Comparing Model sharing and data sharing approaches in decentralized learning systems

Scalable Computing Systems Laboratory (SaCS)

June 10, 2022
School of Computer and Communication Sciences, Lausanne, Switzerland
École Polytechnique Fédérale de Lausanne

Yassine Abdennadher
SCIPER:299273

Project Advisor: Professor Anne-Marie Kermarrec
Project supervisors: Dr.Rafael Pires and Rishi Sharma

Lausanne, EPFL, 2022

To you, who is reading this . . .

Contents

1 Introduction

Machine learning (ML) techniques are a subset of Artificial Intelligence techniques. ML focuses on studying structures in data to enable learning, reasoning, and decision making without human input. Simply defined, ML allows a user to submit massive amounts of data to an algorithm, which then analyzes and makes data-driven suggestions and decisions. The learning algorithm can use additional data to improve its decision-making in the future [1]. As the data size increases, the centralized ML algorithms cannot scale. Indeed, training a ML model with terabytes of data is compute intensive and memory consuming. One promising alternative to address this problem is a decentralized machine learning [2].

Decentralized learning is a ML technique that trains an algorithm on multiple distributed machine's nodes. Each node performs its training on its local data. Once each node has its own model, they start exchanging model parameters (such as weights and biases) to improve their performance metrics. Decentralized ML algorithms improve node's model while keeping the learning complexity reasonable. The main benefit of such learning is the balance it provides between the training speed and the model accuracy while keeping data privacy. For this, this machine learning approach favors sharing the weights parameters of our learning model rather than raw data. Often, companies do not want to share their data with third parties because they are considered confidential. Let's take for example the case of hospitals that want to have a prediction model on the impact of a disease on different patients. Each hospital trains its model on its own collected data except that it is possible that these data do not cover special cases that would be present in the neighboring hospital [3]. In this case, data sharing could improve the performance of the model but compromises data privacy.

To summarize, the messages exchange between neighbours can be done in two different ways:

- Model sharing: Nodes receive models parameters from their neighbors and use an average technique described in Chapter 3 to average them.
- Data sharing: Nodes receive data samples from their neighbors and add them to their local data set for future training steps.

This project compares the two previous approaches (model sharing, i.e. model parameters exchanging and the data sharing, i.e raw data exchanging). The goal is to improve convergence and performance while keeping network traffic low. Concretely speaking, this project aims to analyze these two schemes and select the most appropriate approach and implementation that leads to the best result for a given configuration. For this purpose, we will rely on a Deep learning algorithm (DL) and two data sets named Celeba and Femnist that come from Leaf, a benchmarking framework for learning in decentralized settings [4].

The structure of this report is as follows: Chapter 2 presents the Deep Learning algorithm (DL) used in this work and introduces the decentralized learning algorithms. Chapter 3 details the work carried out within this project, provides an overview of the architecture of the whole system and describes the two models: data sharing and model sharing.

Chapter 4 presents the experimental results. It compares the performances of the model sharing with three implementations of the data sharing model. The performance in expressed in term of training loss, testing loss, test accuracy etc. Finally, Chapter 5 details the perspectives of this work.

2 Background

This chapter presents the foundation architecture and the theory of our decentralized framework that uses decentralized learning. First, we briefly describe the machine learning algorithm used , namely convolutional neural network. Additionally, we explain the advantage of decentralized learning compared to centralized learning.

2.1 Deep Learning

Deep learning refers to a class of machine learning techniques that employ numerous layers to extract higher-level features from raw data. Lower layers in image processing, for example, may recognize edges, whereas higher layers may identify human-relevant concepts like numerals, letters, or faces. Learning can be supervised or unsupervised. In our case, learning will be supervised. Indeed, our raw data that will train is labled [5] . Chapter 3 provides more detail about data sets .

2.1.1 Convolutional neural network

Convolutional neural networks are known to be efficient with picture, speech, or audio signal inputs sets. They have three different types of layers:

- **Convulotional layer**: A Convolutional Neural Network's first layer is usually a Convolutional Layer. Convolutional layers perform a convolution on the input before forwarding the output to the next layer. If you apply a convolution to an image, for example, you will reduce the size of the image while also bringing all of the information in the field together into a single pixel.
- **Pooling layer** : Pooling layers have the same goal as convolutional layers. They

use functions such as max pooling, which takes the largest value in a certain filter region, or average pooling, which takes the average value in a particular filter region. These are commonly employed to lower the network's dimensionality.

- **Fully Connected layer** : Fully connected layers are used to flatten the data before classification and are inserted before the last layer of a CNN.

A convolutional network's first layer is the convolutional layer. While further convolutional layers or pooling layers can be added after convolutional layers, the fully-connected layer is the last layer. The CNN becomes more complicated with each layer, detecting larger areas of the image. Earlier layers concentrate on basic elements like colors and borders. As the visual data passes through the CNN's layers, it begins to recognize larger elements or shapes of the object, eventually identifying the desired object. However, you have to be careful not to have several layers because otherwise the model may overfit .

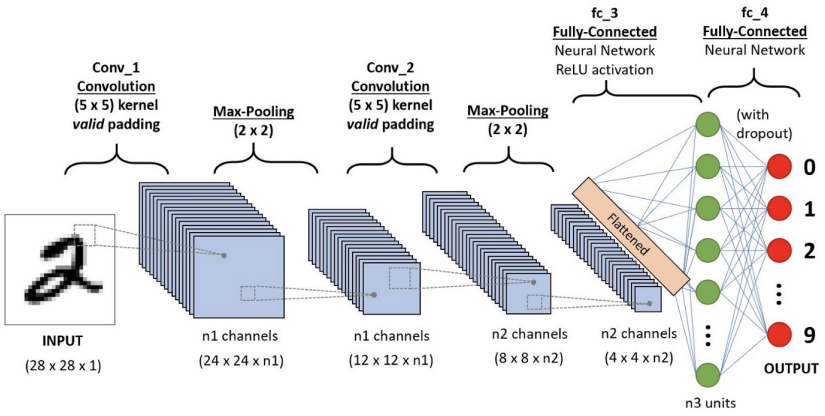


Figure 2.1: A CNN sequence to classify handwritten digits [7]

The two data sets that we will use in Chapter 4 for evaluation has 1.6 millions of parameters and 2.8k of parameters respectively. They use convolutional layer, pooling layer and linear layer. The number of parameters is the count of learnable parameters. They are weights that are learnt during training which will be detailed in the next chapter [6].

2.2 Decentralized learning

As I introduced, the main benefit of Decentralized learning is that if we have a huge amount of data that can't be stored in a unique machine, you can split the data among multiples machines and thus you have much more memory. Moreover, in the case

of model sharing, data privacy is guaranteed because nodes share only the model parameters. The first step consist of building a model that all processes of each machine will use. Then , you push this model out and nodes will train the model on their own data set. One should not confuse Federated learning and Decentralized learning. The main difference between the two lies in the fact that in federated learning we have a central entity that receive information from all nodes and merge it to create a whole unique model [8]. Indeed, in Decentralized learning, nodes communicate directly with each other through TCP connection for example. They send and receive data only from their neighbors. To establish who communicates with whom, we use a a directed graph. The graph can be ring, grid or fully connected for example.

Our system combines previously described approaches and algorithms. Data is uniformly distributed to nodes in the system. Usually,in decentralized learning, local data are not exchanged because of privacy. Indeed, we share only model's parameters i.e weights and bias of each CNN layer but in our case , the goal is to compare model and data sharing so processes can send model's parameters or directly raw data . The new samples will be added to their local data set. Model aggregation is implemented with D-PSGD protocol, described in the next chapter. The SaCS (EPFL) lab already provides the implementation of decentralized communication between nodes in the system and model averaging, and we further use them over our framework.

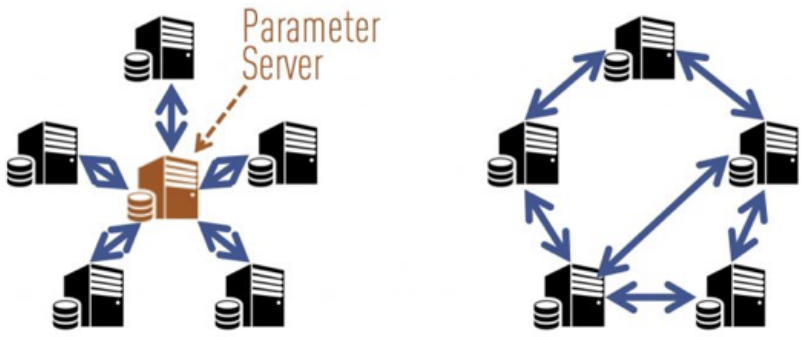


Figure 2.2: Federated learning (left) vs Decentralized learning (right) [9]

3 Implementation

This chapter details the architecture and the implementation of the system developed within this project. A special interest is given to the data sharing model as it has been the main focus of this project. The chapter is structured into five sections: the global architecture (including the Deep Learning algorithm on which our system relies and the data set used for training and testing), the model sharing, the data sharing, the performance metrics and the configuration parameters of the system .

It's worth noting here that the whole system is implemented in Python3. Communication is via TCP/IP sockets.

3.1 System Architecture

The system is composed of a set of connected nodes that behave in the same way. Each node has a set of neighbours and initially hosts a set of data. Roughly speaking, the algorithm executed by each node is as follows:

In case of data sharing, each node iterates through the following three steps:

1. receives a subset of data from each neighbour,
2. Starts the learning algorithm using the local data and the received data. Future learning steps are based on the model generated during previous iterations.
3. Sends back to the neighbours a subset of its local data set

In case of model sharing, each machine iterates through the following three step:

1. Receives from each neighbour its model's parameters.

2. Starts the learning algorithm after aggregating all the received models with the local model

3. Sends back to the neighbours the result of the learning (model's parameters)

In both cases, the workflow of the system (1, 2 and 3 explained above) is illustrated in Fig. 3.1

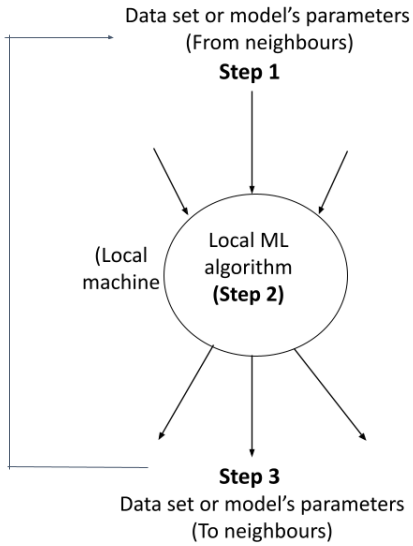


Figure 3.1: Operations that each node perform

In case of data sharing, The system must ensure that there is no data duplication (received from the neighbours) when re-training.

3.1.1 Data Representation

This project used two data sets, namely *Celeba* and *Femnist*. The CelebFaces Attribute Data set (Celeba) is a large facial attribute data set containing celebrity images, each with 40 attribute annotations. The images in this data set cover large pose variations and background clutter. The size of each image is 84 x 84 pixels [10]. On the other hand, Femnist contains 62 different types of handwritten digits and letters (digits 0 to 9, lowercase letters, and uppercase letters) of 3500 users[11]. The size of each image is 28 x 28 pixels. We have about 750.000 samples for Femnist and about 75.000 samples for Celeba.

Finally, a global metric is calculated by averaging the local metrics of all nodes.

3.1.2 Convolutional neural network

This section gives a global overview of the convolutional neural network model (CNN) used as a backbone by our system (Step 2 in Fig. 3.1). The model uses the standard PyTorch approach of forwarding pass, computing the loss, back propagating and updating weights. We use also the Adam optimizer with a 0.001 learning rate and cross entropy loss since we are dealing with classification problems.

Regarding Celeba (Fig. 3.2), the CNN is composed of four convolutional layers followed by a max pooling layer and one flatten layer. it also uses two types of activation functions: ReLU + Linear.

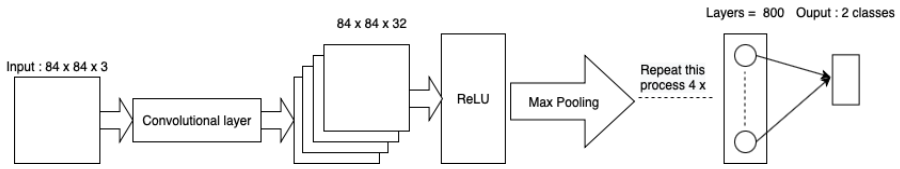


Figure 3.2: CNN model for Celeba

On the other hand, Femnist has two convolutional layers followed by a max pooling layer and 2 flatten layers and use too two types of activation functions: ReLU + Linear (Fig. 3.3).

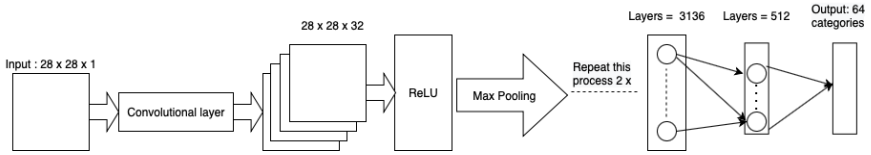


Figure 3.3: CNN model for Femnist

3.2 Model sharing

The aim of the model sharing is to avoid the data exchange. The idea is to protect data that is usually collected and stored at the node level. This is a data privacy issue. As stated earlier, the goal of decentralized learning is that the data is distributed over nodes and each node train on its own data set. After this process, they should communicate between each other in order to improve the performance of the collective model. In this section, we will focus on the technique used by nodes to handle weight's parameters received from their neighbors. This technique is called Decentralized Stochastic Gradient Descent and Model aggregation.

3.2.1 D-PSGD and Model Aggregation

In a decentralized situation, decentralized SGD (Stochastic Gradient Descent) solves the optimization problem of finding the minimal loss function by changing parameters one by one for each data point. After a step of training on local data, nodes in the Decentralized Parallel Stochastic Gradient Descent (D-PSGD) algorithm average their local model with that of all their neighbors. After that, each node communicates its models as well as an integer equal to its degree (number of neighbors) to all of its neighbors. When a model is received, it is blended with its own using a weighted average. To compute this, we use Metropolis- Hastings weights. Indeed, each node attribute weights to the contribution of its N neighbors and that of itself. Indeed,

$$a_e^{m+1} = w_e a_e^m + \sum_{i=1}^{N} w_i e_i^m$$

where a_e^m represent the model's parameter updated at iteration m that belong to node e, $w_i = \frac{1}{1+max(d_e,d_i)}$ with d_i equals to the degree of node i and with $w_e = 1 - \sum_{i=1}^{N} w_i$ [12].

3.3 Data sharing

As introduced, the purpose of this work is to compare the performance of our machine learning algorithm in decentralized learning between model sharing and data sharing. Instead of sharing the model parameters with your neighbors, data sharing consists of sending directly an amount of data samples. As introduced in chapter **??**, the goal of this project is to send as many bytes for data sharing as for model sharing through the communication channel. Before updating their local data set, nodes must check any duplicates. The objective is to minimize duplicates as much as possible because they bias your fitted model and will lead to the model overfitting since your model will learn on the same data. To remedy this, we will introduce three different techniques that we used in order to check duplicates:

- **Indices method**: This method works as follows: at the beginning of the whole process, we initialize a list per node that will be updated at each iteration step. After choosing the samples to send among his own data set, he will store the index of each of theses samples in the list. In case an index already belongs to this list, this data sample associated to this same index will be dropped and therefore it will not be sent. This process is repeated at each iteration before sending data.

- **Hash function method**: The goal of this method is to compute the hash of each data sample and check duplicates using their hash. Here, the hash used is SHA-256 (Secure Hash Algorithm). Note that the collision risk of this hash function is very low or even null.

- **Total comparison method**: After merging the local data set and the new samples received, each sample of this concatenated data set will be compare to the others pixel by pixel. This method might be time consuming.

After receiving data samples and checking duplicates, the training step is done on the whole data set or on some mini-batches. As nodes share data samples, if the learning is done on the whole data set, the amount of time needed for every iteration continually increases with increased data since nodes train on all local data sets.

3.4 Performance metrics

The performance of the system is assessed according to the following metrics:

- *Training loss*: The training loss is a metric used to assess how a machine learning model fits the training data. That is to say, it assesses the error of the model on the training set.

- *Testing loss*: The testing loss is a metric used to assess how a machine learning model fits the test data. That is to say, it assesses the error of the model on the testing set.

- *Testing accuracy*: Accuracy is defined as the percentage of correct predictions for the test data. It can be calculated easily by dividing the number of correct predictions by the number of total predictions. $accuracy = \frac{correctpredictions}{allpredictions}$

- *Run time*: run time of the whole process (Training + Sharing + Updating + Testing)

- *Network traffic*: total size of sent packets to all neighbours (in Bytes).

- *Total memory used*: total memory allocated to that process and is in RAM (in MegaBytes).

3.5 System configuration

The system is assumed to be generic and support different data sets, network topologies, number of nodes in the system, learning algorithms, etc. For this to happen, the following parameters are provided by the user:

- **Data set**: data set used for training and evaluation ; currently, the implemented data sets are Femnist and Celeba
- **ML model**: machine learning algorithm each node uses for training and testing ; currently, only CNN model is implemented
- **nodes**: number of nodes per machine
- **machines**: number of machines
- **Communication rounds**: number of communication rounds
- **Network topologies**: nodes topology. It is possible to select one of the following implementations: grid and ring
- **Sharing method**: Choose between data and model sharing. If data sharing is chosen, choose between the three different implementations.
- **Test after**: the number of times the test on the testing set is performed. This number is at most equal to the number of epochs.
- **Communication**: This parameter specifies the communication protocol supported by the nodes ; currently TCP implementation is implemented
- **Batch size**: the number of samples that will be propagated through the network
- **Learning rate**: the chosen learning rate.
- **Training configuration**: There are two types of training configurations:
  - Epoch: Epoch configuration is when the entire data set is passed forward and backward through the network only once.
  - Step: Step configuration is when 20 mini-batches taken randomly from the training set are passed forward and backward through the network.

The next chapter presents the experimental results from simulations using the above parameters.

4 Evaluation

This chapter demonstrates the system's evaluation in several situations as well as the experimental methods used. We provide an extensive set of experiments that each test different aspects of the system. We start with Celeba data set and demonstrate with plots the benefits of each sharing method, then we show its behavior in different topologies, how network traffic changes, how much memory was being used at the time, how many new samples were actually added and how processing time change. Afterwards, we do the same for Femnist data set. In each experiment, we describe its setup followed by the results and corresponding assessments. Finally we will end this chapter by a discussion where we will analyze our experimental results.

4.1 Experimental environment

This section describes the experimental environment used to compare the performance of data sharing and model sharing. These comparison relies on the metrics defined in chapter 3

4.1.1 Network Topologies

For evaluation, two different topologies are chosen among those that may resemble what one could find in practice: Ring graph and Grid Graph.

- *Ring*: If every node in the network is connected to its immediate rank neighbors, the topology of N nodes is called a ring topology. To confirm this, N must be greater than three and all nodes must have a degree of two. It is shaped like a Ring.

- *Grid*: Here, each node is connected to its spatial neighbors when the nodes are arranged in a square grid. The number of nodes N must be a square of an integer. This look like a N x N matrix.

4.1.2 Test Environment

To run our experiments, machines labostrex 125, 127-130 (in the IC cluster at EPFL) are used. They have the same specifications with processor Intel Xeon CPU E5-2630 v3 at 2.40GHz and 128GB of memory running Ubuntu 20.04.3 LTS kernel 5.4.0-99.

4.2 Experimental results

This section describes the experimental results related to the four following deployments:

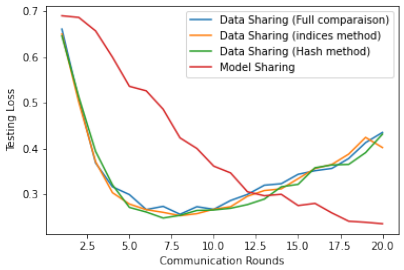
1. Celeba on a grid network composed of 4 machines with 4 nodes per each machine (16 nodes)
2. Celeba on a ring network composed of 4 machines with 4 nodes per each machine (16 nodes)
3. Femnist on a grid network composed of 5 machines with 5 nodes per each machine (25 nodes)
4. Femnist on a ring network composed of 5 machines with 5 nodes per each machine (25 nodes)

For the last deployment, we experimented two numbers of iterations: 25 and 45. We compare the performance of the two approaches (model and data sharing) in the case of these four deployments. The performance is compared with respect to the six metrics defined in section 3.1. A 7th parameter was also considered: number of samples used for the training. Concretely speaking, each experiment/deployment will be evaluated with respect to seven graphs.

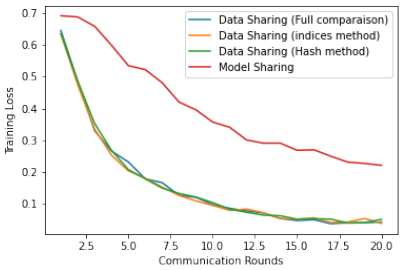
4.2.1 Results obtained on Celeba with 16 nodes and 4 machines on a Grid graph

- *Data set*: Celeba

- *Number of machines*: 4
- *Number of nodes*: 16
- *Number of tests*: 20
- *Number of nodes per machine*: 4
- *Network topology*: Grid
- *Number of communication rounds*: 20
- *Number of tests*: 30
- *Batch size*: 64
- *Optimizer*: Adam
- *Learning rate*: 0.001
- *Configuration*: Epoch configuration

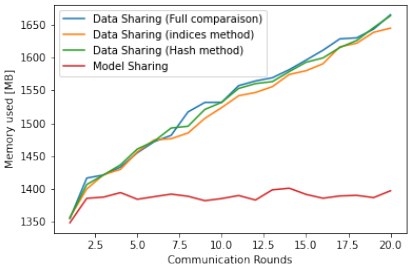


(a) Testing loss average (16 nodes)

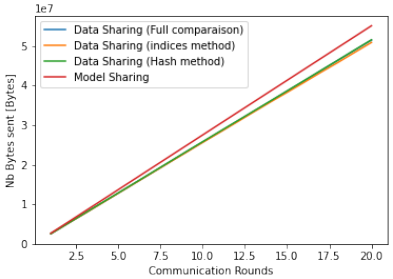


(b) Training loss average (16 nodes)

Figure 4.1: Training and Testing loss

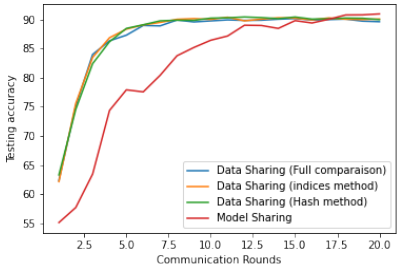


(a) Average memory size in MB (16 nodes)

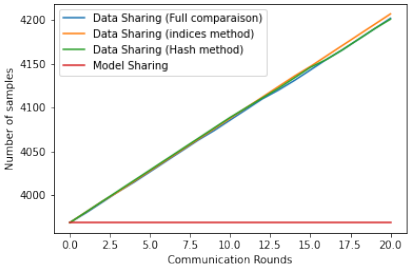


(b) Network traffic in bytes

Figure 4.2: Memory allocation and network traffic



(a) Average of Testing accuracy (16 nodes)



(b) Evolution of the number of samples during the process for one given node

Figure 4.3: Average testing accuracy and evolution of the number of samples for one node

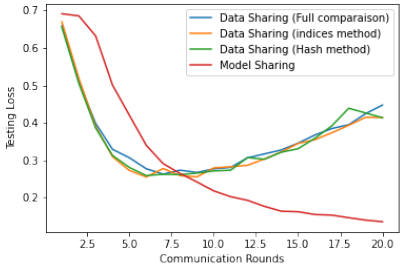
Sharing	Run Time
Data (hash)	45 min
Data (indices)	44 min
Data(total comparaison)	55 min
Model	40 min

(a) Run time comparison

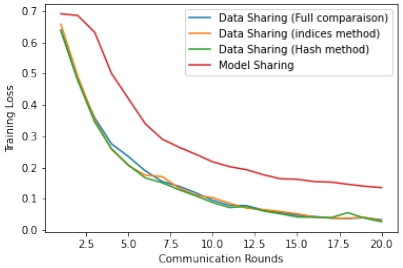
Figure 4.4: Run time

4.2.2 Results obtained on Celeba with 16 nodes and 4 machines on a Ring graph

- *Data set*: Celeba
- *Number of machines*: 4
- *Number of nodes*: 16
- *Number of tests*: 20
- *Number of nodes per machine*: 4
- *Network topology*: Ring
- *Number of communication rounds*: 20
- *Number of tests*: 20
- *Batch size*: 64
- *Optimizer*: Adam
- *Learning rate*: 0.001
- *Configuration*: Epoch configuration

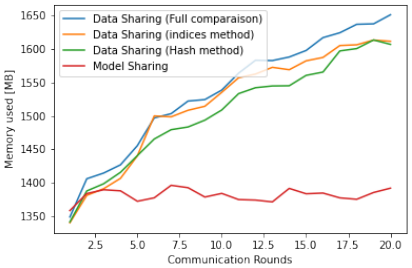


(a) Testing loss average (16 nodes)

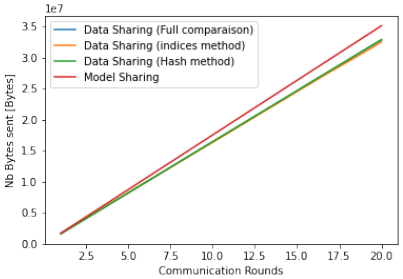


(b) Training loss average (16 nodes)

Figure 4.5: Training and Testing loss

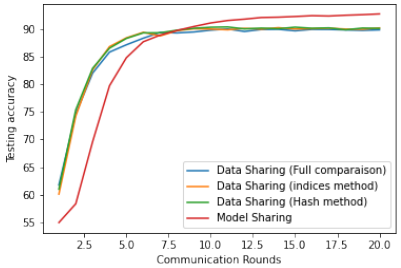


(a) Average memory size in MB (16 nodes)

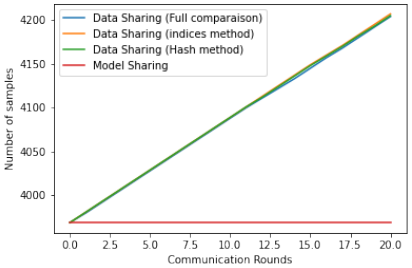


(b) Network traffic in bytes

Figure 4.6: Memory allocation and network traffic



(a) Testing accuracy average (16 nodes)



(b) Evolution of the number of samples during the process for one given node

Figure 4.7: Average testing accuracy and evolution of the number of samples for one node

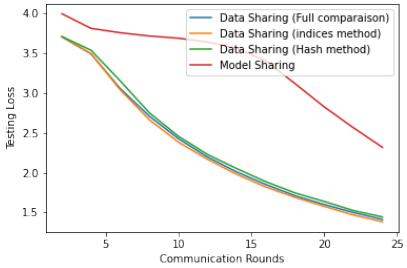
Sharing	Run Time
Data (hash)	46 min
Data(indices)	38 min
Data(total comparaison)	50 min
Model	40 min

(a) Run time comparison

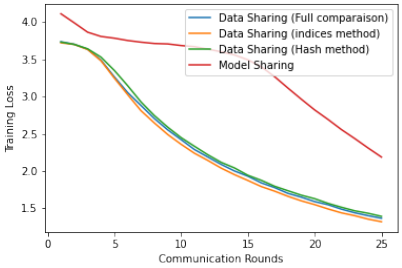
Figure 4.8: Run time

4.2.3 Results obtained on Femnist with 25 nodes and 5 machines on a Grid graph

- *Data set*: Femnist
- *Number of machines*: 5
- *Number of nodes*: 25
- *Number of nodes per machine*: 5
- *Network topology*: Grid
- *Number of communication rounds*: 25
- *Number of tests*: 12; test after 2 iterations
- *Batch size*: 64
- *Optimizer*: Adam
- *Learning rate*: 0.001
- *Configuration*: Step configuration

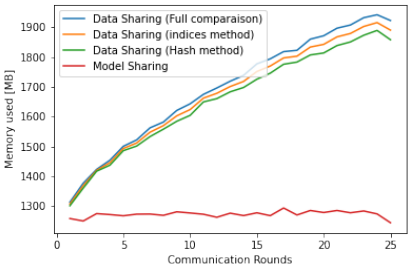


(a) Testing loss average (25 nodes)

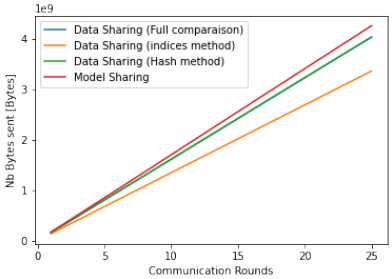


(b) Training loss average (25 nodes)

Figure 4.9: Training and Testing loss

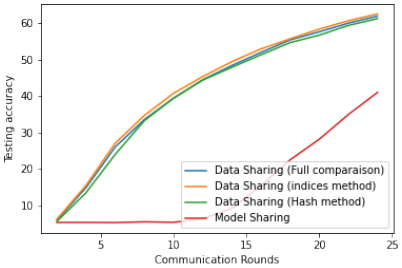


(a) Average memory size in MB (25 nodes)

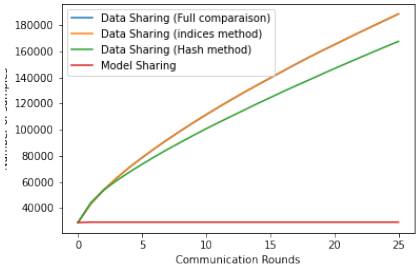


(b) Network traffic in bytes

Figure 4.10: Memory allocation and network traffic



(a) Testing accuracy average (25 nodes)



(b) Evolution of the number of samples during the process for one given node

Figure 4.11: Average testing accuracy and evolution of the number of samples for one node

Sharing	Run Time
Data (hash)	≈ 8 hours
Data(indices)	≈ 9 hours
Data(total comparaison)	≈12 hours
Model	≈2 hours

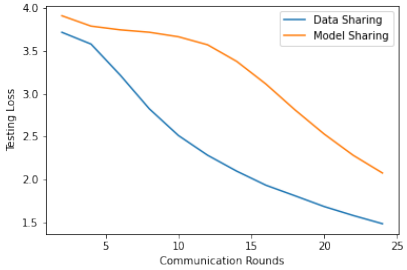
(a) Run time comparison

Figure 4.12: Run time

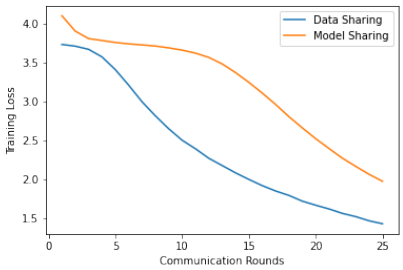
4.2.4 Results obtained on Femnist with 25 nodes and 5 machines on a Ring graph

- *Data set*: Femnist
- *Number of machines*: 5
- *Number of nodes*: 25
- *Number of nodes per machine*: 5
- *Network topology*: Ring
- *Number of communication rounds*: 25
- *Number of tests*: 12; test after 2 iterations
- *Batch size*: 64
- *Optimizer*: Adam
- *Learning rate*: 0.001
- *Configuration*: Step configuration

After noticing that the three different implementations of data sharing converged to the same values, we decided from now to consider only data sharing hash method for our further experiments.

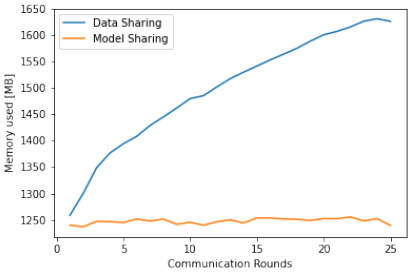


(a) Testing loss average (25 nodes) nodes

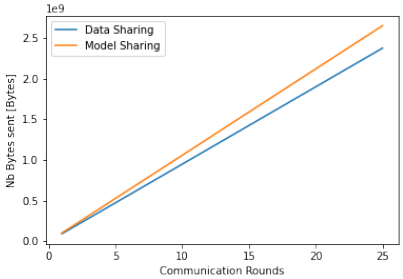


(b) Training loss average (25 nodes) nodes

Figure 4.13: Training and Testing loss

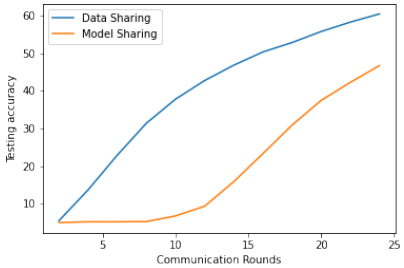


(a) Average memory size in MB (25 nodes)

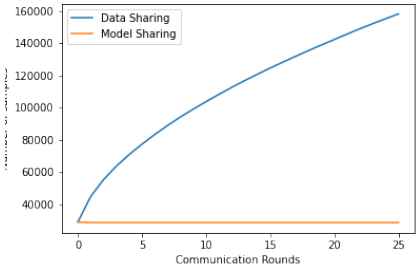


(b) Network traffic in bytes

Figure 4.14: Memory allocation and network traffic



(a) Testing accuracy average (25 nodes)



(b) Evolution of the number of samples during the process for one given node

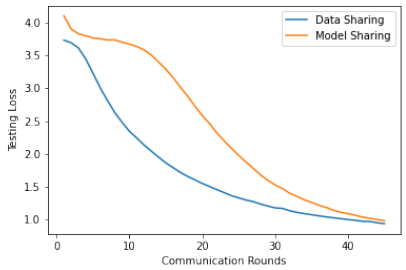
Figure 4.15: Average testing accuracy and evolution of the number of samples for one node

Sharing	Run Time
Data	≈ 5 hours
Model	≈ 2 hours

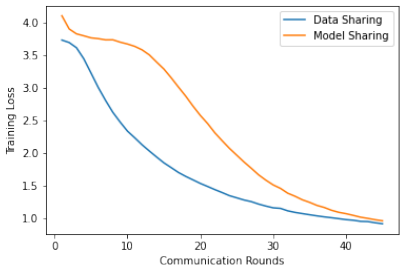
(a) Run time comparison

Figure 4.16: Run time

- *Data set*: Femnist
- *Number of machines*: 5
- *Number of nodes*: 25
- *Number of nodes per machine*: 5
- *Network topology*: Ring
- *Number of communication rounds*: 45
- *Number of tests*: 45
- *Batch size*: 64
- *Optimizer*: Adam
- *Learning rate*: 0.001
- *Configuration*: Step configuration

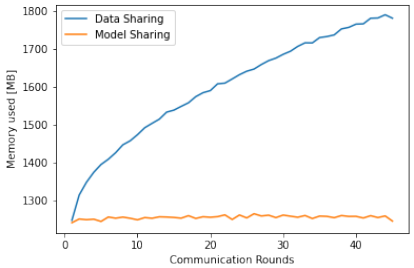


(a) Testing loss average (25 nodes) nodes

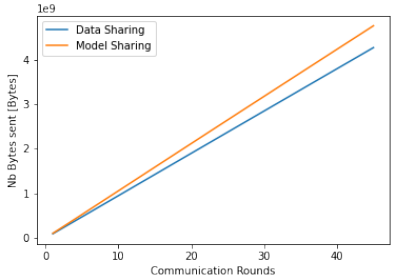


(b) Training loss average (25 nodes) nodes

Figure 4.17: Training and Testing loss

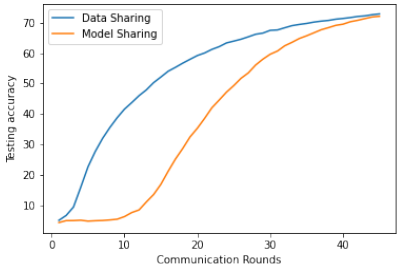


(a) Average memory size in MB (25 nodes)

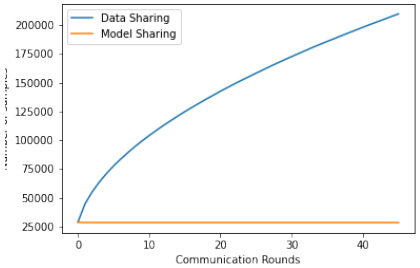


(b) Network traffic in bytes

Figure 4.18: Memory allocation and network traffic



(a) Testing accuracy average (25 nodes) nodes



(b) Evolution of the number of samples during the process for one given node

Figure 4.19: Average testing accuracy and evolution of the number of samples for one node

Sharing	Run Time
Data	≈ 8 hours
Model	≈ 2 hours

(a) Run time comparison

Figure 4.20: Run time

4.3 Discussion

According to figures (4.1, 4.3a, 4.5 and 4.7b), model sharing performs better than data sharing with Celeba while sending the same number of bytes for these two approaches. The curves on the figures mentioned above are very smooth and converges to an optimal solution without overfitting. The testing accuracy achieves 90% while the testing loss is smaller than 0.1. On the other hand, Feminist is better suited with data sharing. According to figures (4.9, 4.11a, 4.13, 4.15a, 4.17 and 4.19a), data sharing performs better on the training set as well as on the testing set.

These analyses refers to the following metrics: Testing loss, Training loss and Testing accuracy.

For Celeba, the overfitting is very prominent in the case of data sharing. The reason for this is that since the size of the exchanged data is limited, the data used in the learning process is almost the same, hence the overfitting. Indeed, each node sends 6 data samples per iteration per neighbor. As we can see on Figure 4.3b and 4.7b, the number of samples is 3900 at the beginning and 4200 at the end. We have only 6 % increase in the number of samples which is negligible compared to the initial number.

Overfitting does not occur with Femnist because we used a different training configuration. The difference is that in Celeba the epoch configuration is used. In this case, the model goes through the whole data set at each training step. On the other hand, at each step iteration, Femnist uses the step configuration which trains the model on 20 mini-batches randomly taken from the training set.

Regarding memory and run time, from the figures(4.2a, 4.4, 4.6a, 4.8, 4.10a, 4.12, 4.14a, 4.16, 4.18a and 4.20), data sharing is time intensive and memory consuming. This is due to the fact that data sharing, after receiving the packets, adds the new samples to its local data base, which increases the memory used. Regarding the run time, concatenating data samples and checking for duplicates takes more time than averaging the received model parameters.

5 Conclusion

The goal of this work was to compare the performance of two approaches used in decentralized learning: data sharing and model sharing. Data sharing consists of exchanging data among nodes to perform learning and generate inference models, while model sharing allows the system to exchange the models themselves and combine them to generate a new aggregated model. Data sharing has two shortcomings:

- The data is exchanged, which raises a privacy issue.
- The network traffic increases if the exchanged data size increases.

In this work, we have limited the network traffic of the data sharing approach to that generated in the model sharing. This constraint has led to an overfitting problem: since the exchanged data size is limited, and the learning process is iterative (Figure 3.1), data set used for learning is often the same. It would be worth testing the performance of data sharing by increasing the size of the data exchanged.

To avoid any duplication of data set during the training, our methodology was to filter received and/or sent data and discard any redundancy. For this purpose, we tested three approaches:

- Indices method
- Hash function method
- Total comparison method

Experimental results have shown that the choice of one of these approaches has no impact on the performance of the system.

The comparative study carried out within this project is based on six metrics, namely:

- Training Loss
- Testing Loss
- Testing accuracy
- Run time
- Network traffic
- Memory usage

5.1 Future work

Since the notion of data sharing is rather new, one of the possibilities to explore is that of data sharing using step configuration on 100-150 communication rounds to have potentially better accuracy than in section 4.2.4 and compare it with model sharing. We can analyze the testing loss and stop our algorithm when we notice that the curve does not evolve during 20 communications rounds for example. This is only possible if we have enough memory in our machine. Moreover, we can try to share model sharing + model Sharing and see if we converge to a faster result than the current model sharing. This would imply that the number of bytes sent per node through the communication channel would increase. One other possibility to explore is implement Software Guard Extensions (SGX) on data sharing to preserve data privacy. This is a set of security-related instruction codes that are built in CPUs. They permit customers and operating systems to define private areas of memory, known as enclaves, whose contents is inaccessible from the outside. [13]

6 Bibliography

1. Netapp. March 2021. *What is machine learning?*. https://www.netapp.com/artificial-intelligence/what-is-machine-learning/

2. TRAVIS ADDAIR. Stanford University.*Decentralized and Distributed Machine Learning Model Training with Actors*. https://www.scs.stanford.edu/17au-cs244b/labs/projects/addair.pdf

3. CEM DILMEGANI. May 29 2021. *What is Federated Learning (FL)? Techniques and Benefits in 2022*. https://research.aimultiple.com/federated-learning/

4. Sebastian SEBASTIEN CALDAS. 2019. *LEAF: A Benchmark for Federated Settings*.https://leaf.cmu.edu

5. https://en.wikipedia.org/wiki/Deep_learning

6. IBM Cloud Education. October 2020. *Convolutional Neural Networks*. https://www.ibm.com/cloud/learn/convolutional-neural-networks

7. ALI KADHUM M. AL-QURABAT. https://www.researchgate.net/figure/CNN-sequence-to-classi fig2_352014824

8. BRENDAN MCMAHAN and DANIEL RAMAGE. April 2017. *Federated Learning: Collaborative Machine Learning without Centralized Training Data*.https://ai.googleblog.com/2017/04/federated-learning-collaborative.html.

9. https://www.sneakersvip365.top/ProductDetail.aspx?iid=114700136&pr=45.88

10. ZIWEI LIU, PING LUO, XIAOGANG WANG and XIAOU TANG . *Large-scale CelebFaces Attributes (CelebA) Dataset*. https://mmlab.ie.cuhk.edu.hk/projects/CelebA.html

11. SEBASTIEN CALDAS. 2019. *LEAF: A Benchmark for Federated Settings*.https://leaf.cmu.edu

12. NEVENA DRESEVIC. June 2021. *The cost of sharing in decentralized recommender systems*. https://www.epfl.ch/labs/sacs/wp-content/uploads/2021/06/nevena.pdf

13. https://en.wikipedia.org/wiki/Software_Guard_Extensions