Frugal and robust computing resource distribution strategies in Federated Learning: a state-of-the-art

Student
Andrew Mary Huet de Barochez
LISTIC Laboratory
andrew.mary-huet-de-barochez@etu.univ-smb.fr

Supervisor
Sébastien Monnet
LISTIC Laboratory
sebastien.monnet@univ-smb.fr

Supervisor
Stéphan Plassart
LISTIC Laboratory
stephan.plassart@univ-smb.fr

Abstract

Federated Learning (FL) is a growing field of machine learning, permitting the scaling of heavy single-machine algorithms to multi-machine distributed system. The main benefit of this technology is the data-privacy by-design: client's data does not need to be shared among participants of the federated systems. Each client trains a local machine learning model with their own data without needing to share any of their private data with a third party. In the next step, an orchestrator aggregates those local models in order to create a global one with more performance. Then, the model is shared to the collaborators allowing improvements on the local models. A lot of research has been done on the optimization of the machine learning models and the way to aggregate them. However only a small part of the literature tackles the architecture of the federated systems themselves. This paper draws the state-of-the-art concerning architecture design for FL, especially focusing on frugality and robustness. In other words, we analyze the different path that leads to a less energy-consuming and more fault tolerant system.

1. Introduction

1.1. Theoretical Background

Presented in 2017 by google [1], *Federated Learning (FL)* gained a lot of attention due to its privacy and handling of heterogeneous data. It aims at training a Machine Learning (ML) algorithm with the help of multiple devices, each of them having their own local data, without explicitly sharing those. It is achieved by training local models on each client, then aggregating those models in order to create a global one, more efficient, with the knowledge of other clients (See Figure 1). The aggregation can differ from FL models to others, but the general principle is to aggregate the local models parameters into a global model. As instance, FedAvg [1] is using weighted arithmetic mean to aggregate the parameters. This means that the more a client trained with data, the more influence it has on the global model.

The need of privacy towards data is ever growing and common centralized machine learning models require to collect the data on one point which represents risks. This is the key motivation to develop FL. For example the approach could be particularly useful in fields such as medicine where data privacy is a strict requirement. We can imagine building a model for diseases recognition based on patient data from multiple laboratories.

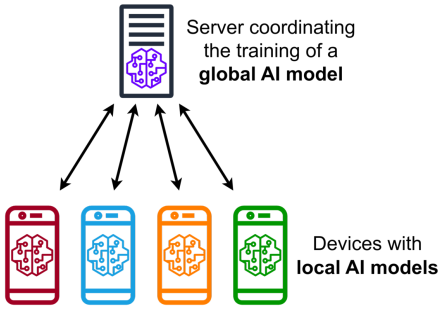


Figure 1: "Federated Learning by Simple diagram (in English) of a centralized Federated Learning protocol" by MarcT0K published under Attribution-ShareAlike 4.0 International license, no changes were made.

FL differs from Distributed Learning (DL) by its purpose: FL tries to solve the problem of learn-

ing with heterogeneous datasets, whereas DL is basically an upgrade in the dimensions of ML by adding scalability thanks to a Distributed System (DS) architecture on multiple machines.

In addition, FL focuses on the non-Identically and Independently Distribution (non-IID) aspect of data in the real world, which was not the case for DL, making it unpractical for many real case situations. Non-IID describes personalised and asymmetric probabilistic data distribution, varying a lot among clients making the generalisation and performance of a ML model harder.

Lastly, FL differs from DL by its environment of execution: while the second is almost always being executed on multiple machines in data centers, Federated clients on the other hand are executed on user devices, thus they face different constraints such as unreliability, dishonesty, be subject to failures, limited by their performances and or communication media.

Centralized and Decentralized FL

Until now, most of the research have been done on Centralized Federated Learning (CFL) – implicitly called FL – implementing an architecture based on a server orchestrator and multiple client devices. However, Decentralized Federated Learning (DFL) have been introduced, proposing a truly decentralized architecture, with clients collaborating into taking decisions concerning the training of a global model.

CFL:
- Predominant approach
- Central orchestration server creates and distribute a global model to the clients
- Clients train locally their own model and sends the parameters to a local server
- The server aggregates the models
- Simple architecture, easy maintenance and accountability

DFL:
- Distribute the aggregation of models between the neighboring participants
- Transmit fast updates computed locally by each nodes
- No single-point of failure, trust dependencies and bottlenecks at the server node
- Better fault tolerance because of constant updates
- Self scaling network

In the rest of the article, CFL will still be referred implicitly as FL.

Cross-silo, Cross-device and Hierarchical FL

FL implementations can be separated into three different categories:

Cross-silo designate a context with few clients with high system availability and low failure potential, likely similar data distribution and/or IID data: we can think of Hospitals consortium as an example. Each hospital is able to gather and pre-process their own data on one (or more) of their machine with correct performances. The machine has a correct network link and communication, which makes the learning process easier. Then all hospitals can learn a global model together with FL, or CFL to avoid having an orchestrator and emphasising collaboration and equity between hospitals while also providing control over the clients identity.

Cross-device, as its name tells, define a context where the system is mostly running on devices such as Internet of Things (IoT), personal computers, devices that are both trainer and user of the model, or all at the same time. The federated system can have thousands of clients with low system availability, high failure potential and various data distributions.

Hierarchical [2] is a mix of the two previous architectures, it is taking advantage of both IoT and cloud world. We can think of hierarchical as multiple silos defined by a myriad of IoT devices with sporadic availability (irregular, episodic, scattered...), linked to an edge server that performs partial model aggregations. Furthermore, silos models can be aggregated together by performing inter-silo aggregations in order to create a global model.

It is clear that **cross-device** and **hierarchical** systems represent more challenge in terms of ar-

chitectural design, robustness, and carbon footprint as they run on heterogeneous devices with less control over.

Carbon footprint of FL

In this study we will focus on the Carbon emissions of FL systems, contributing to the climate change. Yet it seems important to remind that climate change is only one of the 9 planet boundaries defined in 2009 [3]; and to this day 6 have been crossed [4].

Data centers represents 0.3% of overall carbon emissions, while the information and communications technology (ICT) ecosystem as a whole — which includes personal digital devices, mobile-phone networks and televisions — play for more than 2% of global emissions [5].

Classical ML play a strong role in data centers consumption: between 2012 and 2018 a growth of 300,000 has been noticed concerning the number of computation needed in order to train the last bleeding-edge ML model [6]. For example, on average each person on the planet emits 5 tonnes of CO2 equivalent [7], and the training of a large Natural Language Processing (NLP) transformer model with neural architecture search may produce 284 tonnes of CO2e [7].

1.2. Challenges

FL is still subject to scientific limitations that are sparsely and separately explored on this specific field.

Multiple security concerns [8] are yet to be explored in the FL field. For example it has been shown that data can be partially inferred from ML models [9]. This means that the particularity of FL that prevents data sharing between entities is not sufficient to provide guarantee of data-privacy if it can be inferred from the models.

The local optimisation of models are subjects to strong biases when the quantity of data is too low, too diverse and non-IID [10]. Furthermore, FL is often applied on clients with limited resources; it is relevant to optimize local models in frugal [11] and robust [12] ways in order to reduce optimization costs while facilitating the aggregation of the local models.

Some limitations are also imposed by the DS dimension: FL must take into account constraints of energy costs reduction (overall frugality). Furthermore, it is necessary to introduce resilience to breakdowns, latencies and agent unavailability. These constraints are particularly critical when applied in Internet of Things or embedded AI integrating low resources and availability [13].

1.3. Paper structure

Section **Methodology** introduces the procedure involved in the making of this paper. Section **SOTA analysis** discuss problems related to the subject and makes a synthesis of the different solutions available in the state-of-the-art (SOTA). Section **experiments** details future experiments. Section **Future work** tackles the blind spots that are needed to be explored, followed by the **Conclusion** section.

2. Methodology

In order to lead this state-of-the-art analysis, I used mind maps (see Annex 1) to structure my research, my thinking, the articles I found, and the knowledge I acquired.

At first I started by analysing the context of my study, the project related to it, and its different parts. I made a synthesis of the project goals and investigated on the different keywords used.

Secondly, I further analyzed my own part of the project and defined precisely the goals, and what I should accomplish.

Finally, I created a scope for my state-of-the-art analysis in order to create boundaries towards the articles I would chose to review. This helped me identifying relevant article to answer the problematics. Note that some of the articles have also been recommended by my supervisors.

The previous points will now be respectively covered more in depth in the following subsections.

2.1. Context of the study

This study is one of the state-of-the-art made for the project Dynamic and Resilient Riemannian Federated Learning (AFREU) in the LISTIC laboratory.

This project is based on 3 axes defined by different complementary concepts:

1. Frugal local learning based on Riemannian geometry
2. Reinforcement learning strategies for Federated Learning Robustness
3. Frugal and Robust computing resource distribution strategies

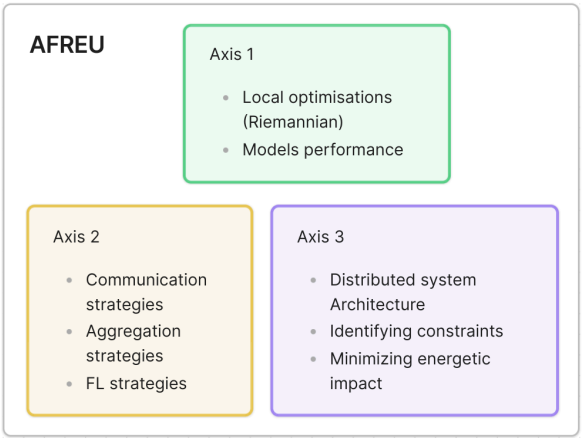


Figure 2: Schema of the AFREU project structure with each axis and their respective goals.

We define frugality as avoiding wasting, reducing resource consumption, scaling the solution to a problem correctly. In this work the frugality will be especially focused on reducing the energetic cost in order to reduce the carbon footprint of such a system.

Robustness designates the capacity of the system to be resilient to problems induced by the Distributed systems field such as latencies, failures, communication loss, dead locks etc. We also take into account the performance of the global model in the robustness.

Those two aspects can be contradictory, for example adding fault tolerance algorithms to each client would make the system more robust, nonetheless it would add more computation work, thus leading to more energy consumption and less frugality. Nevertheless, being resilient against faults can reduce recovery time and prevent doing some computation multiple times. In the end, we need to find a balance between frugality and robustness, in other words, finding the balance between saving energy and model performance.

The project goals by axis are represented by Figure 2, and can be summarized as follows:

- Improve **resilience** of FL in **dynamical systems**
- Improve FL **performances** in general
- Make FL more **fault-tolerant**
- **Reduce energetic cost** of FL

As the title of the article suggests, the third axis will be studied in this paper.

2.2. Study objectives

Objectives were defined for the third axis, and they were used as starting points in order to research and study the state-of-the-art of this field. The objectives are the following:

- Investigate federated system architectures proposed by the state-of-the-art
- Find ideal strategies to distribute computing resources in a **frugal** and **robust** way.
- Find ways to **measure** and/or **estimate** the energy consumption of the FL system.
- Study how **variation in heterogeneity** of the nodes impacts the system in terms of performance and electric consumption.
- Provide a better understanding on how **system constraints** such as latency, connection loss, dead locks and more impacts the model.

2.3. State-of-the-art scope

In this paper, a range of articles to be analyzed have been clearly defined in order to select relevant articles to be reviewed for this subject. The focus will be around a Distributed System view of FL. We won't dive into machine-learning specific description of the models, but more on how everything is working together in order to train a global model.

The constraints imposed by frugality and robustness are already deeply studied in other fields. Nevertheless, some work is starting to be done concerning those subjects in the FL field. The aim of this article is to compile the state-of-the-art for dealing with Distributed Systems constraints applied to FL.

3. SOTA analysis

In the following section, an analysis of the state-of-the-art of our defined topic will be held by

reviewing **Context**, **Problem**, **Solutions**, **Constraints** and **Results**.

Context introduces the setup in which we define our **Problem**. This last addresses the different challenges. **Solutions** provides a detailed view of the different solutions available in order to solve the given problem. **Constraints** tackle the different considerations to take into account with the given **Solutions**. **Results** summarize the different conclusions drew by the analyzed articles.

3.1. Carbon Footprint estimating and minimizing

Context: FL systems are meant to be run on different devices, with varying performance, resources and consumption. Furthermore those devices can be located in different countries and have varying communication signal.

Problem: We want to be able to estimate the carbon footprint of our system in order to minimize it. In a centralized ML configuration, it is fairly simple to estimate the impact of the training. Yet in a distributed setup, it is far more complex due to several constraints:

- **Location** of devices can vary, thus each device may be powered by more or less CO2-emitting energy
- **Consumption** of devices also varies, especially in an IoT setup. Furthermore it may be difficult to estimate the exact consumption produce only by the FL process.
- **Communication costs** includes the powering of switches and network infrastructures devices. The availability of such data can be hard to get, and is still obviously dependent on the **location**.
- **Hardware manufacturing** should be part of a rigorous Carbon Footprint analysis, yet in a FL setup it becomes almost impossible as devices comes from countless constructors, with a ridiculously tiny number of them providing CO2 informations concerning the production of a device.

Solutions: Several studies started to get work done concerning Carbon Footprint estimation in FL.

In [14] the CO2e analysis is performed by accounting the energy required by FL, which includes energy consumed by hardware (server and client devices) and communication energy (network powering, switches etc.). Then the amount is converted to CO2e emissions taking into account the energy source using geographical location. Nevertheless hardware manufacturing information is not taken into account as it remains largely unavailable.

For the energy consumption, GPU and CPU electric consumption is accounted during the training of the model. Energy consumption for hardware such as memory and storage were not taken into account because it is highly dependent on the devices, which can be multiple in a FL context. [15] claims those type of consumption can vary up to 10%.

The article [16] proposes similar energy-accounting methods, but further introduces ways to optimize devices in Federated setups. For example, the devices used in the experiment use low-power radio interfaces supporting deep sleep modes in order to lower the energy consumption.

Constraints:

While the two previous studies introduced good comparison methods, they still encounters some pitfalls. For example, in the first study the communication cost is only taken into account in FL, not classical ML. Indeed it is trivial to assume communication cost is higher for FL, but it also assumes that the big dataset used by the ML approach was already on the server and had no collecting protocol beforehand. This could play a small advantage towards FL as datasets are directly present on each clients that trains the model.

Furthermore this study [17] from 2015 states that cooling represents 40% of data centers consumption. This aspect also plays into favor of FL because clients does not need cooling when in IoT setup.

Results:

The first article describes that in most cases FL performs worse than ML - “FL can emit from 72%

to hundreds of times more carbon than its centralized version." - whereas the second draws better conclusions about the carbon emissions. This can be explained by the fact that the experiments have been applied to Internet of Objects (IoT) devices with lower power-consumption with sleep modes.

However a consensus is that the emissions of FL are highly dependent on the system architecture and parameters used. Here is a list of factors contributing to the increasing of emissions:

1. Usage of electricity produced with high CO2 emissions
2. Slow network connection between client and server
3. Low number of epoch client side
4. non-IID and heterogeneity of datasets

Solutions to fix 1. and 2. would be to only select nodes in locations with lower carbon electricity productions, and to only use geographically close devices in order to reduce communication costs. Nevertheless, both of these solutions comes with the drawback that it would lead to potential biased data as we are selecting subsets of our datasets with location criteria.

The points 3. and 4. worsen the emissions because they both leads to more training rounds meaning more communication among the whole federated system. The optimization of hyperparameters becomes very different: in classical ML we focus on convergence speed and performance of the model, but now we also need to take into account the communication costs and divergence of local models. In other words less local epochs means more communication costs and fewer model divergence, more local epochs means lower communication costs but more divergence with the global model.

Optimizing aggregation algorithms on the orchestrator can also be a solution in order to reduce the number of communication rounds.

3.2. CFL architectures

Context: Centralized Federated Learning systems can be expressed as for 4 steps. Local model training, parameter exchange, model aggregation and second parameter exchange for global model diffusion. In a centralized setup, the clients performs the two first steps, the server the two last, and the process is repeated until the global model converges. The most used topology in this part is the star topology because star networks architectures are the straight forward, reliable and most popular approach in computer networks in general. It is adapted from small to medium size networks, generally -but not limited to- in a cross-silo setup.

Problem: architecture design is crucial when building a FL system in order optimize the frugality and the robustness of the system. We saw in the previous section that communications are a major factor of energy consumption, choosing solutions and design patterns to reduce this factor is the most important point. However, finding robustness in communications often means reducing frugality, so trade-offs should be made. On the other hand, data heterogeneity also plays a role in the energetic consumption as it decreases convergence speed and makes the global model less robust.

Solutions:

The following patterns comes from the paper [18] and are analyzed here in the point of view of frugality and robustness.

Client registry let managing clients via a registry on the central server. The registry contains informations of connected client devices such as ID, connection UP and Downtime, device resource informations; it helps to keep tracking failed, dropout or dishonest nodes. This design grants a simple and reliable star network that work well with small to medium size networks. It also means that the server represents a single point of failure and a bottleneck for communication at each iteration of the model.

Likewise, **Client selector** is based on a star-topology, but the bottleneck problem is slightly mitigated because of the selection process: based on pre-defined criteria, the central server decides to exclude client devices on each round. Those criteria can include *resource optimization*, for example we could decide not to aggregate the N worse local models, thus avoiding wasting resource on aggregating low quality models

and reducing the amount of data being shared. Moreover, a *system performance* criteria could also be used to select only the clients with sufficient bandwidth in order to limit the probability of having crashing clients. Finally, *model performance* can also be brought to the fore by selecting local models with low data heterogeneity to produce an efficient global model.

Note that when a client is not selected, its local model has been computed for nothing. This is obviously contrary to our frugality constraint. Extra information and communication is also needed decision making.

Client cluster opt for a different approach by separating clients into different clusters based on their properties. The clients can be grouped by their data distribution, features similarities or gradient loss. This solution increase the global model performance for highly personalized tasks and provides a faster convergence speed. Thus, it reduces the number of training rounds, leading to less communications overall.

Asynchronous aggregator

Gives the sever the ability to perform model aggregation asynchronously, upon receiving a local model and without having to wait for the slowest client devices to finish training during an iteration. The advantage is that communication bottlenecks are reduced because the transmissions of models are spread out in time. Note that this advantage could be acquired also in a synchronous aggregation pattern by just sending the local models when they finished training, simply the aggregation wouldn't be performed right away.

The asynchronicity of the aggregation may lead to biases as the most efficient client devices will have their local models aggregated first. Furthermore, additional iterations of the models aggregation can be required to complete the learning process.

Secure aggregator

Provides security for model exchanges and aggregations using Secure Multiparty Computation (SMC) protocols. Those protocols ensure that each participants only knows their input and output, but from the others. Homomorphic encryption is often used as an example, yet it is a computation heavy algorithm that would slow dramatically the performance of the system. As a side note, Searchable Symmetric Encryption (SSE) could also be an entry point into implementing a SMC protocol for the models.

Hierarchical Aggregator

Adds an intermediate layer such as edge servers between client devices and the main server. Partial aggregations are computed by the middle layer, delaying computing work for the main server. Related and geographically close client devices can be affected to the same edge server in order to respectively 1. reduce non IID-effects and/or data heterogeneity on the global model; 2. improve communication efficiency.

This architecture also leads to better scalability of the system at the cost of more communication because the network architecture is deeper. It also introduces a semi-Decentralized aspect.

Message compressor

Adds a module on the clients and server in order to compress the model parameters during the model exchange rounds. Even thought it decreases the communication cost, it adds computation work for each client, but especially for the server that will have to deal with decompression of bulky data sent by the clients. We can imagine pairing this pattern with the asynchronous aggregator in order to share the decompression tasks over the time.

This pattern looks useful only in contexts where the balance between communication and computing cost is heavily unfair. For instance, in spatial missions - and to my knowledge - on Cluster II mission [19] some preprocessing of spatial data is performed on board of spaceships in order to reduce the size of data to be send, because communication from space to earth is very slow.

Co-Versioning registry

Keeps records of every local model versions in order to take decisions based on the evolution of the local models. This makes it easier to detect adversarial or dishonest clients, but also to ana-

lyze the impact of the local models on the global one. It provides information that are able to be used to decision making as in the **Client selector** but with no communication price added as the local models have to be sent to the server in any ways. The only downside concerns the extra storage cost needed to keep track of the local models.

The **constraints** and **results** of each respective patterns presented above are resumed in the Annex 2.

3.3. DFL architectures

Context: DFL architectures are characterized by their absence of central server coordinating the whole process. Each nodes collaborate together in an autonomous way in order to train a common model for the network. Roles endorsed by the nodes can be separated into 4 categories: trainer, aggregator, proxy and idle. A trainer trains a local model on its local data and transmits the model to an aggregator. An aggregator aggregate the local models it received and forwards the new global models parameters to the trainers. A proxy's job is to create a connection between trainer(s) and aggregator(s). An idle node is not doing anything.

Problem: in DFL the main problem is linked to the choice of network topology, the Figure 3 illustrates common examples. Choices concerning the number of connection between nodes and the shape of the network in itself impact heavily the robustness and the frugality of the system.

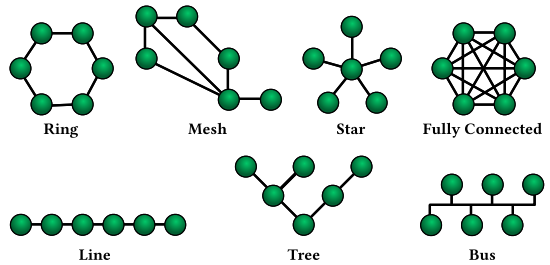


Figure 3: Diagram of different network topologies, Wikipedia.

Solutions:

The following topologies have been consulted in the state-of-the-art article [20], specially focused on DFL. Like in the same section, we will focus on the aspects of robustness and frugality of each of these topologies.

Fully connected

As it's name indicate, this topology makes that every nodes are connected together. Thus each node owns N connection for N number of total nodes in the system. This topology obviously comes with severe scaling issues, as the total number of connection grows dramatically with each additional node. It also makes the task of managing connection harder, resulting in low flexibility. The only advantage of this method is its robustness: as long as the number of nodes stay reasonable (<100), the failure of one node has a insignificant impact on the whole network. So this topology probably provides the best robustness but to a very high price.

Partially connected

With partially connected topologies, the complexity of the network is reduced because every node does not share a direct link with the others. While this makes the network more flexible and scalable, this also means that the communication cost is higher, because nodes have dial through different layers to be able to communicate with some nodes. This topology can be separated into 3 subgroups:

Star topology, which we saw in the previous section, but this time in a decentralized setup the node acting as the link plays a proxy role. We could even elect the node to act as the proxy to make it change over time. However as we saw, this method is also clearly limited in term of scalability while also providing minor robustness with the central node representing a single-point-of-failure. It is rarely used in DFL systems, or as a backup / temporary topology.

Ring networks connects nodes in a circular way, each node having two links for neighbor nodes. Like the *star* topology the number of connections grows linearly, however there is no more single-point-of-failure, but more like parts of the ring that should be repaired if a link of the ring crashes. The communication cost can be optimized by cleverly dispatching trainer and aggregator nodes in the ring. A bad configuration

implies terrible communications cost if one node has to dial another at the opposite side of the ring. This network provides better flexibility while also permitting robustness when adding redundancy to neighbours.

Random networks are implied by heuristics algorithm based on devices proximity or computational capacities of the neighbors. This leads to a dynamic network adapting on real world constraints such as limited connection, availability and proximity. This type of topology offers a high flexibility and moderate fault-tolerance, nevertheless the communication costs and security levels can also be considered as random.

Node clustering

Node clustering techniques are emerging in the field of DFL as it is able to create hierarchical clusters adapting to the distribution of nodes. One could chose to create clusters based on the similarity of the nodes' data using similarity-based clusters. Another option would be to use proxy-based clusters, which make usage of the previous topologies discussed before and links them together with proxy nodes. In the former case, this could speed up the learning speed, but we should pay attention to the robustness of the model and it's generalisation potential. The last case is clearly more situational and would fit in specific scenario.

The **constraints** and **results** of each respective patterns presented above are resumed in the Annex 3.

4. Experiments

This section defines experiments I will do in order to compare the many different architectures possibilities.

Firstly, the dataset used for training the FL model can either be generated in a real use case scenario or taken from an existing dataset. The advantage of making our own dataset is that we can record data with multiple devices, making the data naturally split which suits the real world scenario. If we use an existing dataset the risk is that splitting the data to multiple devices might result in non-realistic data distribution.

Secondly, we would evaluate the performance of each architectures by training the same task on each of them with the same dataset. To achieve this, the architectures patterns will have to be implemented, however the ML algorithms will stay as simple as possible. For example, the principle of FedAvg algorithm could be used (weighted arithmetic mean on the model parameters). These benchmarks would be useful to compare the frugality of each architectures

Thirdly, more benchmarks would be made, this time by adding artificial network instabilities: random node crashing, slow network or congestion, adversarial nodes… These events should obviously be distributed equally during each benchmarks to keep it fair for every architectures. These benchmarks would be used to determine the effect of failures in each architectures in order to estimate and compare their fault-tolerance.

Finally, we could analyze the results from both benchmarks in order to find the best architectures for robustness and frugality. Concluding with the architectures that had the better trade-offs in both of them.

5. Future work

The state-of-the-art of the art of the article showed that multiple studies started to work on constraints related to FL architecture and Carbon footprint. Future work would be to try combining those two aspects in order to build robust and frugal systems.

Open questions are still open concerning the Carbon footprint of FL. For now the estimations are sufficient, but further research could be done to obtain more accurate results. It could be achieved by taking into account the multiple parameters that have not been taken into account (hardware manufacturing, secondary components consumption…).

In addition, the comparison experiments between CFL and DFL is still not explored in terms of frugality and robustness.

Finally, more architectures and topologies should be explored in the field of Distributed Systems because it could give ideas or good starting

points. As an example, Kademlia DHT [21] offers an improved version of the ring topology with great fault tolerance and performance.

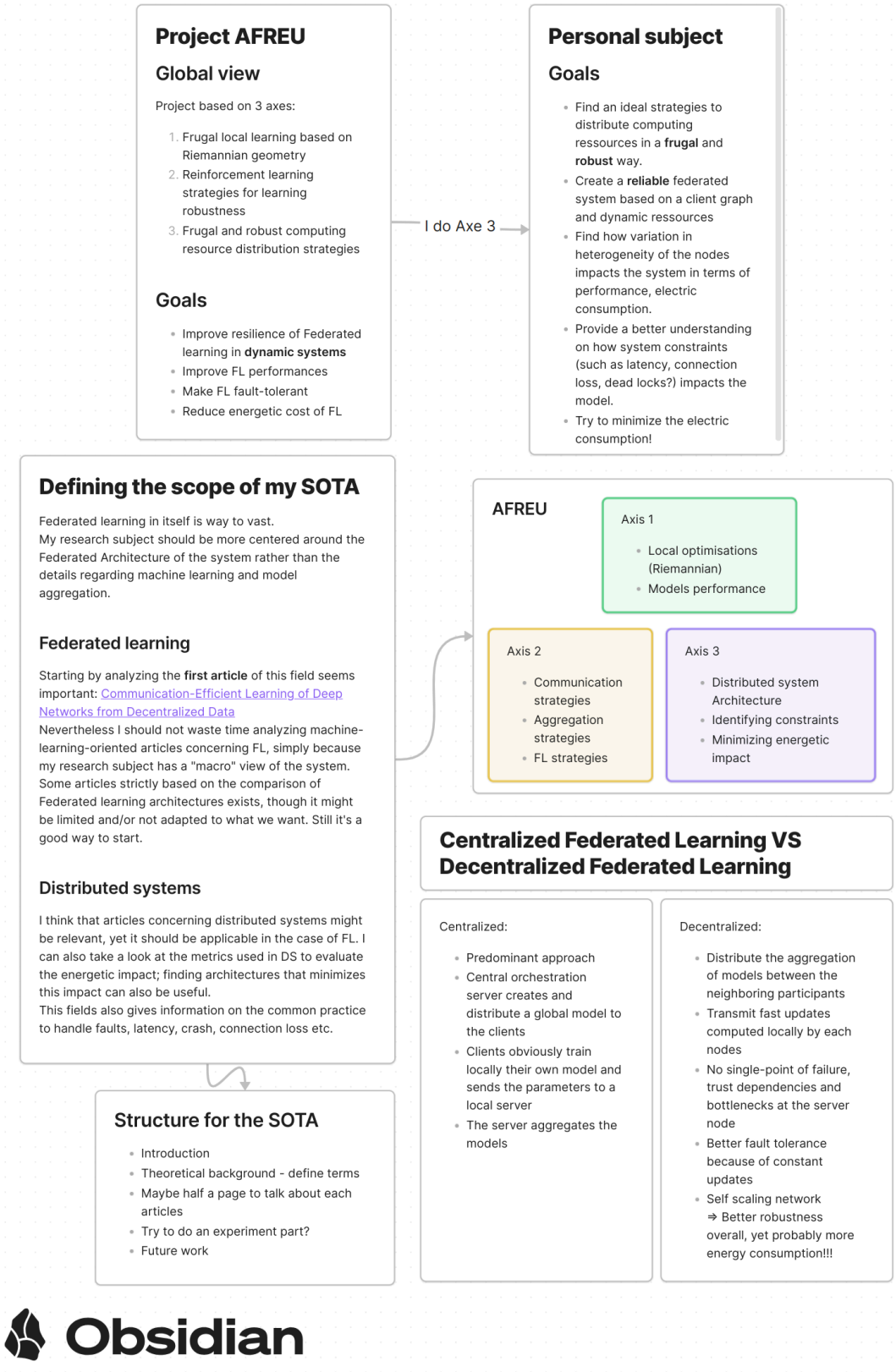
6. Conclusion

To conclude we have showed that Carbon footprint estimation techniques were harder to apply in FL scenario than in classical ML. We identified those constraints and presented methods to estimate Carbon footprint in state-of-the-art articles. Furthermore, the key factors of Carbon emissions have been identified. This helped us to keep in mind the different criteria that would directly or indirectly impact the frugality of the system.

Different available patterns for CFL have been compared. We have seen that each choice of design can bring advantages but almost always bring disadvantages. The architecture decisions must be taken by defining robustness and frugality goals in order to make trade-offs between both.

The topologies used in the state-of-the-art of DFL have also been compared. Some classic networks topologies hardly adapt to the task, while other bring more hope.

7. Appendices



Annex 1: Mind map used during the conduct of the study. Made with Obsidian.

Pattern	Robustness	frugality
Client registry	+ Easy maintainability + Reliable as it provides good node tracking	- Maintaining client device info requires more communication and storage
Client selector	+ Client selection decrease node crashing probabilities + Better model performance	+ Resource optimization by selecting best models - minor extra computing and communication cost for decision making
Client cluster	+ Better model quality for highly personalised tasks + Higher convergence speed	+ Slightly less communication due to convergence speed - Computation cost for client clustering and relationship quantifying
Asynchronous aggregator	+ Less bottleneck issues on the server - Model bias when client training speed is too disproportional	- More communication cost due to more training iterations
Secure aggregator	+ System security	- Potentially large computation cost
Hierarchical Aggregator	+ Speed up model aggregation by sharing the work + Better scalability + Handling heterogeneous data	+ Edge devices closer to clients devices - Security breach on edge servers
Message compressor	- Possible loss of information	+ Lower communication cost in all setups - More computation cost, especially on the server
Co-Versioning registry	+ Detection of adversarial or dishonest clients + Improving model quality by tracking each model contributions by epochs	- Extra storage cost to store each local model versions

Annex 2: Resume of CFL patterns

Topology	Robustness	frugality
Fully connected	+ Extremely robust and node crashing proof - Extremely bad scalability	- Big maintenance cost for the connections - High communication cost and potential flooding of the network
Partially connected: *Star*	- Bad scalability - Bad robustness with single-point-of-failure	+ Linear communication growth
Partially Connected: *Ring*	+ Balance between good robustness and low network complexity + Good flexibility	+ Low communication costs if used properly - High communication costs in worst case scenario
Partially Connected: *Random*	+ High flexibility - Hard to implement security on	- Random or too varying communication costs
Node clustering	+ High flexibility and customization for specific tasks + Can handle data heterogeneity by design	- Varying communication costs, but it can be optimized by algorithms

Annex 3: Resume of DFL topologies

Bibliography

[1] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data", in *Artificial intelligence and statistics*, 2017, pp. 1273–1282.

[2] O. Rana *et al.*, "Hierarchical and Decentralised Federated Learning", *arXiv preprint arXiv: 2304.14982*, 2023.

[3] J. Rockström *et al.*, "Planetary Boundaries: Exploring the Safe Operating Space for Humanity", *Ecology and Society*, vol. 14, no. 2, 2009, Accessed: Jan. 07, 2024. [Online]. Available: http://www.jstor.org/stable/26268316

[4] K. Richardson *et al.*, "Earth beyond six of nine planetary boundaries", *Science Advances*, vol. 9, no. 37, p. eadh2458, 2023, doi: 10.1126/sciadv.adh2458.

[5] N. Jones, "How to stop data centres from gobbling up the world's electricity". 2018.

[6] D. H. Dario Amodei, "AI and compute". 2018.

[7] E. Strubell, A. Ganesh, and A. McCallum, "Energy and Policy Considerations for Deep Learning in NLP", *CoRR*, 2019, [Online]. Available: http://arxiv.org/abs/1906.02243

[8] R. Gosselin, L. Vieu, F. Loukil, and A. Benoit, "Privacy and security in federated learning: A survey", *Applied Sciences*, vol. 12, no. 19, p. 9901, 2022.

[9] M. Nasr, R. Shokri, and A. Houmansadr, "Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning", in *2019 IEEE symposium on security and privacy (SP)*, 2019, pp. 739–753.

[10] H. Zhu, J. Xu, S. Liu, and Y. Jin, "Federated learning on non-IID data: A survey", *Neurocomputing*, vol. 465, pp. 371–390, 2021.

[11] Q. Wu, C. Wang, and S. Huang, "Frugal optimization for cost-related hyperparameters", in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021, pp. 10347–10354.

[12] Y. Gong, Y. Li, and N. M. Freris, "FedADMM: A robust federated deep learning framework with adaptivity to system heterogeneity", in *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, 2022, pp. 2575–2587.

[13] J. C. Jiang, B. Kantarci, S. Oktug, and T. Soyata, "Federated learning in smart city sensing: Challenges and opportunities", *Sensors*, vol. 20, no. 21, p. 6230, 2020.

[14] X. Qiu *et al.*, "A first look into the carbon footprint of federated learning", *Journal of Machine Learning Research*, vol. 24, no. 129, pp. 1–23, 2023.

[15] M. Hodak, M. Gorkovenko, and A. Dholakia, "Towards power efficiency in deep learning on data center hardware", in *2019 IEEE International Conference on Big Data (Big Data)*, 2019, pp. 1814–1820.

[16] S. Savazzi, V. Rampa, S. Kianoush, and M. Bennis, "An energy and carbon footprint analysis of distributed and federated learning", *IEEE Transactions on Green Communications and Networking*, vol. 7, no. 1, pp. 248–264, 2022.

[17] A. Capozzoli and G. Primiceri, "Cooling systems in data centers: state of art and emerging technologies", *Energy Procedia*, vol. 83, pp. 484–493, 2015.

[18] S. K. Lo, Q. Lu, L. Zhu, H.-Y. Paik, X. Xu, and C. Wang, "Architectural patterns for the design of federated learning systems", *Journal of Systems and Software*, vol. 191, p. 111357, 2022.

[19] C. P. Escoubet, M. Fehringer, and M. Goldstein, "Introduction The Cluster mission", *Annales Geophysicae*, vol. 19, no. 10/12, pp. 1197–1200, 2001, doi: 10.5194/angeo-19-1197-2001.

[20] E. T. M. Beltrán *et al.*, "Decentralized federated learning: Fundamentals, state of the art, frameworks, trends, and challenges", *IEEE Communications Surveys & Tutorials*, 2023.

[21] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the xor metric", in *International Workshop on Peer-to-Peer Systems*, 2002, pp. 53–65.