Cloud Native Federated Learning for Streaming: An Experimental Demonstrator

Sergio Barrachina-Muñoz, Engin Zeydan, Luis Blanco, Luca Vettori, Farhad Rezazadeh, Josep Mangues-Bafalluy

Centre Technologic de Telecomunicacions de Catalunya (CTTC), Castelldefels, Barcelona, Spain, 08860.
Emails: {sbarrachina, ezeydan, luis.blanco, lvettori, frezazadeh, jmangues}@cttc.es

***Abstract*—This paper demonstrates an implementation of Federated Learning (FL) for streaming applications using cloud-native technology. Compared to a centralized management, by adopting a decentralized approach, the FL method improves convergence time, reduces communication overhead, and increases network energy efficiency. The cloud-native FL architecture presented comprises three sites, each with its own Kubernetes (K8s) cluster. The edge sites run FL Analytical Engines (AEs)/clients for local training and updates, and the central site runs the aggregation server for FL training. Some other relevant workloads deployed at the clusters are the video streaming server, the orchestrator, and monitoring components. As for the RAN, we showcase a multi-gNB setup from which we obtain monitoring data via custom sampling functions. Following the description of the testbed infrastructure and setup, this demonstration presents the real-time visualization of network parameters during FL training, and the enhancement of video streaming through proactive Central Processing Unit (CPU) scaling, made possible by the resource forecasting.**

***Index Terms*—video streaming, cloud native, federated learning, distributed, network management, experiments**

I. Introduction

Cloud native distributed Artificial Intelligence (AI) algorithm design has emerged as a promising approach for collaborative Machine Learning (ML) in a distributed environment [1]. By allowing multiple parties to collaborate and train ML models on decentralized data sources, Federated Learning (FL) has the potential to enable scalability, low data exchange, and enhanced security in a wide range of new applications and use cases [2]. For managing a massive number of network slices across different technological domains with zero-touch, a scalable, hierarchical, and distributed approach that utilizes AI technology is required. The MonB5G project [3] funded by the EU H2020 proposes a technical approach that distributes management functions among all entities responsible for the lifecycle management of network slices and utilizes distributed closed control loops to assist the lifecycle management entities with state-of-the-art data-driven and AI-based mechanisms. This approach consists of four components: Monitoring System (MS), Analytical Engine (AE), Decision Engine (DE) and Actuator (ACT) [4]. In this paper, we present a novel experimental demonstrator for cloud-native FL for video streaming data relying on MonB5G framework. Our approach leverages the latest advances in cloud-native computing, including Kubernetes, and 5G communication to enable efficient and scalable FL on streaming video traffic data.

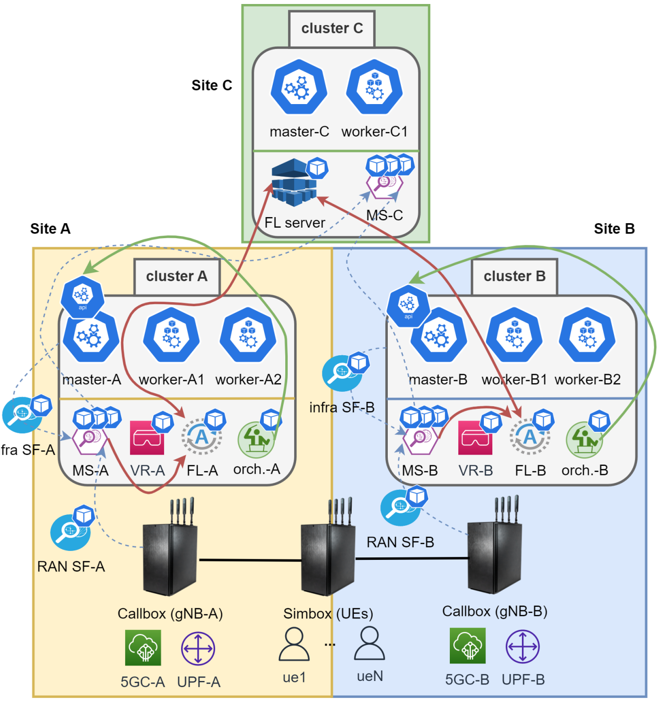


Figure 1: PoC infrastructure and setup.

II. Testbed Infrastructure and Dataset

Fig. 1 shows the Proof of Concept (PoC) infrastructure setup. The testbed instance we will be using in this scenario is designed to evaluate MonB5G solutions in a 5G network environment. It is comprised of a variety of different components that work together to enable the testing and evaluation of FL approach. In particular, Fig. 1 consists of a MS, an AI-enabled resource predictor function called AE, DE and an ACT. Each MS performs slice-level RAN Key Performance Indicator (KPI) data collection to build its local datasets for each slice. A server located at site-C, plays the role of FL model aggregator. Each AE predicts the required amount of resources to ensure low Normalized Mean Squared Error (NMSE) when making predictions. The components can be categorized as follows:

This work has been partially funded by EC H2020 MonB5G project (Grant 871780), Spanish MINECO - Program UNICO I+D (grants TSI-063000-2021-54 and -55) and Grant PID2021-126431OB-I00 funded by MCIN/AEI/ 10.13039/501100011033 and by "ERDF A way of making Europe" .

Cloud-native infrastructure: of the testbed is built using containers and virtual machines. In essence, Virtual Machines (VMs) are used as (worker/master) nodes to deploy Kubernetes (K8s) clusters, while containers run different applications and microservices. This allows for flexibility and scalability in the testbed, as well as the ability to easily replicate and test different configurations. In particular, we consider three K8s clusters: two for the edge/RAN domains and one in the cloud for showcasing the hierarchical features of the monitoring system.

Monitoring System: In order to monitor and troubleshoot the testbed, we deploy three instances of the MonB5G MS. The FL-related MS instances running at the edge sites will run customized sampling functions (for infrastructure monitoring and for Radio Access Network (RAN) monitoring). These monitoring systems will be able to monitor and collect metrics from the testbed infrastructure, as well as from the RAN. We also use Prometheus to expose some metrics of interest, which can then be visually displayed in Grafana for easy analysis. The cloud MS will run a generic Sampling Function (SF) to showcase extracted values from the edge sites.

Orchestration: The testbed uses a Python script based on the K8s scheduler to act as the intermediary between the FL agents and the K8s scheduler. The script triggers actions related to pod scale-out, in order to ensure pre-emptive and efficient resource allocation via the FL process.

Federated learning: The testbed includes a FL client/AE that builds local models in phase I and makes inferences in phase II. The FL client connects to the MS in order to collect and analyse metrics during the learning process. The FL aggregation server is responsible for exchanging weights between FL clients in order to aggregate the learned models.

5G network: The testbed includes a User Equipment (UE) emulator realized with Amarisoft Simbox[1] and 2x gNB realized with Amarisoft Callbox[2]. These components are plugged together in order to create a rich multi-domain setup for testing FL algorithms. There are different alternatives for the 5G core, including Amarisoft's proprietary core or open-source projects like Open5GS, Free5G, or OpenAirInterface Core Network. Each of these open-source cores have been integrated with Amarisoft Callbox gNB, so we have the flexibility to use any of them.

Video Streaming Application: The testbed includes a video on demand (VoD) client and server as shown in Fig. 3 (which shows the average Central Processing Unit (CPU) versus number of active UEs (N) consuming the video stream in FL sites 1 and 2), both of which are cloud-native. High bandwidth video is used to stress the CPU of the server and trigger actions, such as scaling out pods, in order to test the performance of the FL algorithm under different conditions.

Table I summarizes the input features and the supervised output of the local dataset. Here, the CPU load resources are dynamically allocated at the video streaming server level according to the traffic patterns as given in Fig. 2 (previous and current CPU) and radio conditions (average bit rate) of each slice. The model used in FL phase is a custom neural network model with one hidden layer. The final FL model is used to predict the CPU load of the clients.

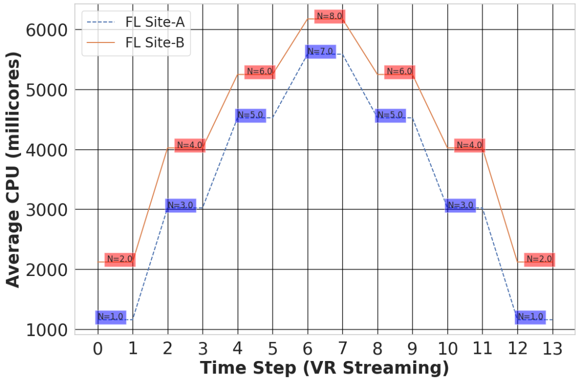


Figure 2: Traffic Patterns generated at different FL sites.

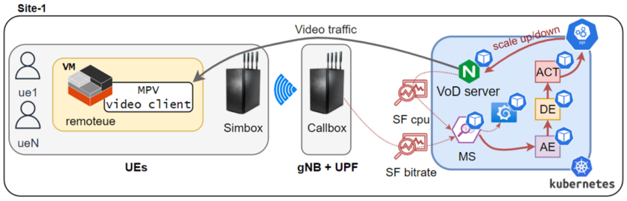


Figure 3: Video Streaming application infrastructure and setup.

Table I: Dataset Features and Output

Feature	Description
Current Bitrate	The instantaneous bit rate at gNodeB when video streaming is running
Previous CPU Load	Previous Time Window CPU resource consumption for video streaming server
Current CPU Load	Current Time Window CPU resource consumption for video streaming server

Output	Description
Next CPU Load	Next Time Window CPU resource consumption for video streaming server

III. Demonstration of the Proposed Approach

In cloud native implementation, we use Docker containers for containerization of MS, AE and use K8s as overall orchestrator. In docker implementation, AEs at different sites simultaneously run on K8s. Through REST API, the FL server and two AEs (clients) can communicate with each other. FastAPI[3] as a REST API is used in our implementation.

The flow of information is shown in Fig. 4 and can be summarized as follows: In the first step, all clients should know the IP address of server node to register themselves with the FL server (or aggregation) node (which is always in running mode). After registration, the administrator starts the automated

[1] Online: https://www.amarisoft.com/products/test-measurements/amari-ue-simbox/, Available: April 2023.

[2] Online: https://www.amarisoft.com/products/test-measurements/amari-lte-callbox/, Available: April 2023.

[3] Online: https://fastapi.tiangolo.com/, Available: 10-2022

FL process in step-2. Then, the server sends training requests to clients and start FL training in step-3. Local training is performed for each client in step-4. The model weights of each clients are sent to the server in step-5. Then, the server computes the average of the model weights in step-6. The overall system parameters are updated in step-7 and the same procedure is repeated for upcoming FL rounds starting from step-3 in Fig. 4.

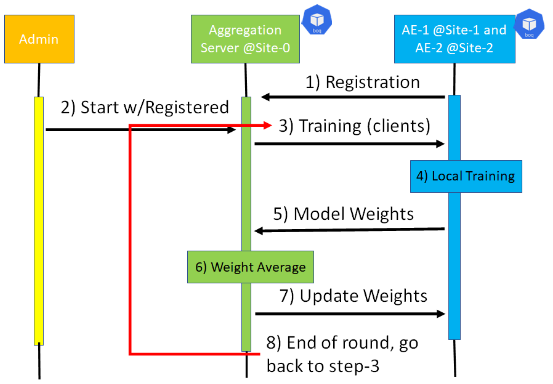


Figure 4: High level workflow of the FL approach.

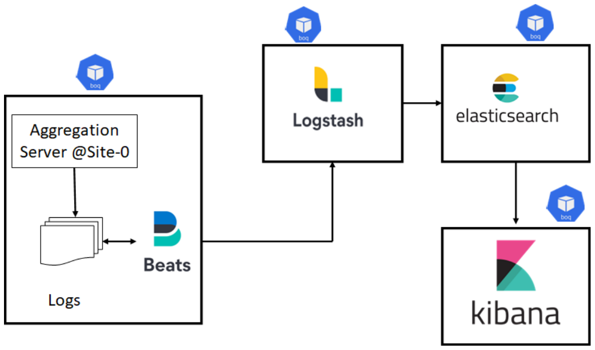
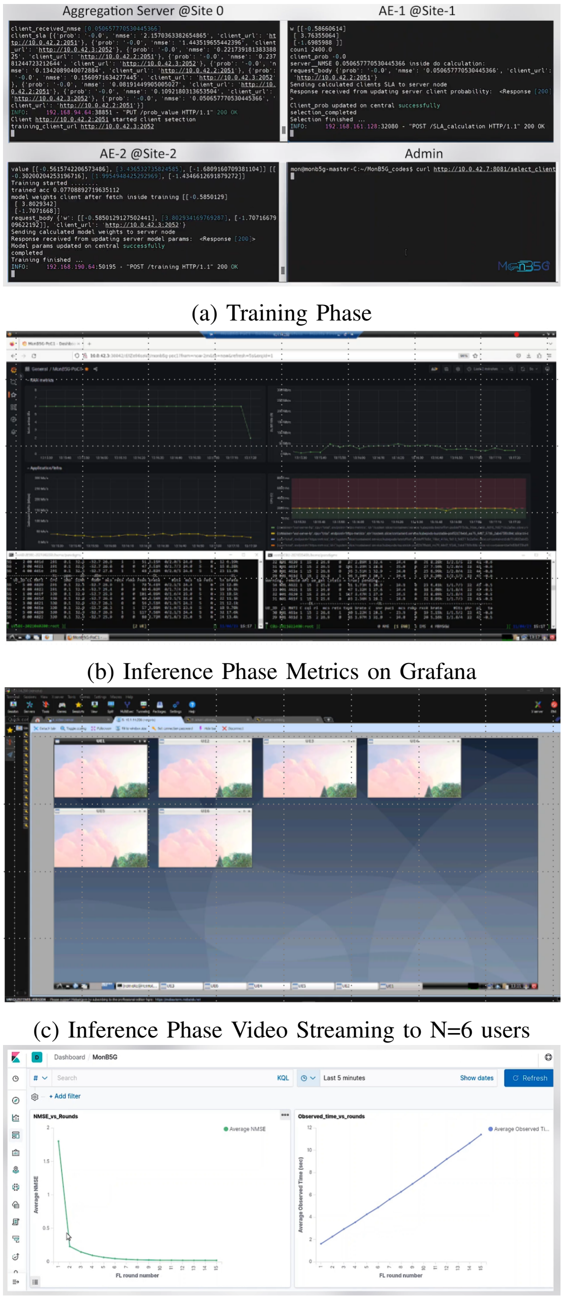


Figure 5: Containerization of real-time visualization platform to keep track of relevant metrics from aggregation server.

For real-time visualization of changes in NMSE over the FL rounds, we used the ELK stack[4] (Logstash as log transformer, Elasticsearch as the data indexer and Kibana for visualization) and Filebeat side container to track the metrics status from the aggregation server as given in Fig. 5. Fig.6 shows all considered metrics as well as the dashboards of the demo visualization. Fig.6 (a) shows the training phase of the considered FL approach. Fig.6 (b) shows the inference phase and the metrics that are observed on the Grafana dashboard [5]. Fig.6 (c) shows the Finally, Fig.6 (d) shows the NMSE values (on left) and observations time over the number of rounds (on right) using Kibana dashboard.

REFERENCES

[1] A. Boudi, M. Bagaa, P. Pöyhönen, T. Taleb, and H. Flinck, "Ai-based resource management in beyond 5g cloud native environment," *IEEE Network*, vol. 35, no. 2, pp. 128–135, 2021.



Aggregation Server @Site 0

```
client_received_nmse [0.050657770530445366]
client_sla [{'prob': '-0.0', 'nmse': '2.1570363382654865', 'client_url': 'ht
tp://10.0.42.2:2051'}, {'prob': '-0.0', 'nmse': '1.443519655442396', 'client
_url': 'http://10.0.42.3:2052'}, {'prob': '-0.0', 'nmse': '0.22173918138338825', 'client_url': 'http://10.0.42.2:2051'}, {'prob': '-0.0', 'nmse': '0.237
81244723212644', 'client_url': 'http://10.0.42.3:2052'}, {'prob': '-0.0', 'n
mse': '0.1342089040072884', 'client_url': 'http://10.0.42.2:2051'}, {'prob':
 '-0.0', 'nmse': '0.1560971634277445', 'client_url': 'http://10.0.42.3:2052'
}, {'prob': '-0.0', 'nmse': '0.08191449905005027', 'client_url': 'http://10.
0.42.2:2051'}, {'prob': '-0.0', 'nmse': '0.1092180313653504', 'client_url':
'http://10.0.42.3:2052'}, {'prob': '-0.0', 'nmse': '0.050657770530445366', '
client_url': 'http://10.0.42.2:2051'}]
INFO:     192.168.94.64:38851 - "PUT /prob_value HTTP/1.1" 200 OK
Client http://10.0.42.2:2051 started client setection
training_client_url http://10.0.42.3:2052
```

AE-1 @Site-1

```
w [[-0.58660614]
 [ 3.76355064]
 [-1.6985988 ]]
coun1 2400.0
client_prob -0.0
server__NMSE 0.050657770530445366 inside do calculation:
request_body {'prob': '-0.0', 'nmse': '0.050657770530445366', 'client_url':
 'http://10.0.42.2:2051'}
Sending calculated clients SLA to server node
Response received from updating server client probability:  <Response [200]
>
Client_prob updated on central successfully
selection_completed
Selection finished ...
INFO:     192.168.161.128:32080 - "POST /SLA_calculation HTTP/1.1" 200 OK
```

AE-2 @Site-2

```
value [[-0.5615742206573486], [3.436532735824585], [-1.6809160709381104]] [[
-0.3020020425319671 6], [1.9954948425292969], [-1.4346612691879272]]
Training started ........
trained acc 0.07708892719635112
model weights client after fetch inside training [[-0.5850129]
 [ 3.8029342]
 [-1.7071668]]
request_body {'w': [[-0.5850129127502441], [3.802934169769287], [-1.70716679
09622192]], 'client_url': 'http://10.0.42.3:2052'}
Sending calculated model weights to server node
Response received from updating server model params:  <Response [200]>
Model params updated on central successfully
completed
Training finished ...
INFO:     192.168.190.64:50195 - "POST /training HTTP/1.1" 200 OK
```

Admin

```
mon@monb5g-master-C:~/MonB5G_codes$ curl http://10.0.42.7:8081/select_client
```

(a) Training Phase

(b) Inference Phase Metrics on Grafana

(c) Inference Phase Video Streaming to N=6 users

(d) Kibana Dashboard for Training phase

Figure 6: Cloud native FL showcasing training phase, inference phase and visualizations in Kibana and Grafana.

[2] Z. Yang, M. Chen, K.-K. Wong, H. V. Poor, and S. Cui, "Federated learning for 6g: Applications, challenges, and opportunities," *Engineering*, vol. 8, pp. 33–41, 2022.

[3] S. Kukliński *et al.*, "Ai-driven predictive and scalable management and orchestration of network slices," *ITU Journal on Future and Evolving Technologies*, vol. 3, no. 3, pp. 570–588, 2022.

[4] H. Chergui, L. Blanco, L. A. Garrido, K. Ramantas, S. Kukliński, A. Ksentini, and C. Verikoukis, "Zero-touch ai-driven distributed management for energy-efficient 6g massive network slicing," *IEEE Network*, vol. 35, no. 6, pp. 43–49, 2021.

[4] Online: https://www.elastic.co/what-is/elk-stack, Available: April 2023.

[5] Online: https://grafana.com/, Available: April 2023.