A Cloud Native SLA-Driven Stochastic Federated Learning Policy for 6G Zero-Touch Network Slicing

Swastika Roy[(1)], Hatim Chergui[(1)], Luis Sanabria-Russo[(1)] and Christos Verikoukis[(2,3)]
[(1)] CTTC, Barcelona, Spain
[(2)] University of Patras, Greece and
[(3)] Iquadrat Informatica S.L., Spain
Contact Emails: {sroy, hatim.chergui}@cttc.es,sanabriarusso@gmail.com,
cveri@ceid.upatras.gr

***Abstract*—Sixth-generation (6G)-enabled massive network slicing is a strong enabler for the expected pervasive digitalization of the vertical market. In such a context, artificial intelligence (AI)-driven zero-touch network automation should present a high degree of scalability and sustainability, especially when deployed in live production networks wherein the collected monitoring datasets at different points are non-independent and identically distributed (non-IID). This paper presents a new cloud-native service-level agreement (SLA)-driven stochastic policy to guarantee a scalable and fast operation of constrained federated learning (FL)-based analytic engines that perform statistical slice-level resource provisioning at RAN-Edge in a non-IID setup. Both simulated and cloud-native emulated scenarios are implemented to demonstrate the superiority of the solution in reducing SLA violation, convergence time and computation cost compared to different FL baselines, showcasing thereby a higher scalability.**

***Index Terms*—6G, cloud-native, federated learning, game theory, proxy-Lagrangian, resource allocation, SLA, stochastic policy, ZSM**

I. Introduction

6G networks are expected to intelligently support a massive number of simultaneous and heterogeneous slices associated with various vertical use cases. In this respect, challenges of scalability and sustainability might surface in the deployment of artificial intelligence (AI)-driven zero-touch management and orchestration (MANO) of the end-to-end (E2E) slices under stringent service level agreements (SLAs). In this respect, ETSI has standardized the zero-touch network and service management (ZSM) framework, where a reference architecture and AI-based closed-loop management automation have been proposed [1]. However, the traditional centralized approach for monitoring, analyzing, and controlling the underlying raw data will be problematic because it suffers from significant overhead and delay and a single point of failure. On the other hand, the decentralized approach ensures scalability, low data exchange and, therefore, more security. In this view, distributed artificial intelligence (AI) approaches, particularly FL techniques can play a vital role in monitoring scattered data across the network while reducing the computational costs and enabling fast local analysis and decision. In practice, Cloud-native approaches are considered the primary enabler to develop, build, run, and manage such FL-based analytic and decision functions that are intended to run in live networks [2]. Nonetheless, both the convergence delay and computation cost often limit FL capability under non-IID real network data [3].

A. Related Work

In [4], the author focused on gradient-descent based federated learning (FedAvg) with the local model update and global model aggregation steps as well as analyzed the convergence bound for federated learning with non-IID data distributions, which turns out to suffer from convergence delay. On the other hand, to solve this performance degradation, the authors in [5] proposed a Hybrid-FL, where the server updates the model using data gathered from the selected clients that hold IID datasets and thereafter merges the resulting model with models trained by the rest of the clients. Similarly, the authors of [6] proposed a entropy-based federated averaging scheme using clustering technique to calculate dataset entropy and stochastically client selection policy to significantly stabilize the training and reduce the corresponding computation cost and convergence time. Also, a strategy to improve training on non-IID data has been proposed in [7], which consists on creating a small subset of globally shared data between all Edge devices under network resource constraints and also analyzed the convergence bound of FL.

B. Contributions

In this paper, we present a containerized Cloud-native emulation system that implements a novel scalable SLA-driven

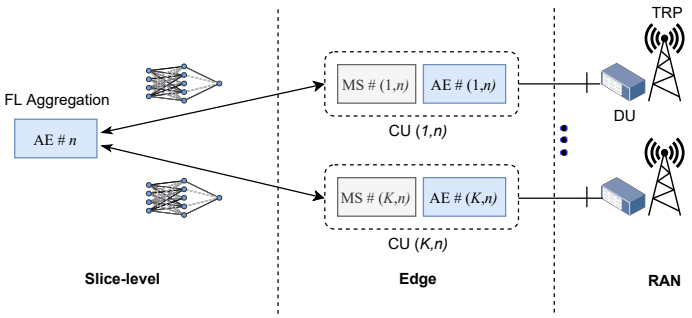

Figure 1: Proposed architecture

stochastic FL for network slicing resource provisioning under SLA constraints. The solution uses Docker compose and Docker containers to facilitate the development and testing of FL applications in 6G networks. Specifically,

- To deal with the FL resource provisioning task at the local analytic engines (AEs), we formulate the corresponding SLA-constrained optimization problem under the *proxy-Lagrangian* framework and solve it via a non-zero sum two-player game strategy.
- To ensure scalability under massive slicing, we design a novel SLA-driven stochastic FL policy for selecting a subset of AEs that will participate in the FL task at each round, which enhances the convergence time while maintaining the same computation cost no matter how the number of AEs increase over the network.
- We deploy the proposed solution in a containerized cloud-native environment and compare our simulated results with the emulated system. Besides, we present our simulated results in both constrained and unconstrained SLA cases.

II. Proposed Network Model and Datasets

As shown in Fig. 1, a 6G RAN-Edge topology under a per-slice central unit (CU)/distributed unit (DU) functional split, wherein each transmission/reception point (TRP) is co-located with its DUs has considered. Basically, each CU consists of a monitoring system (MS) and an AI-enabled slice resource allocation function called analytic engine (AE). Each CU $k\,(k = 1,\ldots,K)$ runs as a virtual network function (VNF) on top of a commodity hardware, and performs slice level RAN key performance indicators data collection to build its local datasets for slice $n\,(n = 1,\ldots,n)$, i.e., $\mathcal{D}_{k,n} = \{\mathbf{x}_{k,n}^{(i)}, y_{k,n}^{(i)}\}_{i=1}^{D_{k,n}}$, where $\mathbf{x}_{k,n}^{(i)}$ stands for the input features vector while $y_{k,n}^{(i)}$ represents the corresponding output.

Assumed that the collected datasets are generally non-exhaustive to train accurate analytical models, a selected subset of AEs takes part in a federated learning task wherein an E2E slice-level AE plays the role of an aggregation server.

Table I summarizes the features and the supervised output of the local datasets, which have been collected from a live LTE-Advanced (LTE-A) RAN with the hourly traffics of the top-10 over-the-top (OTT) applications. These accumulated realistic datasets are non-IID due to the different traffic profiles induced by the heterogeneous users' distribution and channel conditions. Such non-IIDness nature of datasets makes FL algorithms challenging for training.

Table I: Dataset Features and Output

Feature	Description
`OTT Traffics`	Apple, Facebook, Facebook Messages, Facebook Video, Instagram, NetFlix, HTTPS, QUIC, Whatsapp, and Youtube
`CQI`	Channel quality indicator
`MIMO Full-Rank`	MIMO full-rank usage (%)
`# Users`	Downlink Average active users

Output	Description
`CPU Load`	CPU resource consumption (%)

III. SLA Violation-Aware Federated Resource Allocation and Proposed SLA-Driven Stochastic FL Policy

An SLA is established between slice n tenant and the infrastructure provider so that any assigned resource to the tenant should not exceed a range $[\alpha_n, \beta_n]$ with a probability higher than an agreed threshold γ_n. This corresponds to solving a statistically constrained local resource allocation problem with both empirical cumulative density function (ECDF) and complementary ECDF (ECCDF) constraints at FL round $t\,(t = 0,\ldots,T-1)$, i.e.,

$$\min_{\mathbf{W}_{k,n}^{(t)}} \frac{1}{D_{k,n}} \sum_{i=1}^{D_{k,n}} \ell\left(y_{k,n}^{(i)}, \hat{y}_{k,n}^{(i)}\left(\mathbf{W}_{k,n}^{(t)}, \mathbf{x}_{k,n}\right)\right), \tag{1a}$$

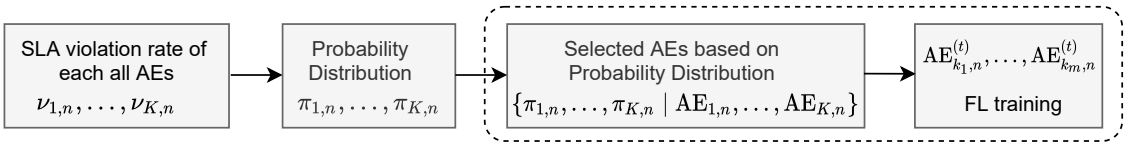

Figure 2: Proposed policy for AE selection.

$$\text{s.t. } F_{\mathbf{x}_{k,n}\sim\mathcal{D}_{k,n}}(\alpha_n) = \frac{1}{D_{k,n}} \sum_{i=1}^{D_{k,n}} \mathbb{1}\left(\hat{y}^{(i)}_{k,n} < \alpha_n\right) \leq \gamma_n, \quad \text{(1b)}$$

$$\tilde{F}_{\mathbf{x}_{k,n}\sim\mathcal{D}_{k,n}}(\beta_n) = \frac{1}{D_{k,n}} \sum_{i=1}^{D_{k,n}} \mathbb{1}\left(\hat{y}^{(i)}_{k,n} > \beta_n\right) \leq \gamma_n, \quad \text{(1c)}$$

which is solved by invoking the so-called *proxy Lagrangian* framework [9]. This consists first on considering two Lagrangians as follows:

$$\begin{aligned}\mathcal{L}_{\mathbf{W}^{(t)}_{k,n}} =& \frac{1}{D_{k,n}} \sum_{i=1}^{D_{k,n}} \ell\left(y^{(i)}_{k,n}, \hat{y}^{(i)}_{k,n}\left(\mathbf{W}^{(t)}_{k,n}, \mathbf{x}_{k,n}\right)\right) \\ &+ \lambda_1 \Psi_1\left(\mathbf{W}^{(t)}_{k,n}\right) + \lambda_2 \Psi_2\left(\mathbf{W}^{(t)}_{k,n}\right),\end{aligned} \quad \text{(2a)}$$

$$\mathcal{L}_{\lambda} = \lambda_1 \Phi_1\left(\mathbf{W}^{(t)}_{k,n}\right) + \lambda_2 \Phi_2\left(\mathbf{W}^{(t)}_{k,n}\right), \quad \text{(2b)}$$

where $\Phi_{1,2}$ and $\Psi_{1,2}$ represent the original constraints and their smooth surrogates, respectively. Specifically, the indicator terms in (1b) and (1c) are replaced with Logistic functions. This optimization task turns out to be a non-zero-sum two-player game in which the $\mathbf{W}^{(t)}_{k,n}$-player aims at minimizing $\mathcal{L}_{\mathbf{W}^{(t)}_{k,n}}$, while the λ-player wishes to maximize $\mathcal{L}_{\lambda}$ [8, Lemma 8]. While optimizing the first Lagrangian w.r.t. $\mathbf{W}_{k,n}$ requires differentiating the constraint functions $\Psi_1(\mathbf{W}^{(t)}_{k,n})$ and $\Psi_2(\mathbf{W}^{(t)}_{k,n})$, to differentiate the second Lagrangian w.r.t. λ we only need to evaluate $\Phi_1\left(\mathbf{W}^{(t)}_{k,n}\right)$ and $\Phi_2\left(\mathbf{W}^{(t)}_{k,n}\right)$. Hence, a surrogate is only necessary for the $\mathbf{W}_{k,n}$-player; the λ-player can continue using the original constraint functions. Via Lagrange multipliers, the λ-player chooses how much to weigh the proxy constraint functions, but does so in such a way as to satisfy the original constraints, and ends up reaching a nearly-optimal nearly-feasible solution [10].

We aim to select only a subset of active AEs in each FL round to optimize the federated learning computation time and the underlying resource consumption. In this regard, we propose an SLA-driven stochastic AE selection policy. Upon the completion of the training at round t, each AE (k, n) evaluates the generalization of its FL model using a typical test dataset $\tilde{\mathcal{D}}_n$ of size $\tilde{D}_n$—that is common to all monitoring systems of slice n— and calculates the so-called SLA violation rate as,

$$\nu_{k,n} = \frac{1}{\tilde{D}_n} \sum_{i=1}^{\tilde{D}_n} \mathbb{1}\left[\left(\hat{y}^{(i)}_{k,n} < \alpha_n\right) \bigcup \left(\hat{y}^{(i)}_{k,n} > \beta_n\right)\right]. \quad \text{(3)}$$

Next, at each FL round t, as presented in Fig. 2, all of the participated AEs send their SLA violation rates to the server which generates a probability distribution using `softmin` function as,

$$\pi_{k,n} = \frac{\exp\{-\nu_{k,n}\}}{\sum_{p=1}^{K} \exp\{-\nu_{p,n}\}}, \quad \text{(4)}$$

Algorithm 1: SLA-Driven Stochastic Federated Learning Policy.

Input: K, m, η_λ, T, L. `# See Table II`
parallel for $k = 1, \ldots, K$ **do**
`# Calculate SLA based violation rate`
AE (k, n) calculates $\nu_{k,n}$ according to 4 and reports it to the aggregation server
end parallel for
`# Federated Learning`
`# Server generates probability distribution using Softmin function`
for $k = 1, \ldots, K$ **do**
  $\pi_{k,n} = \frac{\exp\{-\nu_{k,n}\}}{\sum_{l=1}^{K} \exp\{-\nu_{l,n}\}}$, $k = 1, \ldots, K$
end
Server initializes $\mathbf{W}^{(0)}_n$ with initial training parameter
for $t = 0, \ldots, T-1$ **do**
  `# Server selects the` m `AEs ID using np.random.choice`
  $\mathrm{AE}^{(t)}_{k_1,n}, \ldots, \mathrm{AE}^{(t)}_{k_m,n} \sim \{\pi_{1,n}, \ldots, \pi_{K,n} \mid \mathrm{AE}_{1,n}, \ldots, \mathrm{AE}_{K,n}\}$
  Server broadcasts $\mathbf{W}^{(0)}$ to the m selected AEs
  parallel for $k \in \{k_1, \ldots, k_m\}$ **do**
  `# Local epochs`
  for $l = 0, \ldots, L-1$ **do**
    Solve the proxy-Lagrangian game between $\mathcal{L}_{\mathbf{W}^{(t)}_{k,n}}$ and $\mathcal{L}_{\lambda}$ and get $\mathbf{W}_{k,l}$
  end
  return $\mathbf{W}^{(t)}_{k,n} = \mathbf{W}_{k,L-1}$
  Each local AE k sends $\mathbf{W}^{(t)}_{k,n}$ to the aggregation server.
  end parallel for
  `# FL Server Aggregation`
  return $\mathbf{W}^{(t+1)}_n = \sum_{k\in\{k_1,\ldots,k_m\}} \frac{D_{k,n}}{D_n} \mathbf{W}^{(t)}_{k,n}$
  Broadcasts $\mathbf{W}^{(t+1)}_n$ to all K AEs.
end

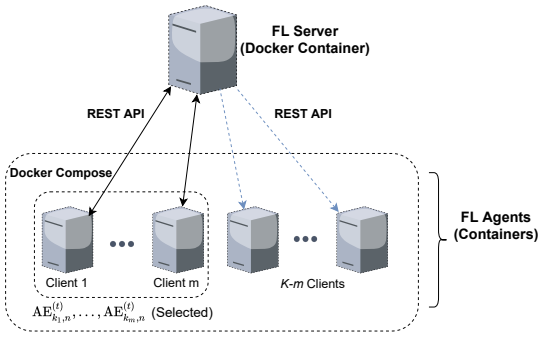

Figure 3: Cloud native implementation of FL agents

wherein AEs with low SLA violation are given a high probability of FL participation to drive the model convergence, but also AEs with high SLA violation might take part in the FL training with a low probability to guarantee the generalization that could stem from their datasets. Based on the probability distribution, only a subset of $m < K$ AEs is drawn at each FL round to

$$\mathrm{AE}^{(t)}_{k_1,n}, \ldots, \mathrm{AE}^{(t)}_{k_m,n} \sim \{\pi_{1,n}, \ldots, \pi_{K,n} \mid \mathrm{AE}_{1,n}, \ldots, \mathrm{AE}_{K,n}\}, \tag{5}$$

Thus, the AEs would have stochastically participated in the FL task while avoiding the concurrent training at each round. And the model averaging at round t is performed as,

$$\mathbf{W}^{(t+1)}_n = \sum_{k \in \{k_1, \ldots, k_m\}} \frac{D_{k,n}}{D_n} \mathbf{W}^{(t)}_{k,n}. \tag{6}$$

Where D_n is the sum of datasets sizes over all slice n's AEs. This proposed procedure is summarized in Algorithm 1. Here, Python with TensorFlow library is used to train and validate the proposed model.

IV. Cloud Native Deployment of FL agents

The main goal is to deploy a cloud-natie approach of FL agents to check the feasibility of our algorithm in the actual scenario and prove the scalability of our proposed policy. To emulate the cloud-native deployments, we use Docker compose tool. The reason behind choosing Docker-compose for deployments is that it usually runs on top of Kubernetes. Also, these kind of implementations expect to be supported by Container Orchestration Engines (COE) offered Slices as Platform as a Service (PaaS), as specified in ETSI NFV IFA 029[11], which is our future target to implement.

A. Architecture

In Fig. 3 illustrates the cloud-native implementation of FL agents where FL Server (OSS server) alongwith one module who is responsible for overall orchestration is besides in one docker container. On the other hand, several AEs, in this case, clients simultaneously run by using Docker compose tool. Through REST API, the Server and clients can communicate with each other. FastAPI as a REST API is used in our implementation because it is a modern, open-source, fast, and highly performant Python web framework used for building Web APIs with Python 3.6+ based on standard type hints [12].

B. Communication process

There are two main modules: the server module, which is responsible for conducting training among clients, and the other module, which is used by all the clients who will participate in the overall FL training process. Communications are performed using the HTTP protocol through several REST interfaces among the server and client nodes.

1) Server Side: The server container contains the main.py module that acts as a controller, the REST API that can communicate with other REST APIs, and the Server class responsible for all the logic. It has the following set of basic REST operations:

- **POST/client:** Registering clients with the Server.
- **GET/select clients:** Initiate policy for selecting clients and corresponding FL training.
- **POST/SLA:** Clients send their SLA violation rate to the Server node.
- **PUT/model-weights:** Clients send calculated model parameters to the Server node.

2) Client side: The client container contains the app.py module responsible for REST API communications and the client module, where the Machine Learning models are implemented. It has the following set of basic REST operations:

- **PUT/SLA:** Server requests each of the clients to calculate their SLA violation rate.
- **POST/training:** Server requests the selected clients to start FL training.

C. Working procedure of the overall network

For running the overall network correctly, the system's first requirement is that the server node is in running mode, and at least one client node is available for training. Following the mentioned way, the overall system will work.

- At first, all clients should know about the server node's IP address, and they register with the server node through the **POST/client** request with their own IP address.
- After registration, the server node send requests to all registered clients to start the proposed clients selection policy through **POST/select-client** request.
- Then, all clients calculate and send their associated SLA violation rate value to the Server through **PUT/SLA** and **POST/SLA**.

- Next, the Server generates the probability ildistribution of all clients using `softmin` function and select clients for training by using `np.random.choice` functions.
- Finally, the Server send **POST/training** requests to the selected clients and start FL training.
- Later, model weights of each clients send to the Server through **PUT/model-weights**, and then the Server calculate the average of model weights and update the overall system and repeat the same procedure for upcoming FL rounds.

V. Results

A. Parameter Settings and Baseline

We consider below three primary slices to analyze the proposed FL policy, defined as follows:

- **eMBB:** Netflix, Youtube and Facebook Video,
- **Social Media:** Facebook, Facebook Messages, Whatsapp and Instagram,
- **Browsing:** Apple, HTTP and QUIC,

Here, the traffic associated with each mentioned slice is the sum of the underlying OTTs' traffics that collects from the hourly traffics of the slices for five days , and the overall summary of those datasets are presented in Table I at section II. We use vectors α, β for the resource bounds, and γ for the thresholds corresponding to the different slices for a particular resource. The parameters settings are mentioned in Table II. Noted that due to the resource constraint of our virtual machine where we run our docker server and clients, we only able to create 40 and 50 containers as clients for training the proposed FL policy.

Table II: Settings

Parameter	Description	Value
N	# Slices	3
K	# AEs	100 (Simul.), $(50, 40)$ (Emul.)
m	# Selected AEs	50 (Simul.), 25 (Emul.)
$D_{k,n}$	Local dataset size	1000 samples
T	# Max FL rounds	30
L	# Local epochs	160
R_λ	Lagrange multiplier radius	Constrained: 10^{-5}
η_λ	Learning rate	0.02

B. Simulated and Cloud-Emulated Results

In the simulated scenario, resources at CU-level are dynamically allocated to slices according to their traffic patterns and radio conditions (average CQI, MIMO full-rank usage). Fig. 3

Convergence: Observing Fig.4, we can conclude that for all slices, policy-based FL converges faster than without policy, since the the AEs with lower violation rate have more chances to participate in the training. On the other hand, Fig 5 shows the results of our emulated FL environment where the proposed

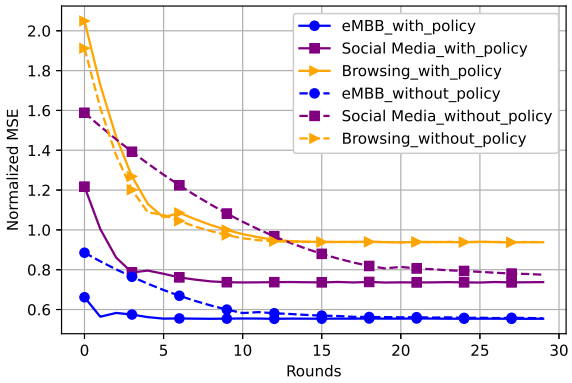

Figure 4: FL training MSE loss vs. number of FL rounds with and without proposed policy for $m = 50$ and $K = 100$. SLA bound, with $\alpha = [0, 0, 0]$, $\beta = [4, 7, 10]$ % and $\gamma = [0.01, 0.01, 0.01]$ in constrained case (Simulated)

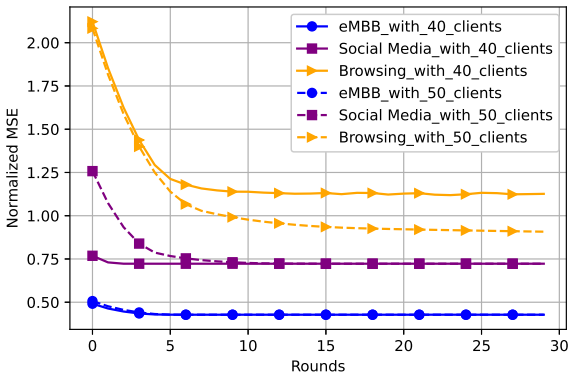

Figure 5: FL training MSE loss vs. number of FL rounds with proposed policy for $m = 25$ and $K = (40, 50)$. SLA bound, with $\alpha = [0, 0, 0]$, $\beta = [4, 7, 10]$ % and $\gamma = [0.01, 0.01, 0.01]$ in real FL network (Emulated)

model has been trained with $K = (40, 50)$ AEs. For both cases, the presented policy-based FL converges faster after 8 rounds with $m = 25$ selected AEs in both simulated and emulated scenarios. Therefore, the selected subset of AEs are sufficient to keep the same performance even if we increase the total number of AEs as the number of slices and CUs increases in the physical network, which ensures scalability.

Scalability: Furthermore, as depicted in Fig. 6, the computation time vs. rounds trend shows that the proposed policy at the emulated system minimizes the computation burden more quickly than the simulated system, while noticing also that the computation load depends on the size of the physical machine. Interestingly, since we settle for a subset of AEs, the computation cost is maintained at the same level no matter how

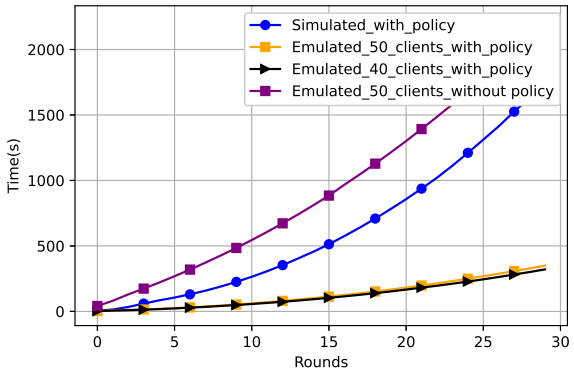

Figure 6: Convergence time of simulated vs emulated with the proposed policy for $m = 25$ and $K = (40, 50)$

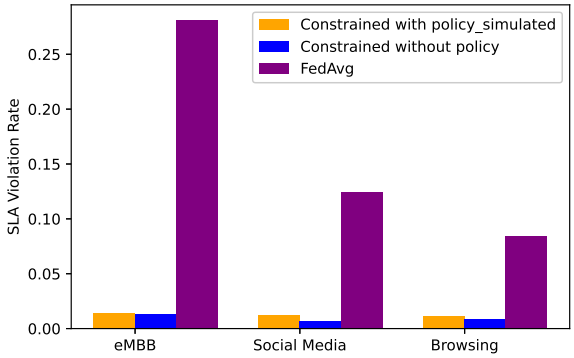

Figure 7: SLA violation rate (Simulated), with $\alpha = [0, 0, 0]$, $\beta = [4, 7, 10]$ % and $\gamma = [0.01, 0.01, 0.01]$

the total number of AEs increases ($K = (40, 50)$)while in the case of deactivated policy, the computation time of emulation system (e.g., $K = 50$ as in Fig. 6) is clearly higher with an exponential trend. Note that we have implemented our docker Server-Clients in such a way that it is only able to tackle IO-bound tasks, not CPU-bound ones. Finally, to investigate the trade-off, the CPU SLA violation rates of the slices are shown in Fig.7, where it is observed that the policy-driven constrained FL presents significantly lower violations compared to FedAvg while preserving the performance of constrained FL without policy (i.e., around 1%).

VI. Future Research Directions

This paper focuses on scalability and sustainability of 6G networks, considering only computing resources. In future, we can concentrate on overall resources such as computing, communication, which may make our model more realistic. Moreover, in actual deployment, the operator/slice tenant needs to understand the FL model's behavior at a specific time (e.g., busy hour) to trust the AI. We can work on the above research directions to build an advanced AI-based trust model, ensure hassle-free processes, and improve security to the 6G heterogeneous networks. Furthermore, we can also focus on further reducing the convergence time more by handling CPU-bound tasks in the emulated system. Also, we intend to integrate our proposed policy in a real multi-technological domain network.

VII. Conclusion

In this paper, we have presented a scalable cloud-native SLA-driven stochastic FL policy for zero-touch network slicing resource allocation. By dynamically selecting only a subset of active analytic engines based on their achieved SLA violation rate, the FL training has been significantly stabilized while minimizing the convergence time and reducing the corresponding computation cost and SLA violation rate. Moreover, Numerical results have been given at both simulated and cloud native emulated scenarios to support these findings and show the proof of scalability using our proposed policy at the emulated system. However, some research challenges are also mentioned in this paper that requires further research effort in that direction.

VIII. Acknowledgment

This work has been supported in part by the training and research project SEMANTIC (861165), the H2020 project MonB5G (871780) and 5G-ERA project (101016681).

References

[1] ETSI GS ZSM 002, "Zero-touch network and Service Management (ZSM); Reference Architecture," Aug. 2019.

[2] O. Arouk and N. Nikaein, "Kube5G: A Cloud-Native 5G Service Platform," in *IEEE Global Communications Conference (Globecom'2020),* pp. 1-6, DOI: 10.1109/GLOBECOM42002.2020.9348073.

[3] P. Kairouz *et al.,* "Advances and Open Problems in Federated Learning," Online. Available: arxiv.org/abs/1912.04977.

[4] H.-B. McMahan *et al.,* "Communication-Efficient Learning of Deep Networks from Decentralized Data.", in *the 20th International Conference on Artificial Intelligence and Statistics (AISTATS'2017).*

[5] N. Yoshida *et al.,* "Hybrid-FL: Cooperative learning mechanism using non-IID data in wireless networks," [Online]. Available: arxiv.org/abs/1905.07210, 2019.

[6] B. Aamer, H. Chergui, M. Benjillali and C. Verikoukis, "Entropy-Driven Stochastic Federated Learning in Non-IID 6G Edge-RAN," in *Frontiers in Communications and Networks,* vol. 2, pp. 45, Oct. 2021. DOI: 10.3389/frcmn.2021.739414.

[7] S. Wang *et al.,* "Adaptive federated learning in resource constrained edge computing systems," in *IEEE Journal on Selected Areas in Communications,* vol. 37, no. 6, pp. 1205-1221, Jun. 2019.

[8] A. Cotter *et al.,* "Two-Player Games for Efficient Non-Convex Constrained Optimization," [Online]. Available: arxiv.org/abs/1804.06500v2.

[9] A. Cotter *et al.,* "Training well-generalizing classifiers for fairness metrics and other data-dependent constraints" [Online]. Available: arxiv.org/abs/1807.00028.

[10] G. J. Gordon, A. Greenwald, and C. Marks, "No-regret learning in convex games," in *ICML'08,* pp. 360-367, 2008.

[11] ETSI GS NFV-IFA 029: "Network Functions Virtualisation (NFV) Release 3; Architecture;Report on the Enhancements of the NFV architecture towards "Cloud-native" and "PaaS".

[12] FastAPI, [Online]. Available: https://realpython.com/fastapi-python-web-apis/. Accessed on 12 Nov. 2021.