Towards Developing a Global Federated Learning Plateform for IoT

Hamza SAFRI
Berger-levrault
Toulouse, FRANCE
hamza.safri@berger-levrault.com

Mohamed Mehdi KANDI
Berger-levrault
Toulouse, FRANCE
mohamed.kandi@berger-levrault.com

Youssef MILOUDI
CARL - Berger-levrault
Toulouse, FRANCE
youssef.miloudi@berger-levrault.com

Christophe BORTOLASO
Berger-levrault
Toulouse, FRANCE
christophe.bortolaso@berger-levrault.com

Denis TRYSTRAM
INRIA-LIG
University Grenoble Alpes
Grenoble, FRANCE
denis.trystram@inria.fr

Frédéric DESPREZ
INRIA
Grenoble, FRANCE
frederic.desprez@inria.fr

***Abstract*—Federated learning (FL) is an approach that enables collaborative machine learning (ML) without sharing data over the network. Internet of Things (IoT) and Industry 4.0 are promising areas for FL adoption. Nevertheless, there are several challenges to overcome before the deployment of FL methods in existing large-scale IoT environments. In this paper, we present one step further towards the adoption of FL systems for IoT. More specifically, we developed a prototype that enables distributed ML model deployment, federated task orchestration, and monitoring of system state and model performance. We tested the prototype on a network that contains multiple Raspberry Pi for a use case of modeling the states of conveyors in an airport.**

***Index Terms*—Internet of Things, Federated Learning, FL Task Orchestration, Deployed Model Monitoring.**

I. Introduction

Federated Learning (FL) is a promising approach for building machine learning models in a distributed way. This approach allows multiple nodes, with different collected data and resources, to participate in training a collaborative model. Each node contributes to the global learning process by independently computing the parameters of a model based on its local data. Federated learning approaches have many advantages: **(1) Reduce training time:** multiple devices train a model in parallel before aggregation, **(2) Reduce inference time:** predictions are made locally in the node level, so we avoid making requests to the cloud, **(3) Privacy:** Local data processing preserves sensitive data without the need for centralized cloud storage of that data, **(4) Collaborative learning:** learning federation enables a kind of crowdsourcing that eases the process of collecting and labeling data, in terms of time and effort, instead of having to collect a massive data set to train a machine learning model.

The FL approach has been used and evaluated in many contexts of distributed systems. It was first proposed by Google for a mobile devices use case [1]. Then, it was adopted by the industrial and scientific communities. In the literature, we find several FL architectures [2] that can be categorized based on the application (Health [3], Industry 4.0 [4], IoT Application [5]...) or the data repatriation (Vertical or Horizontal).

IoT and Industry 4.0 are among the more promising areas for federated learning adoption. However, the implementation of FL faces several issues in edge and IoT environments. These issues are mainly related to the adaptation to existing industrial IoT architectures. These architectures have an important impact on the large-scale deployment of FL ecosystems due to the dependency on several important aspects such as edge/cloud server and edge-IoT communication protocols.

It becomes important to develop a platform that allows the orchestration of federated tasks in this distributed and collaborative learning. The platform must take into consideration the resource constraints of the different participants. It must also consider the existing communication protocols. The model deployment mechanisms must allow the platform to be operational in several usage applications (different types of data, ML model types, and the number of nodes). It must also provide a Dashboard to monitor the state of the platform and show Insights related to the resource consumption and the current model efficiency. In this paper, we present a prototype that we have developed within our organization. This prototype is a step towards the development of an industrial IoT-FL platform and its deployment in large-scale IoT environments.

In the rest of this paper we first present, in section II, the architecture of the system that we developed and details about the main features. Then, we give in section III an overview of the prototype functional validation on a predictive maintenance problem. Finally, we conclude the paper in section IV.

II. System Architecture

FL is based on local training as close as possible to the devices and then an aggregation in the server. To achieve this goal, we propose the learning architecture presented in Fig 1. The architecture (named FedIoT) is composed of the three following sub-layers: **(1) Data Generators (DG)** : It corresponds to a sub-layer of the *devices layer*, which contains

the connected equipments that collect the data. **(2) Local Orchestrator (LO)**: is the second sub-layer of the *devices layer* that contains the local orchestrators (LO) deployed on the IoT nodes (IoT server or gateway) **(3) Global Orchestrator (GO)**: is the main component of the architecture. It has two roles: first, it manages the configuration and the dynamic deployment of the local orchestrators. Second, it synchronizes the training and the aggregation of the outputs of each local orchestrator. The architecture is composed of two networks as

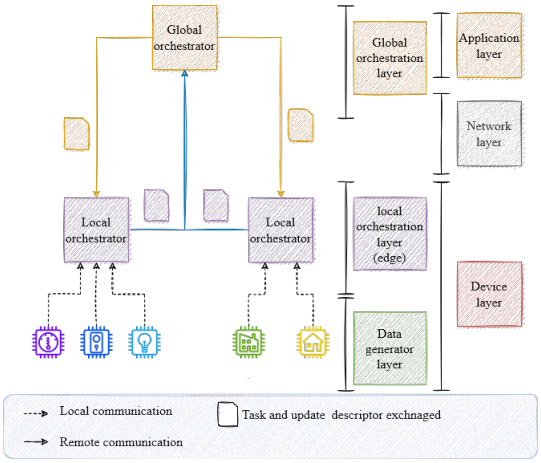

Fig. 1. FedIoT : FL architecture adaptation for IoT

illustrated in Fig 1. A sensor network links the data generating sub-layer and the local orchestrator sub-layer. Its role is the local transfer of data. A second unmeasurable network (WIFI, etc) links the local and global orchestrators. This network allows sending of the necessary parameters for the local training process and to distribute the generalized models to the different local orchestrators.

1) Global Orchestrator (GO): The global orchestrator is the cornerstone of architecture. This component provides several functionalities for the management and configuration of the different participants to orchestrate the collaborative tasks and manage the execution environment. For this we can group the features described in Fig 2 in three groups:

- **Local orchestrator management features:** the OG allows to manage of local orchestrators using an API exposed by the components. this feature enables the deployment, modification of the configuration, or even the deletion of an OL in a host machine
- **Model features :** this feature allows sending the different configurations needed for the training process including the data preparation methods and the model evaluation in addition to the configuration of the local model to be trained.
- **Monitoring features:** Monitoring consists in collecting data on the execution environment of each OL, mainly resources in different phases (data preparation, training, and normal state) in addition to model evaluation parameters. The framework also provides information on average CO_2 per LO and trained model.

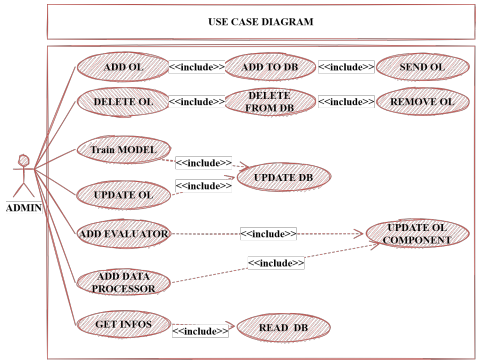

Fig. 2. use case diagram that summarizes the main features of global orchestrators for participant management and training orchestration

These features are provided by several blocks as shown in Fig 3, mainly:

- Aggregation models: in this block we can implement an aggregation logic that will be used by GO to generalize the different local models.
- Training Orchestrator: this block is responsible for triggering training in LOs and ensuring the synchronization of the participants' responses.
- Local Orchestrator API: is a very basic version of the global orchestrator. This component allows a local orchestration of the training taking into account the preparation of the data generator data and the user-defined custom evaluation. In addition to IoT nodes resources monitoring.
- Installer: is the component responsible for the deployment of LOs at the IoT nodes starting from the installation of the dependencies to the component startup.
- Compressor: is a dedicated block for the compression of files and voluminous data.
- Monitoring agents: very light monitoring agents that can collect and store metrics on resource consumption and CO2
- UI interface: provides a visualization of the data collected by the monitoring agents in addition to the performance of the models
- Communication module: a software connector that initiates communication between the global orchestrator and the IoT node.

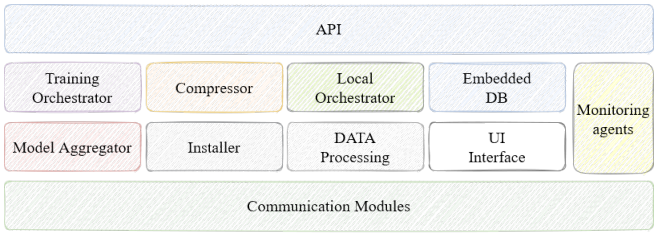

Fig. 3. Global Orchestration architecture: the main components of global orchestrator

2) Communication interfaces: the GO exposes two communication interfaces to interact on the one hand with the external actors and on the other hand with the internal components.

- **Management interface:** The OG exposes an API that allows : (1) the external actors to manage the configurations of the LOs and the necessary configuration of the models, mainly the models and their configuration, data preparation, and the personalized evaluation of the models; (2) the communication with the already deployed LOs.
- **Configuration interface:** This interface allows the OG to manage the deployment and deletion of LOs at the host level. This interface is managed by SSH or MQTT communication.

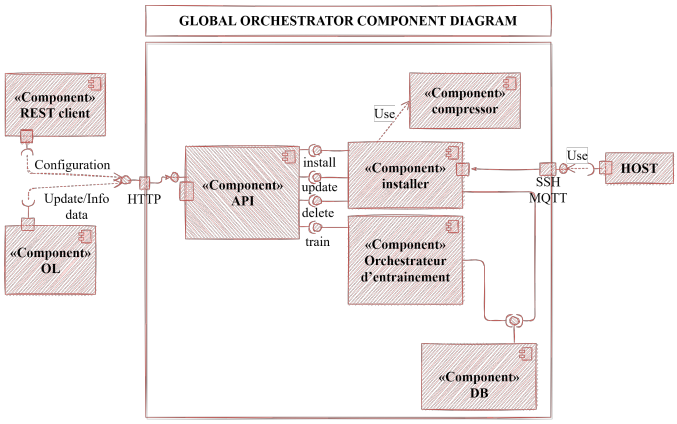

Fig. 4. Composite structure diagram describes the main communication points between the global orchestrator and the external actors

3) Architecture implementation: Fig 5 details the different interactions between the components. The participants are IoT nodes that manage the data of several connected objects. These participants must initiate a communication with the global orchestrator to announce that the node is ready to execute a federated task. In this perspective, the global orchestrator ensures the deployment of a local orchestrator at node IoT to guarantee an orchestration of the federated tasks, mainly the model training. This is the LO Configuration step. It is important to note that each GO manages several LOs. Once the LOs are well configured, a second registration is necessary but this time the DGs concerned by the federated tasks inform the LOs about the location of the collected data. This operation can also be performed via a REST client. This registration can be rejected if it is not correctly described. Each local orchestrator manages several DGs, which means that the LOs orchestrate the training of the local models of each DG. The two previous steps represent the backbone of our architecture. The deployment of these components allows us to launch a training based on the coordination between LOs and GOs which mainly use the API to exchange the necessary configurations for models training and aggregation. These configurations are implemented at the internal component *Training Orchestrator* for the training and the validation and the component *Data Processing* for data preparation of each DG and *evaluator* to describe the evaluation method of local and global models. At the end, the IoT nodes send the output of the models to the server to aggregate and integrate the final model.

III. Demonstration

We are setting up an architecture composed of 6 raspberry Pi 3 Model B and two docker containers that play the role of IoT nodes where we will deploy OL using SSH and MQTT communication, and we are using an HP core i7 computer to deploy the OG. the different components are connected to different sub-networks but can be reached by the OG. The machine learning part uses real-life data collected by the company CARL berger-levrault in the context of predictive maintenance projects. This setup implemented by the technologies described in Table I will allow us to validate the dynamic and on-demand configuration by :

- **Deploying a LO**: To deploy a LO, the GO receives a JSON configuration file from a rest client or MQTT client containing information about the host machine (Ip, username, ssh key or password, and ssh port) and information about the application deployment, such as the port and a unique name for management. This operation allows the GO to connect to the host, install the dependencies (in case of rest request), and start the application.
- **Adding a DG**: each DG must be registered at LO by sending a configuration file that contains the data location and unique identifier.
- **processing and validation scripts :** sending and integrating python scripts for model preparation and evaluation
- **Triggering a training**: this operation is based on a configuration file sent to the LO via API. In this demonstration, we use the Federated stacking algorithm proposed on other contributions. For that, we send a file that describes first the dependencies to instantiate the models and then the models to train, second the parameters of each model, and finally the percentage of data used for the test. Once the file has been sent, the GO orchestrates the training between the different LO. The LO, on the other hand, orchestrates the local training of the different learning algorithms using the data of each DG. Once the model is well trained, we can access the evaluation metrics via the API of each LO.

multirow

TABLE I
THE TECHNOLOGIES USED FOR THE IMPLEMENTATION

Component	Technology
OG	Python - Flask restplus
OL	
Dashboard	Grafana

The dashboard provides the information collected to validate the proper functioning of the mentioned features. The data are retrieved from the APIs (Fig 6) information resource

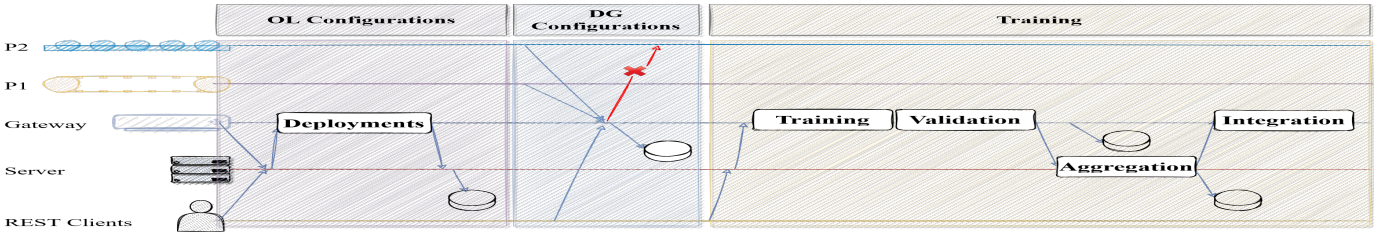

Fig. 5. Communication protocol : Description of different phases of architecture components configuration and training orchestration

Fig 7 shows the web interface of the global orchestrator after deployment of LOs, registration of DGs, and training and evaluation of models. This interface provides an overview of the resource consumption of the OG (CPU, RAM, Battery), the last used model configuration, the registered local orchestrators, and the logs of the last executed queries. For this, Fig 7 shows the same info on the resources of the LO hosts in addition to the CO2 consumption generated by the execution of the models. interfaces also visualize industrial metrics configured by scripts sent by the user for the evaluation of the models

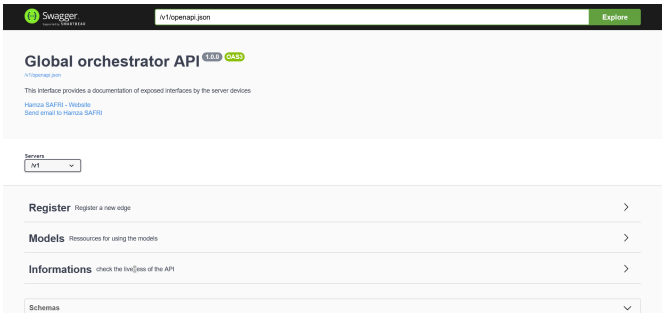

Fig. 6. The OG API Swagger documentation

IV. Conclusion

In this paper, we presented a federated learning prototype for IoT environments. We proposed and implemented an extension of the basic IoT architecture with two levels of orchestration. The first level is an orchestration for participant management and local model training. The second level is an orchestration for global management of local ML model training and aggregation... Despite the positive results presented in this paper, we are still working on new features mainly : (1) Integration and adaptation of the solution to other aggregation methods such as FEDAVG [1], (2) Integration of participant selection methods to minimize training time and resource consumption based on monitoring data.

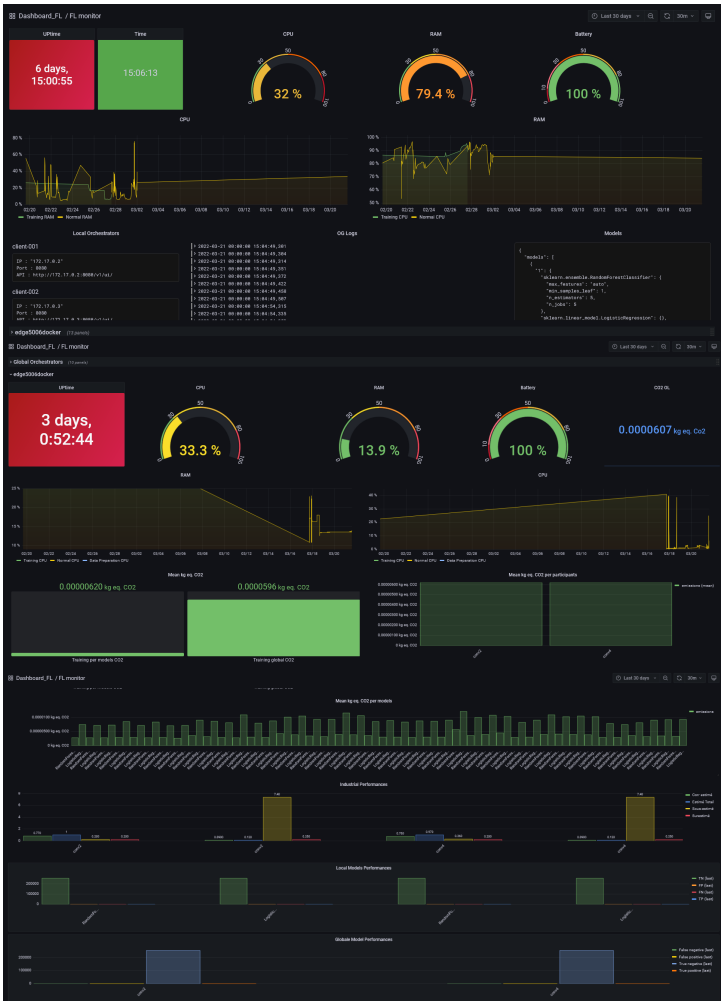

Fig. 7. Web interface visualizes the data collected at the OG level and OL level retrieved via these APIs

References

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

[2] M. Aledhari, R. Razzak, R. M. Parizi, and F. Saeed, "Federated learning: A survey on enabling technologies, protocols, and applications," *IEEE Access*, vol. 8, pp. 140 699–140 725, 2020.

[3] B. Yuan, S. Ge, and W. Xing, "A Federated Learning Framework for Healthcare IoT devices," 2020.

[4] M. Hao, H. Li, X. Luo, G. Xu, H. Yang, and S. Liu, "Efficient and privacy-enhanced federated learning for industrial artificial intelligence," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 10, pp. 6532–6542, 2019.

[5] J. Ren, H. Wang, T. Hou, S. Zheng, and C. Tang, "Federated learning-based computation offloading optimization in edge computing-supported internet of things," *IEEE Access*, vol. 7, pp. 69 194–69 201, 2019.