow 0.0$
  if $t = s$ **then**{Faulty task is the server}
    $total\_cost \leftarrow total\_cost + cost_{jkl} \times ms$
    for $c_i \in C$ **do**
      $current\_vm_{mno} \leftarrow current\_map_{c_i}$
      $total\_cost \leftarrow total\_cost + cost_{mno} \times ms + comm_{jm}$ {$comm_{jm}$ as Equation 6.6}
    end for
  else {Faulty task is client c_t}
    $current\_server\_vm_{mno} \leftarrow current\_map_s$
    $total\_cost \leftarrow total\_cost + cost_{jkl} \times ms + comm_{mj}$ {$comm_{mi}$ as Equation 6.6}
    for $c_i \in C \backslash \{c_t\}$ **do**
      $current\_vm_{pqr} \leftarrow current\_map_{c_i}$
      $total\_cost \leftarrow total\_cost + cost_{pqr} \times ms + comm_{mp}$ {$comm_{mp}$ as Equation 6.6}
    end for
  end if
  return $total\_cost$

The total cost in a single FL round comprises the execution cost of each task and the message exchange costs of all clients to the server. Thus, in Algorithm 2 the execution cost of the server is firstly computed and then the execution and message exchange costs of all clients are also computed. The execution cost of an instance is the multiplication of the makespan and the instance cost while the message exchange cost is computed using $comm_{jm}$ (Equation 6.6), the cost to exchange the FL messages between the server's and the client's providers. Equally to the previous algorithm, when the faulty task t is the server, the server provider p_j is obtained from the input instance vm_{jkl}. Otherwise, if the faulty task is a client, the server's provider p_m is obtained from the current server instance.

Algorithm 3 implements a greedy heuristic for choosing a new instance for a server or a client after a revocation, receiving as input t, the faulty task, I_t, the current set of all

possible instances for task t and old_vm_{jkl}, the current revoked VM instance. Note that if the faulty task is the server then $t = s$ and $I_t = I_s$, the set of possible server instances, and if the task is client c_i, then $t = c_i$ and $I_t = I_{c_i}$, the set of possible instances for client c_i. Moreover, in the first execution of the Dynamic Scheduler, I_t is the sum of all V_{jk} sets for all regions $r_{jk} \in R_j$ for all providers $p_j \in P$.

Algorithm 3 Instance Selection

Input: t, I_t, old_vm_{jkl}
  $remove\_vm\_from\_set(I_t, old\_vm_{jkl})$
  $min\_value \leftarrow \infty$
  for each $vm_{mno} \in I_t$ **do**
    $makespan \leftarrow computes\_new\_makespan(t, vm_{mno})$ {Algorithm 1}
    $cost \leftarrow compute\_expected\_cost(makespan, t, vm_{mno})$ {Algorithm 2}
    $value \leftarrow \alpha \times (cost/cost_{max}) + (1 - \alpha) \times (makespan/T_{max})$
    if $value < min\_value$ **then**
      $new\_instance \leftarrow vm_{mno}$
      $min\_value \leftarrow value$
    end if
  end for
  return $new\_instance$

The first step in Algorithm 3 is to remove the revoked VM from the set of possible VMs. From previous experiments, we observed that once the instance type is revoked in a region in AWS, it cannot be reallocated in the same region immediately [18]. Thus, we assume this behavior to all cloud providers and remove the old_vm_{jkl} from I_t. Note that we replicate the search space of our framework to each task (server and clients), so a choice in a client does not influence the choice in the other. This is done as they can have different dataset sizes leading to different expected execution times which can correspond to different selections in both Initial Mapping and Dynamic Scheduler modules. After removing the revoked VM, our algorithm starts the loop iterating on all VMs from I_t to select the VM to restart task t. The makespan and cost are computed with Algorithms 1 and 2, respectively, using vm_{mno} as the instance for task t. After that, the sum of the makespan and cost is done by using the same weight as the objective function of the Initial Mapping module (Equation 6.3), α for the normalized cost and $1 - \alpha$ for the normalized makespan.

Chapter 7

Experimental Results

This chapter shows our experiments to validate each module in the framework: the Pre-Scheduling, the Initial Mapping, the Fault Tolerance, and the Dynamic Scheduler. First, we describe the applications used in Section 7.1 and the environment in Section 7.2. We present the values obtained by our Pre-Scheduling module in Section 7.3. We executed our experiments in AWS and GCP, mainly the ones validating our Initial Mapping module, and in CloudLab the other ones. Thus, Section 7.4 presents and analyses our experiments to validate the Initial Mapping module, mainly on AWS and GCP. Then, from Section 7.5 on, we execute our experiments on CloudLab, due to scalability issues. Section 7.5 describes the ones focusing on the Fault Tolerance module, and Section 7.6 presents the ones focusing on the Dynamic Scheduler. Finally, Section 7.7 presents a Proof of Concept that our *Multi-FedLS* framework works on a real multi-cloud environment.

7.1 Applications

Most of our experiments use our real-world use-case application, described in Chapter 4. We also selected two FL benchmarks from LEAF [21], a hub of benchmarks for different FL settings, mostly focused on Cross-Device FL (thousands of clients with fewer samples). We selected two datasets, the Shakespeare and the FEMNIST ones, and we adapted them to a Cross-Silo FL setting (fewer clients with more samples).

The Shakespeare dataset is based on the book *The Complete Works of William Shakespeare*, with each character of each play being a different client. We selected clients with more than 18000 samples combining the training and test datasets, which left us with 8 clients. Their training datasets vary from 16488 to 26282 samples and their test datasets vary from 1833 to 2921 samples. This dataset is used to predict the next character in

the sentence, and we implemented the reference model in LEAF [21], which uses an embedding of dimension 8 and a Long Short Term Memory (LSTM) network of two layers containing 256 units each. This LSTM network is a recurrent neural network that saves information from the previous sample to be used by the current sample. We selected the Shakespeare application as the clients have bigger datasets with a small model to train.

The FEMNIST dataset creates a federated scenario to the extended MNIST dataset, containing both handwritten letters and digits of size 28×28. In this FEMNIST dataset, each client has the handwritten characters of a single user. We selected users with more than 440 samples in total, which left us with 5 clients. After that, we replicated each client's datasets to double their number of samples. Thus, the training datasets of these five clients vary from 796 to 1050 samples and their test datasets vary from 90 to 118 samples. To predict the character in an image in the FEMNIST dataset, we create a more robust CNN than the reference one in LEAF [21] using 2 convolutional layers followed by 10 fully connected layers with 4096 neurons each. We selected this application as the clients have smaller datasets with a more robust model to train compared to the Shakespeare application. Both applications allow us to understand the behaviour of our *Multi-FedLS* with a wide range of different characteristics (small vs. big dataset; simple vs. complex model).

All three applications used FedAvg as the server aggregation method.

7.2 Experimental setup

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. All regions are inside the United States of America (USA).

Initially, we planned to have a testbed composed of several VMs with Nvidia GPUs in both AWS and in GCP, varying from the Kepler GPU architecture to the Ampere one. However, CUDA version 10.0, used by the use-case 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 7.1 summarizes the VM instance selection in both providers, giving also the amount of memory and cost per hour (obtained in May 2022) of each instance, and IDs that allow 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) [107] in the N. Virginia region (*us-east-1*) and the Cloud Storage [52] in the Iowa region (*us-central1*).

Both AWS nor GCP do not increase our GPU's quotas, maintaining only 4 simultaneous GPUs. To achieve scalability, we changed the environment to execute our experiments in CloudLab [32], a platform that allows users to simulate cloud environments. It behaves similarly to a cloud platform, with pre-defined instances types in clusters in at least three different US states[1]. However, CloudLab does not use virtualization as a real provider, it uses a *bare metal* approach to isolate the instances requested by a user from others. Moreover, CloudLab does not limit the number of vCPUs and GPUs per region allocated by users. On the other hand, they use a reservation system, in which users must specify which instances they want to use, and they will be available during all the scheduled reservation. CloudLab has 5 different clusters, with 2 of them in the same U.S. state. We simulate two different clouds splitting these 5 clusters in *Cloud A* and *Cloud B*. Each cluster simulates a different region inside the respective cloud.

Table 7.2 summarizes the instance selection in this environment. The VMs that have GPUs are vm_{326} with a P100 (12 GB of memory) and vm_{338} with a Tesla V100S (32 GB of memory). As CloudLab does not charge for its instances, we computed the on-demand price for each VM based on Google Cloud Provider's policy. We used the values of December 2022 of the computer-optimized instances of GCP to Cloud A, $0.03398 per

[1]https://www.cloudlab.us/hardware.php

Table 7.1: Instance types selected in AWS and GCP

Prov.	Region	VM	vCPUS	RAM (GB)	GPU	GPU mem. (GB)	Costs per hour ($)		ID
							On-demand	Spot	
AWS	N. Virginia (us-east-1)	*g4dn.2xlarge*	8	32	Tesla T4 Tensor Core	16	0.752	0.318	vm_{111}
		g3.4xlarge	16	122	Tesla M60	8	1.140	0.638	vm_{112}
		t2.xlarge	4	16	-	-	0.186	0.140	vm_{113}
	Oregon (us-west-2)	*g4dn.2xlarge*	8	32	Tesla T4 Tensor Core	16	0.752		vm_{121}
		g3.4xlarge	16	122	Tesla M60	8	1.140		vm_{122}
		t2.xlarge	4	16	-	-	0.186		vm_{123}
GCP	Iowa (us-central1)	*n1-standard-8* with Turing GPU	8	30	Tesla T4 Tensor Core	16	0.730	0.196	vm_{211}
		n1-standard-16 with Pascal GPU	16	60	Testa P4	8	1.360		vm_{212}
		n1-standard-8 with Volta GPU	8	30	V100 Tensor Core	16	2.860	0.857	vm_{213}
		e2-standard-4	4	16	-	-	0.134	0.040	vm_{214}
	Oregon (us-west1)	*n1-standard-8* with Turing GPU	8	30	Tesla T4 Tensor Core	16	0.730		vm_{221}
		n1-standard-8 with Volta GPU	8	30	V100 Tensor Core	16	2.860	0.857	vm_{222}
		e2-standard-4	4	16	-	-	0.134	0.040	vm_{223}

vCPU and $0.00455 per GB of RAM per hour, and general-purpose instances to Cloud B, $0.031611 per vCPU and $0.004237 per GB of RAM per hour. GPU's architectures have different prices per hour per GPU being $1.46 for P100, and $2.48 for V100. To compute the spot price, we assume a 70% discount on the on-demand one.

Table 7.2: Instance types selected in CloudLab

Cloud	Cluster/Region	VM	vCPUS	RAM (GB)	Costs per hour ($)		ID
					On-demand	Spot	
Cloud A	Utah	c6525-100g	48	128	2.213	0.664	vm_{311}
		c6525-25g	32	128	1.670	0.501	vm_{312}
		d6515	64	128	2.757	0.827	vm_{313}
		m510	16	64	0.835	0.250	vm_{314}
		xl170	20	64	0.971	0.291	vm_{315}
	Wisconsin	c220g1	32	128	1.670	0.501	vm_{321}
		c220g2	40	160	2.087	0.626	vm_{322}
		c220g5	40	192	2.233	0.670	vm_{323}
		c240g1	32	128	1.670	0.501	vm_{324}
		c240g2	40	160	2.087	0.626	vm_{325}
		c240g5	**40**	**192**	**4.693**	**1.408**	$\boldsymbol{vm_{326}}$
	Clemson	c6320	56	256	3.068	0.920	vm_{331}
		c6420	64	384	3.922	1.177	vm_{332}
		c8220	40	256	2.524	0.757	vm_{333}
		c8220x	40	256	2.524	0.757	vm_{334}
		dss7500	24	128	1.398	0.419	vm_{335}
		r650	144	256	6.058	1.817	vm_{336}
		r6525	128	256	5.514	1.654	vm_{337}
		r7525	**128**	**512**	**11.159**	**3.348**	$\boldsymbol{vm_{338}}$
Cloud B	APT	c6220	32	64	1.283	0.385	vm_{411}
		r320	12	16	0.574	0.172	vm_{412}
	Massachusetts	rs440	64	192	2.837	0.851	vm_{421}
		rs630	40	256	2.349	0.705	vm_{422}

7.3 Pre-Scheduling slowdowns

We computed the computational slowdown for all instances in Table 7.1 with two scenarios, when the clients datasets were stored in AWS and when they were stored in GCP. 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 7.3 shows the average values of each step (training and test) of the first (1º r.) and second (2º r.) round, and the computed slowdown (Sl) when the datasets are in AWS. We executed the client in each instance type three times and observed an average standard deviation of 10.96%.

Table 7.3: 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 7.4 presents the average values of each step (training and test) of the first (1º r.) and second (2º r.) round, and the computed slowdown (Sl) when the datasets are in GCP. We executed the client in each instance type three times and observed an average standard deviation of 11.93%.

Table 7.4: 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

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.

Now the values for the CloudLab instances. We computed the computational slowdown for all instances in Table 7.2 using vm_{321} as the baseline VM. We executed one client of the TIL use-case application, with 38 training samples and 21 test samples. We assume the datasets stored in Cloud_A. Table 7.5 shows the obtained slowdowns. Note that all VMs have different execution times, varying the slowdown from 0.04 and 2.33.

Table 7.5: Time of one client with five local epochs, dataset stored in Utah region of Cloud A

Cloud	Region	VM ID	Training time		Test time		Sl
			1º r.	2º r.	1º r.	2º r.	
Cloud A	Utah	vm_{311}	119.86	117.84	1.31	1.20	1.03
		vm_{312}	123.12	120.93	1.61	1.47	1.06
		vm_{313}	124.62	122.60	1.19	1.08	1.07
		vm_{314}	163.16	158.95	4.71	4.62	1.42
		vm_{315}	113.22	110.32	2.95	2.86	0.98
	Wisconsin	vm_{321}	119.89	112.83	2.30	2.22	1.00
		vm_{322}	139.04	131.74	1.93	1.96	1.16
		vm_{323}	133.41	130.83	2.75	2.73	1.16
		vm_{324}	119.05	110.45	2.23	2.12	0.97
		vm_{325}	137.21	131.63	2.14	1.87	1.16
		vm_{326}	16.37	4.53	1.44	0.62	**0.04**
	Clemson	vm_{331}	121.92	112.75	1.85	1.77	0.99
		vm_{332}	102.73	96.46	1.16	0.95	0.85
		vm_{333}	184.59	172.01	4.65	4.59	1.53
		vm_{334}	184.28	177.15	4.83	4.54	1.58
		vm_{335}	128.46	122.39	2.79	2.67	1.09
		vm_{336}	84.90	81.72	0.95	0.83	0.72
		vm_{337}	144.03	136.23	0.97	0.91	1.19
		vm_{338}	71.67	60.14	5.39	5.24	0.57
Cloud B	APT	vm_{411}	147.79	141.62	4.22	4.26	1.27
		vm_{412}	263.89	256.73	11.18	11.13	**2.33**
	Massachusetts	vm_{421}	94.23	92.42	1.26	1.20	0.81
		vm_{422}	112.44	103.59	1.91	1.75	0.92

Regarding the communication slowdowns, Table 7.6 shows the average times in seconds in all possible pairs of regions between AWS and GCP (Table 7.1) 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).

We also computed the communication slowdowns between the pair of regions in Cloud A and Cloud B in Table 7.7 using the pair of regions APT (Cloud B) and APT (Cloud B) as the baseline. The training phase exchanges a total of 2GB in messages and the test

Table 7.6: Communication times between each pair of regions in AWS and GCP. 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

phase exchanges a little more than 1GB in total. Note that the communication times vary a lot inside CloudLab, with the slowdown ranging from 0.37 to 24.73. As CloudLab is an emulator and not a real cloud provider, it does not use any level of virtualization to isolate the users' VMs. They allocate different physical machines to each user, which makes the network usage in CloudLab vary a lot depends on the region we execute.

Table 7.7: Communication times between each pair of regions in Cloud A and Cloud B. 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	
APT (Cloud B) & APT (Cloud B)	5.61	3.05	1.00
APT (Cloud B) & Clemson (Cloud A)	12.05	5.94	2.08
APT (Cloud B) & Mass (Cloud B)	106.90	54.51	18.64
APT (Cloud B) & Utah (Cloud A)	4.84	2.58	0.86
APT (Cloud B) & Wis (Cloud A)	16.19	7.64	2.75
Clemson (Cloud A) & Clemson (Cloud A)	5.36	2.91	0.95
Clemson (Cloud A) & Mass (Cloud B)	75.63	32.31	12.46
Clemson (Cloud A) & Utah (Cloud A)	11.39	5.34	1.93
Clemson (Cloud A) & Wis (Cloud A)	6.65	3.53	1.18
Mass (Cloud B) & Mass (Cloud B)	5.23	2.81	0.93
Mass (Cloud B) & Utah (Cloud A)	86.08	35.95	14.09
Mass (Cloud B) & Wis (Cloud A)	138.31	75.85	**24.73**
Utah (Cloud A) & Utah (Cloud A)	2.07	1.15	**0.37**
Utah (Cloud A) & Wis (Cloud A)	21.81	10.57	3.74
Wis (Cloud A) & Wis (Cloud A)	5.77	3.08	1.02

We validate these slowdowns when executing the Initial Mapping module in both AWS and GCP (Section 7.4.3) and CloudLab (Section 7.4.5).

7.4 Initial Mapping experiments

In these experiments, we use only VMs in the on-demand market and does not use any Fault Tolerance mechanism. We first show a scalability analysis of our mathematical formulation in Section 7.4.1. Then, we present theoretical scheduling results in Section 7.4.2 and practical results with 4 clients executing in a multi-cloud Platform in Section 7.4.3. We discuss these initial results in Section 7.4.4 and present CloudLab validation experiments in Section 7.4.5.

7.4.1 Analysis of the Scalability of the Proposed Mathematical Formulation

We used *Gurobi Optimizer* [68], a state-of-the-art solver for mathematical programming models, for solving the proposed formulation in our Initial Mapping module with an Academic License executing in a local machine with a Intel Core i5 CPU and 8GB of RAM. 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 7.1, we have created five synthetic VMs multiplying the execution time and the VM cost by different constants from 0.7 to 1.5. Thus, we have a total of 78 VMs, 54 with GPUs and 24 without any GPUs.

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 7.1 presents the relation between the number of clients and Gurobi's execution time.

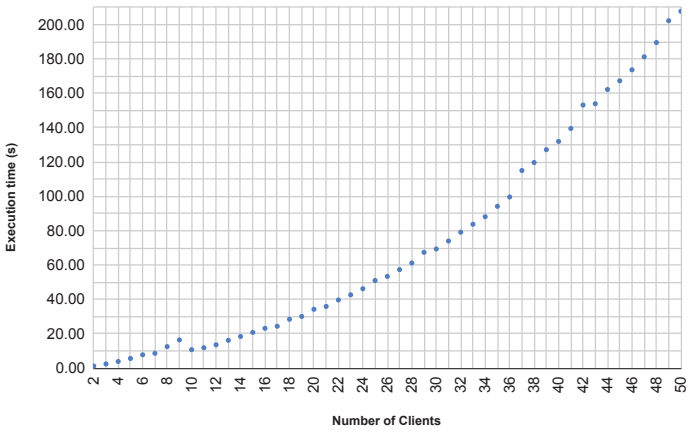


Figure 7.1: Relation between number of clients and Gurobi's execution time.

We observe in Figure 7.1 that our mathematical formulation can obtain the optimal solution, in a realistic time, with a large search space and a considerable number of clients.

7.4.2 Theoretical Analysis of the Initial Mapping module against User Random Selection

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 an 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 user-oriented 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 user-oriented 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 48.34% of the costs.

Table 7.8 summarizes these two scenarios. It shows the location of the datasets (Scenario), the optimal setup, the execution time and costs computed by the mathematical formulation, along with the user-oriented approaches described above, and the difference between the latter and the optimal setup. 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. Note that in all setups presented by our framework or user-oriented, each task (server and each client) executes in a different VM to prevent malicious clients from accessing the others' dataset.

Table 7.8: 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

7.4.3 Analysis of the Initial Mapping module 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 Chapter 4 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 are 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 the 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 the 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 7.9, and the configuration of each VM can be found in Table 7.1.

Table 7.9: 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}

We first present the execution time and the cost for a single round of FL to all setups in Table 7.10. The optimal values come from the mathematical formulation and the random scheduling setups are computed using the slowdowns from the previous subsections. We also show 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 7.10 that our mathematical model presents better results

Table 7.10: 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

in all scenarios with an average execution time reduction of 20.03% compared to the randomly selected scenarios and an average difference in monetary costs of -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 an 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 7.11 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%.

Table 7.11: 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

We can observe in the table that the computed differences between the optimal values and the random ones are close to the theoretical results (Table 7.10) but not equal. Besides, the real cloud differences in the second random setup of scenario AWS(4) are far from the theoretical ones. In [82], 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 not modelled found between Table 7.10 and Table 7.11.

7.4.4 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 environment. On the contrary, GCP instances presented a stable performance. We also observed that the time taken to access datasets allocated in AWS S3 (Table 7.3) was higher than the time to access the corresponding datasets in the GCP Cloud Storage (Table 7.4). Such behavior is coherent with other ones from the related literature. Leitner *et al.* [82] show 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. It can be explained by the difference in cost and execution time of VM with GPU. For example, AWS charges a little more in VMs with older GPUs, which takes more to execute. Thus, whenever the objective is to minimize only the costs, only the execution time or both equally, most experiments yielded the same optimal setup.

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 [138] 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 (up to 15%).

7.4.5 Validation of CloudLab environment

From now on, we execute our experiments with CloudLab to achieve scalability. Here, we validate the CloudLab environment using the same configuration as above to compare the results. The baseline execution time (training and test phases) for each client is 2765.4 seconds and the communication baseline is 8.66 seconds. We assume the transfer costs inside both clouds are the same as Google Cloud Provider, $0.012 per sent GB.

Table 7.12 shows the model output for the FL runtime and costs along with the values for the real execution. We emphasize that the values shown are the average of three executions.

Table 7.12: Validating CloudLab with Initial Mapping module using the TIL application

Setup	Model output		Real Execution	
	Runtime	Costs	Runtime	Costs
c_1 in vm_{326}, c_2 in vm_{326}, c_3 in vm_{326}, c_4 in vm_{326}, s in vm_{321}	0:22:38	$15.44	0:24:47	$16.18

We observed that the model has a similar execution time and cost compared to the real one for the TIL use-case application. The difference is 8.69% in the execution time and 4.53% in costs, validating this new test environment with the same setup used in [16].

However, we also noticed that our framework took longer to start the FL execution in all VMs compared to real cloud providers. We compared the time to execute the TIL application in CloudLab, in AWS, and in GCP, using the results presented in [16]. The FL execution time in AWS corresponds to 91.45% of the total *Multi-FedLS* execution time when there is no revocation. In CloudLab it is only 31.67%, which can be explained by the *bare-metal* approach, which increases the VM preparation time (2:34 in AWS versus 39:43 in CloudLab). The same behavior is found when compared to GCP. There is a huge difference in the preparation time (13:35 in GCP versus 39:43 in CloudLab). In GCP, the FL execution time corresponds to 53.30% of the *Multi-FedLS* execution time. Besides, one disadvantage of CloudLab is that once the instance is terminated, the data modified in it gets lost, and we need to download the results before terminating the VMs, which adds more than 20 minutes in the *Multi-FedLS* execution. In AWS and GCP, we stored the produced data in an extra volume that was not deleted with the VM and we download them separately. Table 7.13 present the summary of these times, with the optimal execution in each environment.

Table 7.13: Execution time of *Multi-FedLS* with on-demand VMs in AWS, GCP and CloudLab with different GPUs.

Execution	VMs prep.	FL exec. time	Download results	Termin. process	Multi-FedLS total time
AWS (T4)	0:02:34	1:31:18	-	0:05:59	1:39:50
GCP (V100)	0:13:35	0:22:00	-	0:05:42	0:41:17
CloudLab (P100)	0:39:43	0:29:28	0:22:10	0:01:42	1:33:03

7.5 Fault Tolerance experiments

The next experiment analyses the impact of the Fault Tolerance module on the FL execution time and the total framework time using only the TIL use-case application since it has the model with the highest number of weights and the most costly checkpoint (504 MB). We increase the number of rounds to reflect on the execution time and make the use of the Fault Tolerance and Dynamic Scheduler modules more reasonable. We have two different types of checkpoints: one on the server every X rounds and the client checkpoint every round. We test them separately. Figure 7.2 shows the average of the *Multi-FedLS* and the FL execution time of three executions varying the X within 10, 20, 30, and 40 rounds.

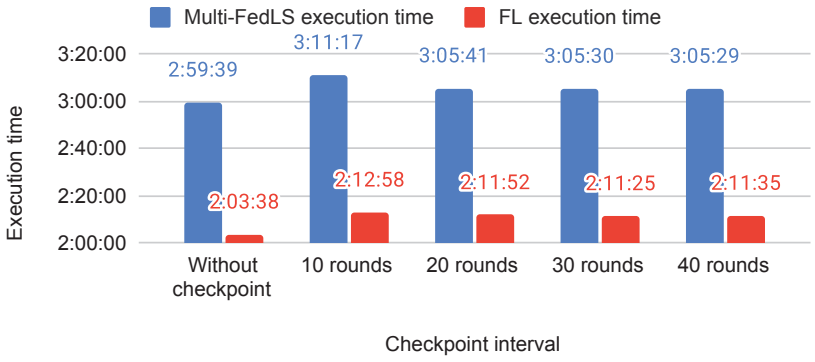


Figure 7.2: Server checkpoint overhead.

We can observe that the frequency of checkpointing impacts similarly both execution times. The overhead compared to the FL execution without checkpoint varies from 6.29% (30 rounds) to 7.55% (10 rounds). This overhead is mostly influenced by the saving time to the local disk, as the checkpoints transmission to another location overlap the server's waiting for clients' messages.

Clients do not send their checkpoints to an external location, so we compute the clients' checkpoint overhead saving the weights in the disk after every evaluation step. The average execution time from three runs was 3:03:44 to the *Multi-FedLS* time and 2:06:20 when putting only the FL execution time, which corresponds to a 2.17% overhead compared to no checkpoint execution.

7.6 Dynamic Scheduler experiments

We execute this test considering all three applications to observe the behaviour of the whole Multi-FedLS. We first show the results for the real-world application and then the results for the benchmark applications.

7.6.1 TIL application

To validate the Dynamic Scheduler module, we simulated the VM revocation using a Poisson distribution [2] with a revocation rate $\lambda = 1/k_r$, where k_r is the average time between failures in seconds. We observed patterns in the revocation frequency in instances with GPU in AWS [18] which gave us two different values to our experimental k_r, 2 hours and 4 hours. Thus, there are two revocation rates: (i) $k_r = 7200$ and $\lambda = 1/7200$, and (ii) $k_r = 14400$ and $\lambda = 1/14400$.

The revocation of a client and the server impact differently in the whole FL execution. Thus, we created three simulation scenarios to explore this difference. The first scenario (Server and clients on Spot VMs) uses Spot (preemptible) VMs for all tasks. The second scenario (Server on an On-demand VM and clients on Spot VMs) uses an On-demand VM to the server, which increases the costs compared to the first scenario, but increases the reliability of the server task. The last scenario (Server on a Spot VM and clients on on-demand VMs) is the opposite, we have clients using On-demand VMs and only the server in a Spot VM.

Before presenting the execution times and costs, we compared the average final metrics of the executions without revocation (Section 7.5) and executions with revocation, from this section. The biggest difference was the precision metricm that yielded a 2.22% lower average when there were revocations compared to when there were not revocations.

Table 7.14 shows the results for three executions. It presents the simulation scenario (Scenario), the termination rate in question, the average number of revocations (Avg # revoc.), the average execution time (Avg exec. time), and the average total costs (Avg total costs). As a comparison, if all tasks execute on On-demand VMs without checkpoints, the execution time is 2:59:39 and the total costs are $50.51.

Our dynamic scheduler always chooses the same instances when there was a revocation. Clients start on a VM vm_{326} and restart on a VM vm_{338}. The server starts on a VM vm_{321} and restarts in a VM vm_{412}. There was no more than one VM revocation per

Table 7.14: Failure simulation using TIL application changing to another VM in CloudLab

Scenario	Termination rate (k_r)	Avg # revoc.	Avg exec. time	Avg total costs ($)
Server and clients on on-demand VMs	-	-	2:59:39	$50.51
Server and clients on spot VMs	7200	3.67	10:01:46	81.12
	14400	0.00	3:04:37	15.64
Server on an on-demand VM and clients on spot VMs	7200	1.00	6:31:44	55.60
	14400	0.00	3:05:39	19.27
Server on a spot VM and clients on on-demand VMs	7200	0.00	3:00:44	48.11
	14400	0.00	3:04:16	49.13

task in each execution. In the first scenario, where all tasks execute in Spot VMs, two of the three executions had four revocations (3 clients and the server) and the last one had three revocations (2 clients and the server), having an average revocation number of 3.67. In the second scenario, where the server executes in an On-demand VM and the clients are in Spot VMs, one of the executions had two revocations, another one revocation and the last one did not have any revocation, resulting in one average revocation.

We can observe that Spot VMs are not an advantage in CloudLab, as no two VMs execute the clients at similar times. The VM vm_{338} has a slowdown of 0.57 while the VM vm_{326} has a slowdown of 0.04 (Table 7.5). Moreover, the time to prepare the VMs also impacts the execution time, and, eventually, the total costs with Spot VMs surpass the costs when using only On-demand VMs.

Real commercial cloud providers usually have VMs with similar or even equivalent configurations in different cloud regions. For example, AWS provides *g4dn.2xlarge* instances (that have Turing GPUs) in 23 of its regions[2] and GCP provides Turing GPUs in seven of its regions[3]. These equivalent machines tend to have similar execution times (with a slight difference), as observed in the slowdowns in Tables 7.3 and 7.4. On CloudLab, each instance type has completely different hardware configurations, but it is allowed to use the same instance type immediately after a revocation. Now, our dynamic scheduler does not remove the VM that was revoked from the available instance types (first line of Algorithm 3) and it allows to choose the same instance type in every revocation.

Table 7.15 shows the results from three executions restarting in the same instance type, where columns have the same meaning as Table 7.14.

[2]https://aws.amazon.com/ec2/pricing/on-demand/, accessed in June 2023
[3]https://cloud.google.com/compute/docs/gpus/gpu-regions-zones, acessed May 2023

Table 7.15: Failure simulation using TIL application changing to the same VM in CloudLab

Scenario	Termination rate (k_r)	Avg # revoc.	Avg exec. time	Avg total costs ($)
Server and clients on on-demand VMs	-	-	2:59:39	$50.51
Server and clients on spot VMs	7200	1.33	4:14:16	22.55
	14400	0.00	3:04:35	5.64
Server on an on-demand VM and clients on spot VMs	7200	0.33	3:14:38	20.16
	14400	0.00	3:01:49	18.99
Server on a spot VM and clients on on-demand VMs	7200	0.33	5:33:27	95.73
	14400	0.00	3:12:57	49.28

In the first scenario with the termination rate of 1/(2 hours) (1/7200), one execution had two clients' revocations, another just one client revocation and the third one just the server revocation, totaling an average of 1.33 revocations. In the second (resp., third) scenario with the same termination rate, just one execution revoked one client (resp., the server once), having 0.33 as the average revocation number.

From these results, we can observe that the revocation of a client impacts less the execution time than the server's revocation (3:14:38 vs. 5:33:27). Moreover, the execution of clients in On-demand VMs increases drastically the costs, *e.g.*, $49.28 when executing the server on a Spot VM and the clients on On-demand ones vs. $18.99 the opposite (clients on Spot VMs and server on an On-demand one), which makes the clients' execution in On-demand VMs not worthy. Regarding the server execution in an On-demand VM vs. all tasks in Spot VMs, we can see that when there is no revocation, the costs increase by 21.30% ($18.99 vs. $15.64), while when there are revocations, the costs decrease by 10.58% ($20.16 vs. $22.55).

7.6.2 Benchmarks

For the Shakespeare application, we executed 20 rounds with 20 epochs per round, and the FEMNIST application executed 100 rounds with 100 epochs per round, following the LEAF configuration [21].

When executing both benchmarks, we observed that their execution times were smaller than the TIL application. Because of that, there was no revocation with the first termination rate. Thus, we created a third revocation rate, k_r being 1 hour, *i.e.*, (iii) $k_r = 3600$ and $\lambda = 1/3600$.

Their execution only in On-demand VMs yields the following results. The Shakespeare application executes for almost 1 hour and 54 minutes (1:53:54) having a total cost of $53.31, while the FEMNIST application executes for 1 hour and 56 minutes (1:56:37) with a total cost of $35.68. The Shakespeare application costs more because it has 8 clients and the FEMNIST one has only 5 clients, despite using the same VMs for server and clients.

Table 7.16 presents the results of executing the Shakespeare application in CloudLab with the three different scenarios (all tasks in Spot VMs; server on On-demand VM and clients on Spot VMs; and server on Spot VM and clients on On-demand VMs) with the two termination rates: 1/(1 hour) and 1/(2 hours). Table 7.17 shows the same results for the FEMNIST application. Note that both tables present the average values for 3 executions.

Table 7.16: Failure simulation executing Shakespeare application and changing to the same VM in CloudLab

Scenario	Termination rate (k_r)	Avg # revoc.	Avg exec. time	Avg total costs ($)
Server and clients on on-demand VMs	-	-	1:53:54	$53.31
Server and clients on spot VMs	3600	1.33	2:17:12	20.02
	7200	0.00	1:58:31	17.03
Server on an on-demand VM and clients on spot VMs	3600	2.67	2:32:12	23.46
	7200	0.00	1:57:56	17.27
Server on a spot VM and clients on on-demand VMs	3600	0.33	2:00:47	57.06
	7200	0.00	1:54:06	53.29

Table 7.17: Failure simulation using FEMNIST application and changing to the same VM in CloudLab

Scenario	Termination rate (k_r)	Avg # revoc.	Avg exec. time	Avg total costs ($)
Server and clients on on-demand VMs	-	-	1:56:37	$35.68
Server and clients on spot VMs	3600	2.00	2:34:33	14.63
	7200	0.00	1:52:21	10.21
Server on an on-demand VM and clients on spot VMs	3600	1.67	2:38:05	16.10
	7200	0.00	1:56:02	11.35
Server on a spot VM and clients on on-demand VMs	3600	0.67	2:14:55	42.15
	7200	0.00	1:51:07	33.10

The three executions of the first scenario with the first termination rate in the Shake-

speare application revoked only clients, with 2 of them revoking only one and the last one revoking 2 clients, resulting in an average of 1.33 revocations. In FEMNIST, two executions revoked a client and the server (2 tasks in total each) and the last one revoked 2 clients, resulting in the average of 2 revocations.

Similarly to the TIL application, we can observe that the use of spot instances with these benchmarks is an advantage, especially to the clients tasks. The use of a Spot VM to the server and On-demand VMs to the client yields a small cost reduction in the Shakespeare application compared to executing everything on On-demand VMs as this application has 8 clients ($53.29 vs. $53.31). Moreover, when there is at least one revocation in the server, the costs surpass using only On-demand VMs ($57.06). In the FEMNIST application, the reduction is bigger when there is no revocation ($33.10 vs. $35.68), but it still surpasses the purely On-demand VMs when there is any revocation ($42.15). Regarding the server execution in an On-demand VM vs. all tasks in Spot VMs, we can see that there were only increases in the costs on both Shakespeare and FEMNIST applications, with or without revocations. In the Shakespeare application, the increase is only by 1.39% when there is no revocation ($17.27 vs. $17.03) and by 17.16% when there are revocations ($23.46 vs. $20.02). In the FEMNIST application, both increases are similar, 11.12% when there is no revocation ($11.15 vs. $10.21) and 10.10% when there are revocations ($16.10 and $ 14.63).

7.7 Proof of concept

In this final experiment, we demonstrate a proof of concept that our entire framework executes in a real multi-cloud scenario. Due to a project termination, we needed to change our AWS account and thus, our regional vCPU and GPU limits decreased. Due to these restricting GPU and vCPU limitations, we execute our real-world application with only 2 clients, one client storing its datasets in AWS and the other one in GCP, for 30 communication rounds. Moreover, we only have 3 regions from Table 7.1 to execute in total: region *us-east-1* (N. Virginia) in AWS and regions *us-central1* (Iowa) and *us-west1* (Oregon) in GCP and some instances in GCP could not be instantiated. Thus, this test has the following instances available to be chosen: vm_{111}, vm_{112}, vm_{113}, vm_{211}, vm_{213}, vm_{214}, vm_{222}, and vm_{223} (Table 7.1).

Our Initial Mapping module computes the optimal setup as all tasks executing in AWS, with the server in the VM vm_{113} and the clients in vm_{111} VMs. The average values

of three executions in only On-demand VMs yield a runtime of 1:57:56 and a cost of $3.21. Table 7.18 shows the execution with the termination rate of 2 hours ($k_r = 7200$ and $\lambda = 1/7200$) storing the checkpoints in N. Virginia region of Amazon S3 (AWS) or in Iowa region of Cloud Storage (GCP). We present the average number of revocations, execution time and total costs of three executions. We also present the percentage compared to the on-demand execution.

Table 7.18: Proof of concept executing our real-world application with 2 clients in a multi-cloud environment

Ckpt storage	Avg # revocations	Avg exec. time	Avg total cost	Difference to on-demand execution (%)	
				Execution time	Total cost
AWS	1.33	2:06:51	$1.41	7.55	-55.96
GCP	2.00	2:19:03	$1.49	17.89	-53.74

When we stored the server checkpoints in AWS, our simulator terminated the server in all three executions and only 1 client in the first execution, which led to the 1.33 average revocation number. When the checkpoints were sent to GCP, there were 2 revocations in all executions, being one from the server and the other from the client. After a client revocation, our framework always chose the VM vm_{211} to restart the affected client and after a server revocation, the framework always chose to restart the server in a vm_{214} VM.

We can observe from Table 7.18 that our framework is robust as it reduces the costs by 54.85% on average while increasing the execution time by only 12.72% on average. Moreover, using Amazon S3 instead of Cloud Storage seems to be a better choice, but more tests are necessary.

Chapter 8

Conclusion and Future Work

This chapter describes the main results and contributions of this thesis. Section 8.1 highlights the main contributions, while Section 8.3 gives some directions for future research.

8.1 Contributions

Federated Learning is a research field of distributed Machine Learning that tackles the prohibition to share data among different institutions due to data privacy. When the data produced by each institution increases rapidly, they can rely on cloud storage services to reduce their upcoming cost, as there is no building and maintenance costs when using these services compared to on-site storage clusters. As most FL applications use DNNs with lots of matrix multiplications, the use of GPUs to train each local model is necessary and cloud providers offers different GPUs architectures to execute in an on-demand request.

Multi-FedLS is a robust, adaptive, and flexible framework that focuses on executing FL applications in a multi-cloud environment with each client dataset stored in different cloud providers, reducing execution time and costs. It also takes advantage of preemptible VMs, which have a significant discount compared to on-demand ones, but can be revoked at any time. Thus, this framework is robust as it guarantees the correct and whole execution of the FL application through its Initial Mapping, Fault Tolerance, and Dynamic Scheduler modules. It is adaptive as the Pre-Scheduling module collects data from new parameters in the environment without repeating the metrics for the old parameters already collected. Based on our experiments, it is possible to use the Pre-Scheduling or the Initial Mapping modules without triggering the Fault Tolerance or the Dynamic Scheduler ones, which makes *Multi-FedLS* flexible. Moreover, we implement our framework in a modular way, so each module connects easily to the whole framework execution.

We evaluated our modules using both a real multi-cloud environment and a simulated environment due to scalability issues. Our Initial Mapping module presented a reduction of 40.38% in the execution time with a monetary cost increase of 24.61% when compared to user random VM selections. The Fault Tolerance module yields a small overhead of only 7.55% due to the asynchronous sending of the checkpoints to another location. Concerning the Dynamic Scheduler, we could observe that the impact of a server revocation is bigger than a client one, up to more than 2 hours of difference in the execution time. When executing the whole framework in a real multi-cloud environment, we obtained a 54.85% reduction in costs while increasing the execution time by only 12.72% compared to the execution with On-demand VMs only.

8.2 Limits of Multi-FedLS

As Flower needs to execute through a known number of rounds, we known before the execution the total expected execution time and we could calculate the remaining rounds easily returning from a revocation. However, usually FL practitioners do not know the best number of rounds, they need to execute several different configurations to compare the final ML metrics. Thus, in this scenario, the FL practitioners would execute *Multi-FedLS* several times, each time with a different configuration and they need to take into consideration the total budget and total time to execute these configurations to separate them among the several executions.

8.3 Future Work

In this section, we propose some promising future directions derived from the contributions of this thesis.

- **Evaluation of different ML models in our framework:** We presented our tests with three different applications. However, all of them were DNNs with several matrix multiplications needing GPUs to execute in a reasonable amount of time. There are other ML models that can be trained in an FL approach, for example, Linear Regression [102]. Thus, conducting experiments with other ML models is important to confirm the robustness of our framework.
- **Execution of multiple FL applications at once:** As shown throughout this thesis, *Multi-FedLS* supports the execution of a single FL application per execution. How-

ever, when a new FL approach is designed for an application, there is no formula to compute the best values for communication rounds and epochs per round. With the current version of our framework, researchers need to execute it multiple times to change the concerned application. One future direction could be the addition of one or more modules to execute multiple FL applications concurrently. They can start all together in the beginning or transform our offline scheduling problem into an online problem allowing the insertion of new FL applications during the framework execution. Thus, our framework needs to divide the current limits from all providers to all executing applications and guarantee their complete execution.

- **Active transfer to another VM:** Currently, our framework only changes the VM assigned to a task after an error or a revocation in a reactive way. One improvement would be to actively change the VM to any task due to performance degradation. For example, our framework could monitor message exchanges in each FL round and change a VM when a recurrent network delay is observed.

Bibliography

[1] Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G. S.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Goodfellow, I.; Harp, A.; Irving, G.; Isard, M.; Jia, Y.; Jozefowicz, R.; Kaiser, L.; Kudlur, M.; Levenberg, J.; Mané, D.; Monga, R.; Moore, S.; Murray, D.; Olah, C.; Schuster, M.; Shlens, J.; Steiner, B.; Sutskever, I.; Talwar, K.; Tucker, P.; Vanhoucke, V.; Vasudevan, V.; Viégas, F.; Vinyals, O.; Warden, P.; Wattenberg, M.; Wicke, M.; Yu, Y.; Zheng, X. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Ahrens, J. H.; Dieter, U. Computer methods for sampling from gamma, beta, poisson and bionomial distributions. *Computing 12*, 3 (1974), 223–246.

[3] Angell, H.; Galon, J. From the immune contexture to the Immunoscore: the role of prognostic and predictive immune markers in cancer. *Current Opinion in Immunology 25*, 2 (2013), 261 – 267. Lymphocyte development / Tumour immunology / Cancer immunology: Clinical translation.

[4] Armbrust, M.; Fox, A.; Griffith, R.; Joseph, A. D.; Katz, R.; Konwinski, A.; Lee, G.; Patterson, D.; Rabkin, A.; Stoica, I., et al. A view of cloud computing. *Communications of the ACM 53*, 4 (2010), 50–58.

[5] Azure, M. Cloud Computing Services. `https://azure.microsoft.com/en-us/`, 2021. Accessed 19 December 2021.

[6] Bao, Y.; Peng, Y.; Wu, C.; Li, Z. Online job scheduling in distributed machine learning clusters. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications* (2018), pp. 495–503.

[7] Ben-Nun, T.; Hoefler, T. Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis. *ACM Comput. Surv. 52*, 4 (8 2019).

[8] Beutel, D. J.; Topal, T.; Mathur, A.; Qiu, X.; Parcollet, T.; Lane, N. Flower: A friendly federated learning research framework. *ArXiv abs/2007.14390* (2020).

[9] Bohn, R. B.; Messina, J.; Liu, F.; Tong, J.; Mao, J. Nist cloud computing reference architecture. In *2011 IEEE World Congress on Services* (2011), pp. 594–596.

[10] Breiman, L. Random forests. *Machine Learning 45*, 1 (Oct 2001), 5–32.

[11] Brotcorne, L.; Ezpeleta, J.; Galé, C. A biobjective model for resource provisioning in multi-cloud environments with capacity constraints. *Operational Research 23*, 2 (May 2023), 31.

[12] Brum, R.; Drummond, L.; Castro, M. C.; Teodoro, G. 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.

[13] Brum, R.; Teodoro, G.; Drummond, L.; Arantes, L.; Castro, M.; Sens, P. 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, 2021), SBC, pp. 6–10.

[14] Brum, R. C.; Arantes, L.; Castro, M. C.; Sens, P.; Drummond, L. M. A. Evaluating Execution Times and Costs of a Federated Learning Application on different Cloud Providers. In *COMPAS 2022 - Conférence francophone d'informatique en Parallélisme, Architecture et Système* (Amiens, France, July 2022).

[15] Brum, R. C.; de Castro, M. C. S.; Arantes, L.; de A. Drummond, L. M.; Sens, P. Multi-fedls: a framework for cross-silo federated learning applications on multi-cloud environments, 2023.

[16] Brum, R. C.; Sens, P.; Arantes, L.; Castro, M. C.; de A. Drummond, L. M. Optimizing execution time and costs of cross-silo federated learning applications with datasets on different cloud providers. In *2022 IEEE 34th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)* (2022), pp. 253–262.

[17] Brum, R. C.; Sens, P.; Arantes, L.; Castro, M. C.; Drummond, L. M. d. A. Towards a federated learning framework on a multi-cloud environment. In

2022 International Symposium on Computer Architecture and High Performance Computing Workshops (SBAC-PADW) (2022), pp. 39–44.

[18] Brum, R. C.; Sousa, W. P.; Melo, A. C. M. A.; Bentes, C.; de Castro, M. C. S.; Drummond, L. M. d. A. A fault tolerant and deadline constrained sequence alignment application on cloud-based spot gpu instances. In *Euro-Par 2021: Parallel Processing* (Cham, 2021), L. Sousa, N. Roma, and P. Tomás, Eds., Springer International Publishing, pp. 317–333.

[19] Burlachenko, K.; Horváth, S.; Richtárik, P. FL_PyTorch: Optimization Research Simulator for Federated Learning. In *Proceedings of the 2nd ACM International Workshop on Distributed Machine Learning* (New York, NY, USA, 2021), DistributedML '21, Association for Computing Machinery, p. 1–7.

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

[21] Caldas, S.; Duddu, S. M. K.; Wu, P.; Li, T.; Konečný, J.; McMahan, H. B.; Smith, V.; Talwalkar, A. Leaf: A benchmark for federated settings, 2019.

[22] Chen, Y.; Sun, X.; Jin, Y. Communication-efficient federated deep learning with layerwise asynchronous model update and temporally weighted aggregation. *IEEE Transactions on Neural Networks and Learning Systems 31*, 10 (2020), 4229–4238.

[23] Chhabra, A.; Huang, K.-C.; Bacanin, N.; Rashid, T. A. Optimizing bag-of-tasks scheduling on cloud data centers using hybrid swarm-intelligence meta-heuristic. *The Journal of Supercomputing 78*, 7 (May 2022), 9121–9183.

[24] Cloud Foundry, I. Cloud Foundry - Open Source Cloud Native Application Delivery. `https://www.cloudfoundry.org/`, 2022. Accessed 10 January 2022.

[25] Deldari, A.; Salehan, A. A survey on preemptible iaas cloud instances: challenges, issues, opportunities, and advantages. *Iran Journal of Computer Science 4*, 3 (Sep 2021), 1–24.

[26] Deng, L. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine 29*, 6 (2012), 141–142.

[27] Developers, G. Cloud Storage for Firebase. https://firebase.google.com/products/storage/, 2022. Accessed 11 January 2022.

[28] Dikaiakos, M. D.; Katsaros, D.; Mehra, P.; Pallis, G.; Vakali, A. Cloud computing: Distributed internet computing for it and scientific research. *IEEE Internet Computing 13*, 5 (2009), 10–13.

[29] Docs, M. What is Azure Databricks? https://docs.microsoft.com/en-us/azure/databricks/scenarios/what-is-azure-databricks, 2022. Accessed 16 January 2022.

[30] Duong, T. N. B. *FC*^{2}: cloud-based cluster provisioning for distributed machine learning. *Cluster Computing 22*, 4 (Dec 2019), 1299–1315.

[31] Duong, T. N. B.; Sang, N. Q. Distributed Machine Learning on IAAS Clouds. In *2018 5th IEEE International Conference on Cloud Computing and Intelligence Systems (CCIS)* (2018), pp. 58–62.

[32] Duplyakin, D., et al. The design and operation of CloudLab. In *Proceedings of the USENIX Annual Technical Conference (ATC)* (jul 2019), pp. 1–14.

[33] ElGamal, T. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory 31*, 4 (1985), 469–472.

[34] Elkhatib, Y. Mapping Cross-Cloud Systems: Challenges and Opportunities. In *8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 16)* (Denver, CO, June 2016), USENIX Association.

[35] Fabra, J.; Ezpeleta, J.; Álvarez, P. Reducing the price of resource provisioning using ec2 spot instances with prediction models. *Future Generation Computer Systems 96* (2019), 348–367.

[36] Fang, C.; Guo, Y.; Wang, N.; Ju, A. Highly efficient federated learning with strong privacy preservation in cloud computing. *Computers & Security 96* (2020), 101889.

[37] FedAI. FATE. https://fate.readthedocs.io/en/latest/, 2019. Accessed 27 December 2021.

[38] FedAI. FATE Serving - FATE. https://fate.fedai.org/fate-serving/, 2019. Accessed 27 December 2021.

[39] FedAI. FATEBoard - FATE. https://fate.fedai.org/fateboard/, 2019. Accessed 28 December 2021.

[40] FedAI. FATEFlow - FATE. https://fate.fedai.org/fateflow/, 2019. Accessed 27 December 2021.

[41] FedAI. Federated Network - FATE. https://fate.fedai.org/federated-network/, 2019. Accessed 27 December 2021.

[42] FedAI. FederatedML - FATE. https://fate.fedai.org/federatedml/, 2019. Accessed 27 December 2021.

[43] FedAI. KubeFATE - FATE. https://fate.fedai.org/kubefate/, 2019. Accessed 28 December 2021.

[44] Federated, T. Federated Core. https://www.tensorflow.org/federated/federated_core, 2021. Accessed 22 December 2021.

[45] Federated, T. Federated Learning. https://www.tensorflow.org/federated/federated_learning, 2021. Accessed 22 December 2021.

[46] Federated, T. Federated Learning for Text Generation. https://www.tensorflow.org/federated/tutorials/federated_learning_for_text_generation, 2021. Accessed 22 December 2021.

[47] Foster, I.; Zhao, Y.; Raicu, I.; Lu, S. Cloud computing and grid computing 360-degree compared. In *2008 Grid Computing Environments Workshop* (2008), pp. 1–10.

[48] Furht, B., Ed. *SIMD (Single Instruction Multiple Data Processing)*. Springer US, Boston, MA, 2008, pp. 817–819.

[49] Goodfellow, I.; Bengio, Y.; Courville, A. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[50] Google. Google Workspace Essentials. https://workspace.google.com/essentials/, 2022. Accessed 11 January 2022.

[51] Google Cloud, P. Cloud Computing Services. https://cloud.google.com/, 2021. Accessed 19 December 2021.

[52] Google Cloud, P. Cloud Storage. https://cloud.google.com/storage, 2021. Accessed 19 December 2021.

[53] Google Cloud, P. Quotas & limits - Cloud Storage. https://cloud.google.com/storage/quotas, 2021. Accessed 19 December 2021.

[54] Google Cloud, P. App Engine Application Platform. https://cloud.google.com/appengine/, 2022. Accessed 10 January 2022.

[55] Google Cloud, P. Artifact Registry. https://cloud.google.com/artifact-registry, 2022. Accessed 11 January 2022.

[56] Google Cloud, P. Cloud Computing Services. https://cloud.google.com/products/storage, 2022. Accessed 11 January 2022.

[57] Google Cloud, P. Cloud Tensor Processing Units (TPUs). https://cloud.google.com/tpu/docs/tpus, 2022. Accessed 19 January 2022.

[58] Google Cloud, P. Compute Engine. https://cloud.google.com/compute, 2022. Accessed 10 January 2022.

[59] Google Cloud, P. Felistore. https://cloud.google.com/filestore, 2022. Accessed 11 January 2022.

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

[61] Google Cloud, P. Local SSD. https://cloud.google.com/local-ssd, 2022. Accessed 11 January 2022.

[62] Google Cloud, P. Persistent Disk. https://cloud.google.com/persistent-disk, 2022. Accessed 11 January 2022.

[63] Google Cloud, P. Storage Transfer Service. https://cloud.google.com/storage-transfer-service, 2022. Accessed 11 January 2022.

[64] Google Cloud, P. Transfer Appliance. https://cloud.google.com/transfer-appliance/docs/4.0, 2022. Accessed 11 January 2022.

[65] Google Cloud, P. Backup and DR Service. https://cloud.google.com/backup-disaster-recovery, 2023. Accessed 10 June 2023.

[66] Google Cloud, P. Geography and regions - Documentation. https://cloud.google.com/about/locations, 2023. Accessed 10 June 2023.

[67] gRPC Authors. gRPC. https://grpc.io/, 2022. Accessed 04 January 2022.

[68] Gurobi. Gurobi optimizer, 2022.

[69] Hall, A. J.; Jay, M.; Cebere, T.; Cebere, B.; van der Veen, K. L.; Muraru, G.; Xu, T.; Cason, P.; Abramson, W.; Benaissa, A.; Shah, C.; Aboudib, A.; Ryffel, T.; Prakash, K.; Titcombe, T.; Khare, V. K.; Shang, M.; Junior, I.; Gupta, A.; Paumier, J.; Kang, N.; Manannikov, V.; Trask, A. Syft 0.5: A platform for universally deployable structured transparency. *arXiv preprint arXiv:2104.12385* (2021).

[70] Hashem, I. A. T.; Yaqoob, I.; Anuar, N. B.; Mokhtar, S.; Gani, A.; Ullah Khan, S. The rise of "big data" on cloud computing: Review and open research issues. *Information Systems 47* (2015), 98–115.

[71] Heroku. Cloud Application Platform. `https://www.heroku.com/`, 2022. Accessed 10 January 2022.

[72] Hong, J.; Dreibholz, T.; Schenkel, J. A.; Hu, J. A. An overview of multi-cloud computing. In *Web, Artificial Intelligence and Network Applications* (Cham, 2019), L. Barolli, M. Takizawa, F. Xhafa, and T. Enokido, Eds., Springer International Publishing, pp. 1055–1068.

[73] Huang, Y.; Chu, L.; Zhou, Z.; Wang, L.; Liu, J.; Pei, J.; Zhang, Y. Personalized Cross-Silo Federated Learning on Non-IID Data. *Proceedings of the AAAI Conference on Artificial Intelligence 35*, 9 (May 2021), 7865–7873.

[74] Ingerman, A.; Ostrowski, K. TensorFlow Blog: Introducing TensorFlow Federated. `https://blog.tensorflow.org/2019/03/introducing-tensorflow-federated.html`, 2019. Accessed 16 August 2021.

[75] Ji, S.; Pan, S.; Long, G.; Li, X.; Jiang, J.; Huang, Z. Learning private neural language modeling with attentive aggregation. In *2019 International Joint Conference on Neural Networks (IJCNN)* (2019), pp. 1–8.

[76] Juve, G.; Deelman, E.; Vahi, K.; Mehta, G.; Berriman, B.; Berman, B. P.; Maechling, P. Scientific workflow applications on Amazon EC2. In *2009 5th IEEE International Conference on E-Science Workshops* (2009), pp. 59–66.

[77] Karaja, M.; Chaabani, A.; Azzouz, A.; Ben Said, L. Efficient bi-level multi objective approach for budget-constrained dynamic bag-of-tasks scheduling problem in heterogeneous multi-cloud environment. *Applied Intelligence 53*, 8 (Apr 2023), 9009–9037.

[78] Ke, G.; Meng, Q.; Finley, T.; Wang, T.; Chen, W.; Ma, W.; Ye, Q.; Liu, T.-Y. Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems* (2017), I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30, Curran Associates, Inc.

[79] Kotas, C.; Naughton, T.; Imam, N. 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.

[80] Le, H.; Gupta, R.; Hou, L.; Abousamra, S.; Fassler, D.; Torre-Healy, L.; Moffitt, R. A.; Kurc, T.; Samaras, D.; Batiste, R.; Zhao, T.; Rao, A.; Van Dyke, A. L.; Sharma, A.; Bremer, E.; Almeida, J. S.; Saltz, J. Utilizing automated breast cancer detection to identify spatial distributions of tumor-infiltrating lymphocytes in invasive breast cancer. *The American Journal of Pathology 190*, 7 (Jul 2020), 1491–1504.

[81] Leavitt, N. Storage challenge: Where will all that big data go? *Computer 46*, 09 (2013), 22–25.

[82] Leitner, P.; Cito, J. Patterns in the chaos—a study of performance variation and predictability in public iaas clouds. *ACM Trans. Internet Technol. 16*, 3 (apr 2016).

[83] Li, A.; Yang, X.; Kandula, S.; Zhang, M. CloudCmp: Comparing Public Cloud Providers. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement* (New York, NY, USA, 2010), IMC '10, Association for Computing Machinery, p. 1–14.

[84] Li, Q.; Diao, Y.; Chen, Q.; He, B. Federated learning on non-iid data silos: An experimental study. *ArXiv abs/2102.02079* (2021).

[85] Li, T.; Sahu, A. K.; Zaheer, M.; Sanjabi, M.; Talwalkar, A.; Smith, V. Federated optimization in heterogeneous networks. *arXiv preprint arXiv:1812.06127* (2018).

[86] Li, T.; Sanjabi, M.; Beirami, A.; Smith, V. Fair resource allocation in federated learning, 2020.

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

[88] Liu, H. Big data drives cloud adoption in enterprise. *IEEE Internet Computing 17*, 4 (2013), 68–71.

[89] Liu, L.; Yu, H.; Sun, G.; Zhou, H.; Li, Z.; Luo, S. Online job scheduling for distributed machine learning in optical circuit switch networks. *Knowledge-Based Systems 201-202* (2020), 106002.

[90] Liu, L.; Zhang, J.; Song, S.; Letaief, K. B. Client-edge-cloud hierarchical federated learning. In *2020-2020 IEEE International Conference on Communications* (2020).

[91] Malta, E. M.; Avila, S.; Borin, E. 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* (New York, NY, USA, 2019), UCC'19, Association for Computing Machinery, p. 21–29.

[92] McMahan, B.; Moore, E.; Ramage, D.; Hampson, S.; y Arcas, B. A. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics* (Fort Lauderdale, FL, USA, 4 2017), A. Singh and J. Zhu, Eds., vol. 54 of *Proceedings of Machine Learning Research*, PMLR, pp. 1273–1282.

[93] Mell, P.; Grance, T., et al. The nist definition of cloud computing.

[94] Mitchell, T. M., et al. *Machine learning*. McGraw-Hill, Inc., New York, NY, USA, 1997.

[95] Mohammadi Amiri, M.; Gündüz, D. Computation scheduling for distributed machine learning with straggling workers. *IEEE Transactions on Signal Processing 67*, 24 (2019), 6270–6284.

[96] National Human Genome Research Institute. The Cancer Genome Atlas. https://cancergenome.nih.gov/, June 2017.

[97] Nguyen, V.-D.; Sharma, S. K.; Vu, T. X.; Chatzinotas, S.; Ottersten, B. Efficient federated learning algorithm for resource allocation in wireless iot networks. *IEEE Internet of Things Journal 8*, 5 (2021), 3394–3409.

[98] NVIDIA; Vingelmann, P.; Fitzek, F. H. Cuda, release: 10.2.89, 2020.

[99] Office, M. Office 375. https://www.office.com/, 2022. Accessed 10 January 2022.

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

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

[102] Rajendran, S.; Obeid, J. S.; Binol, H.; D'Agostino, R.; Foley, K.; Zhang, W.; Austin, P.; Brakefield, J.; Gurcan, M. N.; Topaloglu, U. Cloud-Based Federated Learning Implementation Across Medical Centers. *JCO Clinical Cancer Informatics*, 5 (2021), 1–11. PMID: 33411624.

[103] Ren, J.; Ni, W.; Nie, G.; Tian, H. Research on resource allocation for efficient federated learning, 2021.

[104] Ren, J.; Sun, J.; Tian, H.; Ni, W.; Nie, G.; Wang, Y. 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.

[105] Ryffel, T., et al. A generic framework for privacy preserving deep learning. *ArXiv abs/1811.04017* (2018).

[106] Saltz, J.; Gupta, R.; Hou, L.; Kurc, T.; Singh, P.; Nguyen, V.; Samaras, D.; Shroyer, K. R.; Zhao, T.; Batiste, R., et al. Spatial Organization And Molecular Correlation Of Tumor-Infiltrating Lymphocytes Using Deep Learning On Pathology Images. *Cell reports 23*, 1 (2018), 181.

[107] Services, A. W. Amazon S3. https://aws.amazon.com/s3/, 2021. Accessed 19 December 2021.

[108] Services, A. W. Cloud Services. https://aws.amazon.com/, 2021. Accessed 19 December 2021.

[109] Services, A. W. Amazon DataSync. https://aws.amazon.com/datasync/, 2022. Accessed 11 January 2022.

[110] Services, A. W. Amazon EBS. https://aws.amazon.com/ebs, 2022. Accessed 11 January 2022.

[111] Services, A. W. Amazon EC2. https://aws.amazon.com/ec2/, 2022. Accessed 10 January 2022.

[112] Services, A. W. Amazon EFS. https://aws.amazon.com/efs/, 2022. Accessed 11 January 2022.

[113] Services, A. W. Amazon FSx. https://aws.amazon.com/fsx/, 2022. Accessed 11 January 2022.

[114] Services, A. W. Amazon Snow Family. https://aws.amazon.com/snow/, 2022. Accessed 11 January 2022.

[115] Services, A. W. AWS Backup. https://aws.amazon.com/backup/, 2022. Accessed 11 January 2022.

[116] Services, A. W. AWS Elastic Beanstalk - Deploy Web Applications. https://aws.amazon.com/elasticbeanstalk/, 2022. Accessed 10 January 2022.

[117] Services, A. W. AWS Elastic Disaster Recovery. https://aws.amazon.com/disaster-recovery/, 2022. Accessed 11 January 2022.

[118] Services, A. W. AWS Storage Gateway. https://aws.amazon.com/storagegateway/, 2022. Accessed 11 January 2022.

[119] Services, A. W. AWS Transfer Family. https://aws.amazon.com/aws-transfer-family/, 2022. Accessed 11 January 2022.

[120] Services, A. W. Cloud Storage on AWS. https://aws.amazon.com/products/storage/, 2022. Accessed 11 January 2022.

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

[122] Services, A. W. Amazon File Cache. https://aws.amazon.com/filecache/, 2023. Accessed 10 June 2022.

[123] Services, A. W. Region and Zones - Amazon Elastic Compute Cloud. https://aws.amazon.com/about-aws/global-infrastructure/, 2023. Accessed 10 June 2023.

[124] Shastri, S.; Irwin, D. Cloud index tracking: Enabling predictable costs in cloud spot markets. In *Proceedings of the ACM Symposium on Cloud Computing* (New York, NY, USA, 2018), SoCC '18, Association for Computing Machinery, p. 451–463.

[125] Shen, S.; Zhu, T.; Wu, D.; Wang, W.; Zhou, W. From distributed machine learning to federated learning: In the view of data privacy and security. *Concurrency and Computation: Practice and Experience n/a*, n/a (2020).

[126] Simonyan, K.; Zisserman, A. Very deep convolutional networks for large-scale image recognition. In *3rd Int. Conf. on Learning Representations, ICLR 2015* (2015).

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

[128] Teylo, L.; Arantes, L.; Sens, P.; Drummond, L. M. d. A. Scheduling bag-of-tasks in clouds using spot and burstable virtual machines. *IEEE Transactions on Cloud Computing 11*, 1 (2023), 984–996.

[129] Teylo, L.; Brum, R.; Arantes, L.; Sens, P.; Drummond, L. Avaliação dos serviços de armazenamento da amazon web services para gravação e recuperação de checkpoints. In *Anais do XXI Workshop de Testes e Tolerância a Falhas* (Porto Alegre, RS, Brasil, 2020), SBC, pp. 29–40.

[130] Teylo, L.; Brum, R. C.; Arantes, L.; Sens, P.; Drummond, L. M. d. A. Developing Checkpointing and Recovery Procedures with the Storage Services of Amazon Web Services. In *49th International Conference on Parallel Processing - ICPP: Workshops* (New York, NY, USA, 2020), ICPP Workshops '20, Association for Computing Machinery.

[131] Toosi, A. N.; Calheiros, R. N.; Buyya, R. Interconnected cloud computing environments: Challenges, taxonomy, and survey. *ACM Comput. Surv. 47*, 1 (may 2014).

[132] Truong, N.; Sun, K.; Wang, S.; Guitton, F.; Guo, Y. Privacy preservation in federated learning: An insightful survey from the gdpr perspective. *Computers & Security 110* (2021), 102402.

[133] Ullman, J. D. Np-complete scheduling problems. *Journal of Computer and System sciences 10*, 3 (1975), 384–393.

[134] Vaquero, L. M.; Rodero-Merino, L.; Caceres, J.; Lindner, M. A Break in the Clouds: Towards a Cloud Definition. *SIGCOMM Comput. Commun. Rev. 39*, 1 (12 2009), 50–55.

[135] Varshney, P.; Simmhan, Y. Autobot: Resilient and cost-effective scheduling of a bag of tasks on spot vms. *IEEE Transactions on Parallel and Distributed Systems 30*, 7 (2019), 1512–1527.

[136] Villars, R. L.; Olofson, C. W.; Eastwood, M. Big data: What it is and why you should care. *White paper, IDC 14* (2011), 1–14.

[137] Wagenländer, M.; Mai, L.; Li, G.; Pietzuch, P. Spotnik: Designing Distributed Machine Learning for Transient Cloud Resources. In *12th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 20)* (July 2020), USENIX Association.

[138] Ward, J. S.; Barker, A. Observing the clouds: a survey and taxonomy of cloud monitoring. *Journal of Cloud Computing 3*, 1 (Dec 2014), 24.

[139] Workspace, G. Gmail: Free, Private & Secure Email. `https://www.google.com/intl/en_us/gmail/about/`, 2022. Accessed 10 January 2022.

[140] Workspace, G. Google Docs: Free Online Document Editor. `https://www.google.com/intl/en/docs/about/`, 2022. Accessed 10 January 2022.

[141] Xia, W.; Quek, T. Q. S.; Guo, K.; Wen, W.; Yang, H. H.; Zhu, H. Multi-armed bandit-based client scheduling for federated learning. *IEEE Transactions on Wireless Communications 19*, 11 (2020), 7108–7123.

[142] Xu, J.; Du, W.; Jin, Y.; He, W.; Cheng, R. Ternary compression for communication-efficient federated learning. *IEEE Transactions on Neural Networks and Learning Systems* (2020), 1–15.

[143] Xu, M.; Yoon, S.; Fuentes, A.; Park, D. S. A comprehensive survey of image augmentation techniques for deep learning. *Pattern Recognition 137* (2023), 109347.

[144] Yang, H. H.; Liu, Z.; Quek, T. Q. S.; Poor, H. V. Scheduling policies for federated learning in wireless networks. *IEEE Transactions on Communications 68*, 1 (2020), 317–333.

[145] Yang, Q.; Liu, Y.; Chen, T.; Tong, Y. Federated Machine Learning: Concept and Applications. *ACM Trans. Intell. Syst. Technol. 10*, 2 (Jan. 2019).

[146] Yin, L.; Zhou, J.; Sun, J. A stochastic algorithm for scheduling bag-of-tasks applications on hybrid clouds under task duration variations. *Journal of Systems and Software 184* (2022), 111123.

[147] Yu, M.; Liu, J.; Wu, C.; Ji, B.; Bentley, E. Toward efficient online scheduling for distributed machine learning systems. *IEEE Transactions on Network Science and Engineering* (2021), 1–1.

[148] Zhang, H.; Stafman, L.; Or, A.; Freedman, M. J. Slaq: Quality-driven scheduling for distributed machine learning. In *Proceedings of the 2017 Symposium on Cloud Computing* (New York, NY, USA, 2017), SoCC '17, Association for Computing Machinery, p. 390–404.

[149] Zhang, Q.; Zhou, R.; Wu, C.; Jiao, L.; Li, Z. Online scheduling of heterogeneous distributed machine learning jobs. In *Proceedings of the Twenty-First International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing* (New York, NY, USA, 2020), Mobihoc '20, Association for Computing Machinery, p. 111–120.

[150] Zhang, X.; Wang, J.; Joshi, G.; Joe-Wong, C. Machine learning on volatile instances. In *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications* (2020), pp. 139–148.

[151] Zhao, X.-P.; Jiang, R. Distributed machine learning oriented data integrity verification scheme in cloud computing environment. *IEEE Access 8* (2020), 26372–26384.

[152] Zhou, A. C.; Lao, J.; Ke, Z.; Wang, Y.; Mao, R. Farspot: Optimizing monetary cost for hpc applications in the cloud spot market. *IEEE Transactions on Parallel and Distributed Systems 33*, 11 (2022), 2955–2967.

APPENDIX A – Complementary studies

During this PhD, there were complementary studies that served as motivation or inspiration to this thesis.

In [130] and [129], we analyse the checkpointing recording and dump time of three different storage services from AWS, namely the Amazon Elastic Block Storage (EBS), the Amazon Simple Storage Service (S3) and the Amazon Elastic File System (EFS). Our results showed that the EBS has the fastest dump time to store a checkpoint, but it does not allow concurrent access. Comparing the concurrent access in EFS and S3, we observed that S3 handles better concurrent access than EFS. Regarding costs, EFS is the most expensive of all three although it presents faster times when a single task is using it. Thus, we concluded that the best cost-benefit storage service for a concurrent application is S3, but these results motivated the present thesis as the best storage service depends on the access pattern to it.

In [18], we proposed a framework to execute a Sequence Alignment application in clouds, using both Spot and on-demand VMs in AWS only. This application has an application-level checkpoint and executes in a single VM with GPU. This simple framework used a slowdown concept that represented the slowdown the application had when executing in VM a compared to VM b. This slowdown was an inspiration to our execution and communication slowdowns described in Chapter 6 (Section 6.1. Moreover, this framework selected the application VM using greedy heuristics in both initial scheduling and when occurred a revocation, the latter being the inspiration to our Dynamic Scheduler algorithms (Section 6.4). Finally, we conducted several experiments with 5 different VMs of AWS and observed revocation patterns in two instance types, which led us to create three revocation rates in this study, $1/(2hours)$, $1/(4hours)$ and $1/(6hours)$, used in the experiments in this thesis (7.6). This study was also a motivation to schedule FL applications in a multi-cloud environment as we observed that different GPUs in the same cloud have different execution times, which makes the exclusive use of Spot VMs in a single-cloud prohibitive.

APPENDIX B – Published Papers

- Teylo, L.; **Brum, R.**; Arantes, L.; Sens, P.; Drummond, L. Developing Checkpointing and Recovery Procedures with the Storage Services of Amazon Web Services. *16th International Workshop on Scheduling and Resource Management for Parallel and Distributed Systems*, ICPP, 2020.

- **Brum, R.**; Bernardini, F.; Alves, M.; Drummond, L. Using Machine Learning Techniques to Classify the Interference of HPC Applications in Virtual Machines with Uncertain Data. *XXI Simpósio em Sistemas Computacionais de Alto Desempenho*, WSCAD, 2020.

- Teylo, L.; **Brum, R.**; Arantes, L.; Sens, P.; Drummond, L. Avaliação dos Serviços de Armazenamento da Amazon Web Services para Gravação e Recuperação de Checkpoints. *XXI Workshop de Testes e Tolerância a Falhas*, WTF, 2020.

- **Brum, R.**; Sousa, W.; Melo, A.; Bentes, C.; de Castro, M.; Drummond, L. A Fault Tolerant and Deadline Constrained Sequence Alignment Application on Cloud-Based Spot GPU Instances. *27th International European Conference on Parallel and Distributed Computing*, EuroPar, 2021.

- **Brum R.**; Drummond L.; Castro M.; Teodoro G. Towards Optimizing Computational Costs of Federated Learning in Clouds. *1st Workshop on Cloud Computing*, SBAC-PAD, 2021.

- **Brum, R.**; Teodoro, G.; Drummond, L.; Arantes, L.; Castro, M.; Sens, P. Evaluating Federated Learning Scenarios in a Tumor Classification Application. *VII Escola Regional de Alto Desempenho do Rio de Janeiro*, ERAD-RJ, 2021.

- **Brum, R.**; Arantes, L.; Castro, M.; Sens, P.; Drummond, L. Evaluating Execution Times and Costs of a Federated Learning Application on different Cloud Providers. *Conférence francophone d'informatique en Parallélisme, Architecture et Système*, COMPAS, 2022.

- **Brum, R.**; Sens, P.; Arantes, L.; Castro, M.; Drummond, L. Towards a Federated Learning Framework on a Multi-Cloud Environment. *2nd Workshop on Cloud Computing*, SBAC-PAD, 2022.

- **Brum, R.**; Sens, P.; Arantes, L.; Castro, M.; Drummond, L. Optimizing Execution Time and Costs of Cross-Silo Federated Learning Applications with Datasets on different Cloud Providers. *34th International Symposium on Computer Architecture and High-Performance Computing*, SBAC-PAD, 2022.

- **Brum, R.**; Castro, M.; Arantes, L.; Drummond, L; Sens, P. Multi-FedLS: a Framework for Cross-Silo Federated Learning Applications on Multi-Cloud Environments. *Journal of Parallel and Distributed Computing*, JPDC (in review).

- Vasconcelos, A.; **Brum, R.**; Paes, A.; Drummond, L. Detecção de Depressão nas Mídias Sociais usando Transformers com Aprendizado Federado. *VIII Escola Regional de Alto Desempenho do Rio de Janeiro*, ERAD-RJ, 2023.

- **Brum, R.**; Teylo, L.; Arantes, L.; Sens, P. Ensuring Application Continuity with Fault Tolerance Techniques (book chapter). *High Performance Computing in Clouds*, Springer, 2023.

- Sousa, W.; Soares, F.;**Brum, R.**; Figueiredo, M.; Melo, A; Castro, M.; Bentes, C. Biological Sequence Comparison on Cloud-Based GPU Environment (book chapter) *High Performance Computing in Clouds*, Springer, Cham, 2023

- Vasconcelos, A.; Drummond, L; **Brum, R.**; Paes, A.. Exploring Federated Learning to Trace Depression in Social Media with Language Models *The 2023 Chicken-egg HPC/DL Workshop*, SBAC-PAD, 2023.