On Demand/Agile Deployment of Edge Cloud Infrastructures for Federated Learning

Eduardo Huedo[1], Rafael Moreno-Vozmediano[1], Rubén S. Montero[1], Ignacio M. Llorente[2]
[1]Universidad Complutense de Madrid
[2]OpenNebula Systems

1. Introduction

Edge Computing transfers computational resources and data closer to consumers, whether they are users or devices, mainly providing latency reduction, which enables new responsive services; and traffic savings, which reduce congestion in the core network.

Federated learning (FL) is a method for cooperatively training neural networks between multiple nodes. A server distributes an initial model to clients, who independently update the model using local data and send the model back to the server to update the global model. The main advantage of FL is that privacy is preserved, as data is kept local to each node involved in the training.

FL is well suited for edge computing, since it can leverage the computation power of edge servers to process the data collected on widely dispersed devices. Performing FL on the edge provides benefits like more performant servers, battery saving in the devices or access to aggregated data.

Most research works that propose edge-based FL solutions are based on mathematical models or simulations, but they are not tested nor validated on a real edge computing infrastructure. The main contribution of this work is to demonstrate the viability of deploying a real FL framework on top of a real geo-distributed edge computing infrastructure, based on a commercial edge provider. Furthermore, we propose the use of OpenNebula cloud platform to manage and orchestrate the deployment of the edge infrastructure.

The structure of this paper is as follows. Section 2 describes the related work, both in Edge Computing and Federated Learning. Section 3 introduces the proposed platform to deploy edge clusters on demand. Section 4 describes our approach to perform Federated Learning on the Edge. Section 5 presents the experiments and provides results. Finally, Section 6 summarizes the main conclusions and depicts the future work.

2. Related Work

Edge Computing [Shi16, Sa17] is a many-sided concept that started being applied in different fields, from telecommunications, to Cloud Computing or Internet of Things (IoT). Until Edge Computing was widely adopted, it received different names, like Micro Data Centers (MDCs) [Aa15], Cloudlets [Sa09], Multi-access Edge Computing [MEC] or Fog

Computing [Bo12]. Today, the consensus is that there are actually two types of edge: the far edge and the near edge. The far edge is computing infrastructure deployed in a location farthest from the cloud data centers and closest to the users, for example, 5G cell towers or IoT appliances; while the near edge is computing infrastructure deployed in a location between the far edge and the cloud data centers, for example, in telecom central offices or in ISP's (Internet Service Provider) PoPs (Point of Presence).

Most Edge Computing platforms are based on a distributed architecture, where each edge node is independently managed and an upper-level edge gateway is responsible for interacting with clients and routing client requests to the appropriate bottom edge nodes. This approach is highly scalable; however, it delegates most complex management tasks to the edge nodes, which can consume a lot of their resources. Furthermore, installing, configuring, and maintaining the edge nodes can be an extremely complex and time-consuming task.

In contrast, we proposed a disaggregated architecture based on a centralized management, which releases edge nodes from complex management tasks [IC19]. This approach shows a good tradeoff of performance and design complexity, and provides a uniform view of all the distributed edge infrastructure. The main disadvantage of this architecture is its scalability, which is limited with regard to the number of edge nodes that can be monitored and managed. In a previous work, we presented a disaggregated edge cloud platform based on OpenNebula[1] to create a cloud infrastructure using resources from cloud and edge providers [JGC21].

There is interesting research about the management of distributed edge computing infrastructures. However, most works present simulated results and few of them present real platforms tested with real applications. For instance, ENORM is an edge resource management framework that provides provisioning and auto-scaling of edge node resources tested using AWS and a gaming application [Wa20]. There are also MEC platforms to virtualize the access network and provide cloud-computing capabilities at the network edge [Jarar16, MoRo17].

Regarding technologies, OpenStack StarlingX[2] provides a container-based platform to build mission critical edge clouds, and the Distributed Cloud subproject supports an edge computing solution by providing central management and orchestration for a geographically distributed network of StarlingX Kubernetes edge clusters, which are centrally managed and synchronized over L3 networks from a central cloud. Also, KubeEdge[3] enables the orchestration and management of edge clusters similar to how Kubernetes manages in the cloud.

Finally, cloud providers are providing solutions to extend their services to the client premises. For example, AWS Outposts is a fully managed service that extends AWS infrastructure, services, APIs, and tools to customer premises, using the same programming interfaces as in AWS Regions, while using local compute and storage resources for lower latency and

[1] https://www.opennebula.io
[2] https://www.starlingx.io
[3] https://kubeedge.io

local data processing needs. Similarly, Google Anthos is an application management platform based on Kubernetes that provides a consistent development and operations experience for cloud (Google, AWS and Azure) and on-premises environments.

From the point of view of federated learning, most initial works were based on a cloud model, with a single central server that collects and aggregates the trained ML models of the different clients. However, more recently, edge computing has emerged as a naturally suitable environment to deploy and implement FL solutions [Lim20] [Xia21] [Abr22].

For example, authors in [Wan21] propose a cluster-based FL mechanism for Mobile Edge Computing (MEC) environments, where edge nodes, which act as FL clients, are divided into K clusters by balanced clustering. On each cluster, a leader node (LN) is chosen, which is responsible for aggregating all local models in this cluster. All the edge nodes in the same cluster implement a synchronous aggregation mechanism, while all LNs communicate with the central server for global aggregation following an asynchronous mechanism.

Another interesting proposal is EdgeFed [Ye20], which introduces an additional layer in the FL architecture called ‘client-edge-cloud’. This three-layer model divides the process of updating model parameters into two parts: the local updates of model parameters are performed on ‘client-edge’, and the global aggregation is between ‘edge-cloud’. This model allows to reduce both the computational cost of the mobile devices and the global communication expense.

This three-layer architecture is also analyzed in [Liu20], which compares three different FL models: cloud-based, edge-based and client-edge-cloud hierarchical FL. Cloud-based FL can involve several millions of clients, providing massive datasets, but communications with the cloud server can be slow and unpredictable. In edge-based FL, the server is placed in an edge infrastructure, closer to the clients, thus reducing communication latency. Nevertheless, the main disadvantage is the limited number of clients each server can access, which can result in a training performance loss. Finally, client-edge-cloud hierarchical FL systems get the best of the previous two models: it reduces the costly communication with the cloud, complemented by efficient client-edge updates, and allows managing a large number of clients, so more data can be accessed by the cloud server, which improves the model training performance.

The main limitation of all the above mentioned works is that they are mostly theoretical, and their results are based on mathematical models or simulations, but the proposed FL solutions are not tested nor validated on a real edge computing infrastructure. In this work, we propose and analyze a three-layer edge-based cross-silo FL architecture, and we achieve a real deployment of a FL use case on top of a real edge infrastructure made of several geographically distributed edge nodes from a commercial edge provider (Equinix Metal), and using the OpenNebula platform for the provisioning and deployment of the edge infrastructure.

3. Deployment of Edge Cloud Infrastructures

In this section, we propose an Edge Cloud platform based on OpenNebula that provides the ability to dynamically grow a cloud infrastructure by instantiating Edge Clusters, built using virtual or physical resources from cloud and edge providers like Amazon Web Services, Google Cloud or Equinix Metal. Moreover, resources (e.g. hosts or IP addresses) can be easily added or removed to/from an existing cluster, providing full elasticity. This enables true edge, hybrid and multi-cloud environments to meet latency, bandwidth, privacy or data regulation needs of the workload.

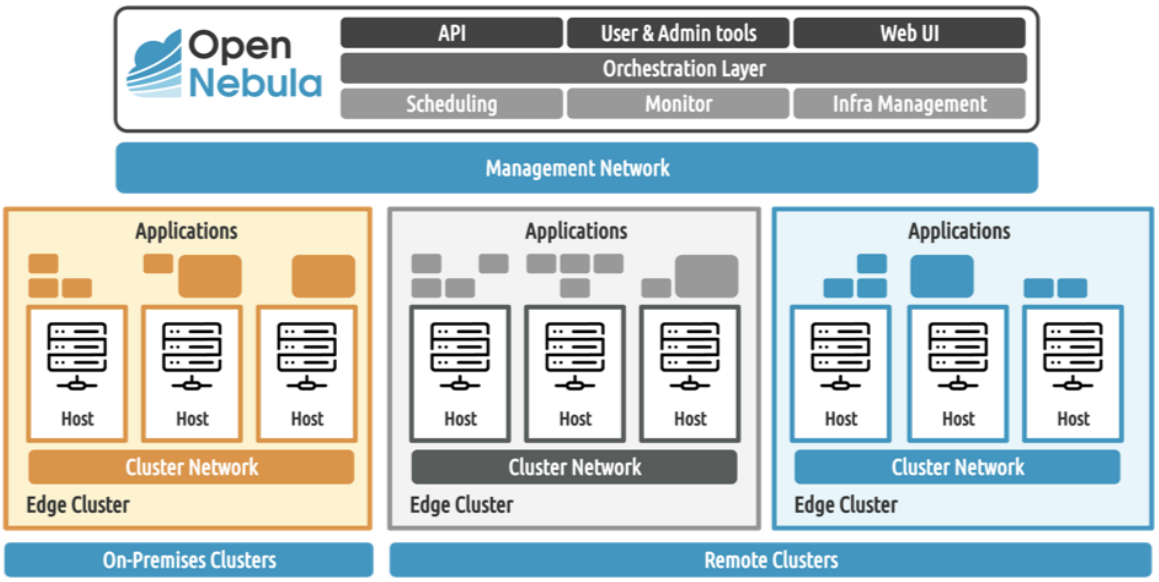

Figure 1. Edge cloud platform.

3.1. Anatomy of an Edge Cluster

EdgeClusters can be created from any of the available providers included in the Provider Catalog. The Provider Catalog includes bare-metal providers, like Equinix Metal, Amazon Web Services or Vultr; as well as virtual machine providers, like Amazon Web Services, Google Cloud, Digital Ocean or Vultr. On premises physical resources can also be used to create an Edge Cluster. Edge Clusters support different workload types, namely: application containers, virtual machines or Kubernetes clusters. As virtualization technology, KVM, Firecracker and LXC can be used with physical resources, while LXC is used with virtual machine providers.

A cluster consists of the logical resources in OpenNebula and the corresponding resources in the provider. OpenNebula provides Edge Cluster templates with common configurations that includes the following resources:

- **Cluster**: Each provision creates one OpenNebula cluster containing all the resources. There is a one-to-one relationship between the provision and the cluster.
- **Datastores**: Each provision deploys two datastores, the Image Datastore, to store a repository of disk images; and the System Datastore, to hold disks for running virtual machines, usually cloned from the Image Datastore.

- **Virtual Networks**: A private and public network are created for the cluster.. The private network by default is based on the VXLAN overlay to not interfere with the provider network management. The public network uses the elastic IP service of the provider to pre-allocate IPs, so VMs have public connectivity.
- **Hosts**: The servers to run the virtual machines or containers. The servers can be bare-metal instances or virtual machines depending on the provider.

During the provision of the cluster, all these resources and their corresponding objects in the provider are created using Terraform[4]. Then, the resources are configured with Ansible[5] in order to install and configure the OpenNebula software. Once the resources are ready, they are enabled in OpenNebula as hosts. A command-line tool (`oneprovision`) and a GUI (FireEdge) provide access to this functionality.

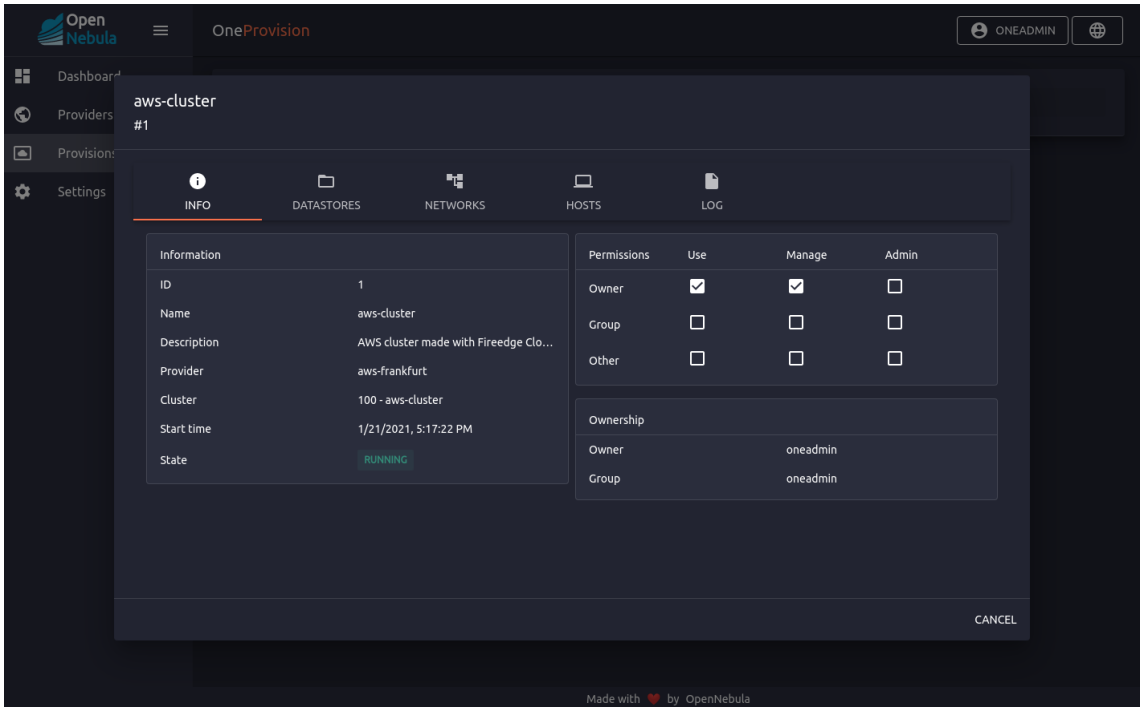

Figure 2. FireEdge GUI for OneProvision.

3.2. Architecture of Amazon AWS and Equinix Edge Clusters

The Amazon AWS Edge Cluster consists of the following resources:

- **Instance**: Virtual or metal AWS Instances to be used as OpenNebula Hosts.
- **Virtual Private Cloud (VPC)**: Isolated virtual network for all the deployed resources.
- **Subnet**: To allow communication between VMs that are running in the provisioned Hosts.
- **Internet Gateway**: To allow VMs to have public connectivity over the Internet.
- **Security Group**: All the traffic is allowed by default, but custom security rules can be defined by the user to allow only specific traffic to the VMs.

[4] https://www.terraform.io
[5] https://www.ansible.com

The network model is implemented in the following way:

- **Public Networking**: Elastic IPs are requested from AWS using a specific OpenNebula IPAM (IP Address Management) driver. Then, the IP is assigned to the Host where the VM or container is running. Finally, inside the Host, IP forwarding rules are applied so the VMs or containers can communicate over the public IP assigned by AWS.
- **Private Networking**: VXLAN (Virtual eXtensible Local Area Network) is used as the overlay protocol and EVPN (Ethernet Virtual Private Network) is used to communicate MAC and IP addresses through BGP (Border Gateway Protocol). This is called VXLAN-EVPN or VXLAN BGP-EVPN.

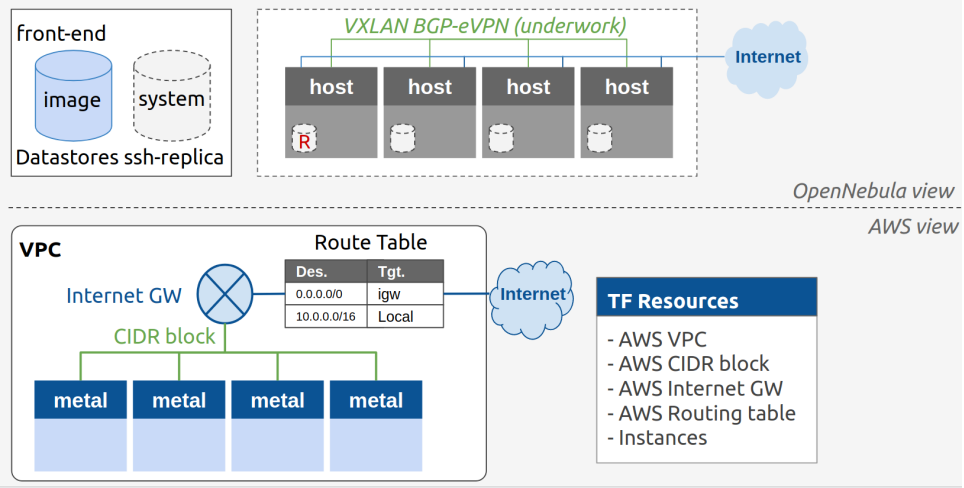

Figure 3. AWS deployment.

For Equinix Metal, a Device is created for the Host, and the network model is implemented in a similar way as in AWS.

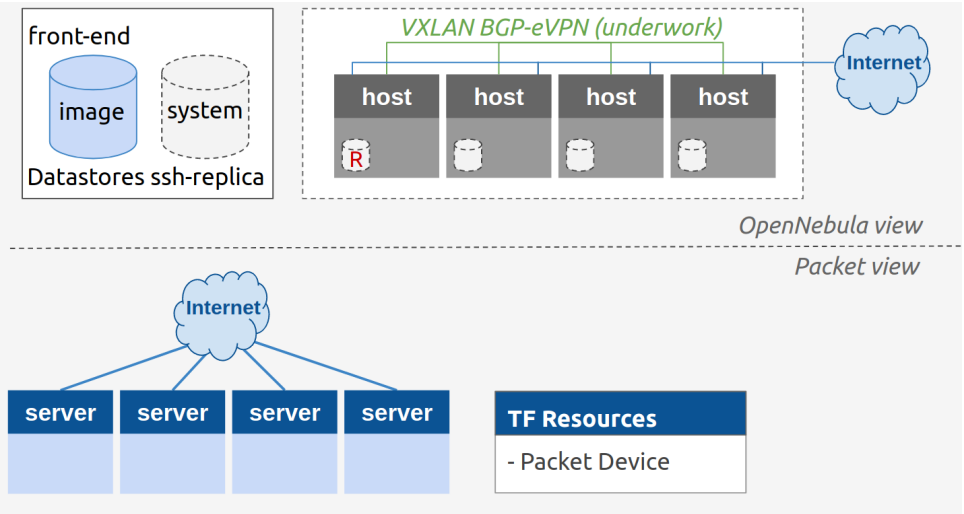

Figure 4. Equinix Metal deployment.

3.3. Distributed Storage for the Edge Cluster

Storage is based on a specialized solution for the efficient management of disk images in highly distributed environments. It has been developed using common technologies that already exist in the Linux operating system, reducing the complexity and technology footprint compared to using specialized distributed storage systems.

The Edge Cluster storage system combines a 3-tier global architecture for image distribution with an enhanced Datastore with replica caching, and backups within each Cluster. The 3-tier storage architecture consists of:

- **Marketplace**. This tier consists of the remote servers and storage implementing the global application image repositories. It includes third-party sites like Docker Hub or Linux Containers, as well as OpenNebula Public Marketplace.
- **Image Datastores**. This tier consists of the Image Datastores and provides the primary image storage location for OpenNebula. This storage area is provided for one or more dedicated nodes. Its contents are cached and replicated on demand to each cluster. This supports the deployment of clusters on any location as well as to scale the on-premises infrastructures.
- **Cluster Replicas**. Application images are cached within a cluster in dedicated replica hosts to minimize image transferring. The replica hosts make use of a specialized distribution system to make the images available to all cluster hosts.

4. Federated Learning on the Edge

Federated Learning (FL) is a machine learning technique which enables multiple parties (clients) to collaborate in training a machine learning model, without exchanging their local raw data, and thus preserving the privacy of this data. FL systems usually work under the coordination of a central server that collects and aggregates the trained models of the different clients into a single model, which is forwarded to the clients. This shared model can be evaluated and refined again by the clients in multiple rounds.

FL systems can be classified into two major categories according to the scale of federation [Kai19]:

- Cross-device FL systems, consisting of a large number of thin clients, such as mobile or IoT devices, typically with limited computational capabilities and a limited training dataset per client. Also, the number of clients is variable, as they can be dynamically added or disabled.
- Cross-silo FL systems, where clients are organizations, data centers, or edge nodes, which can collect training data from multiple local users or devices. Typically, the number of clients is relatively small and almost fixed, with higher computational power and larger training datasets per client.

In this paper we propose an edge-based cross-silo FL implementation, where edge nodes acting as FL clients can be dynamically deployed on an on-demand basis. These edge nodes can collect and aggregate training data from multiple nearby clients or devices. For example, an organization with multiple sites (e.g. hospitals, banks, etc.) may deploy a per-site edge infrastructure to collect training data from local site users; or a service provider

may deploy multiple geographically distributed edge nodes, close to end users or devices, to collect training data from them.

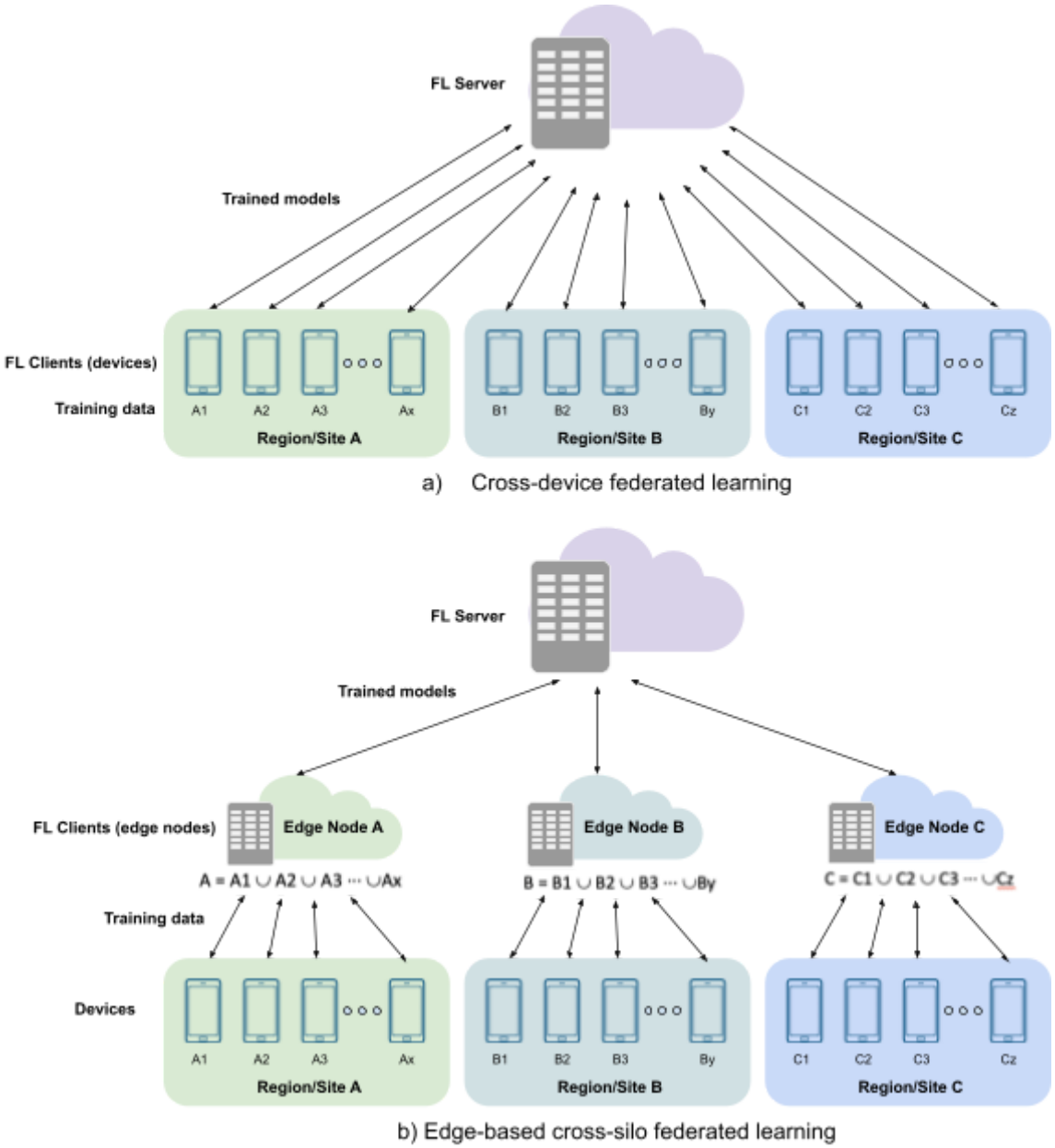

Figure 5. Cross-device vs. Edge-based cross-silo FL

Edge-based coss-silo FL has several advantages compared to cross-device FL:

- Edge nodes typically have higher computational power than end devices, thus the execution time of the ML model training can be reduced.
- Edge nodes collect and aggregate training data from multiple users or devices, so they work with larger datasets than individual devices, resulting in a higher accuracy of the models trained in the edge clients.
- Regarding data communications, we have to distinguish between the transmission of training data and the transmission of trained models. In the edge-based coss-silo FL model, raw training data must be transmitted from end devices to edge nodes before starting the training process, while this is not required in the cross-device FL model.

However, because edge nodes are typically deployed in close proximity to end devices, this transmission is usually done over high-speed, low-latency edge networks, and does not consume Internet bandwidth, so that the impact on overall training time is not significant. On the other hand, considering the transmission of ML models, on each training round, clients transmit their trained models to the server, and the server aggregates these models in a single model which is sent back to the clients. These trained models usually have a fixed size, therefore, in an edge-based coss-silo FL, where the number of clients is much smaller than in cross-device FL, the bandwidth consumed in transmitting trained models is also much lower.

The main disadvantage of edge-based coss-silo FL is data privacy, since training data needs to be transmitted from devices to edge nodes. In this case, we assume, first, that the edge provider is authorized to store this data, and second, that data transmission is secured by using some privacy preserving scheme [Zha20, Wan21].

The main goal of this work is twofold: first, to demonstrate that it is possible to provision and deploy an edge-based cross-silo FL infrastructure easily and within a limited time, using OpenNebula; and second to prove that the performance of this implementation is better than a cross-device FL solution, from the point of view of training time, accuracy of the resulting ML model, and communication latencies and bandwidth consumption.

5. Experiments

5.1. Experimental environment

The experimental environment used in this work is based on the Flower FL benchmark[6] [Beu20] running on an edge-based cross-silo FL infrastructure, which is provisioned and deployed using OpenNebula.

Flower is a novel FL framework that supports experimentation with both algorithmic and systems-related challenges in FL. Major design goals for Flower are being machine learning framework-agnostic, client-agnostic, expandable, accessible and scalable. Flower has several features that are very interesting for our experimental environment: first, Flower can manage a large number of clients training concurrently, allowing researchers to deploy large-scale FL problems and, using reasonable levels of compute, can obtain results at acceptable speed; second, Flower can run on heterogeneous client environments, allowing both to both simulate heterogeneity and to execute FL on real edge devices, and even mixing mixing simulated and real environments; finally, Flower can deal with different ML frameworks, allowing clients to use varying ML frameworks in the same workload.

The main Flower architecture is made of a central server and a variable number of clients. On the server side, there are three major components involved: the *FL loop*, which orchestrates and controls the progress of the learning process; the *RPC server* which is responsible for interfacing with clients, and sending and receiving *Flower Protocol* messages (e.g., *fit*, *evaluate*, etc.); and the *Strategy*, which defines the federated averaging algorithms

[6] https://flower.dev

used for aggregating the model parameters across clients. Different aggregation strategies can be used in Flower. The basic one is *FedAvg* (Vanilla Federated Averaging) that consists of an equal distribution of model parameters for every local model. Other strategies are *Fault Tolerant FedAvg*, which can deal with faulty client conditions such as client disconnections or laggards; *FedProx*, which is able to extend FL to heterogeneous network conditions; *QFedAvg*, which encourage fairness in FL; and *FedOptim*, a family of server-side optimizations that include several advanced algorithms.

Regarding the client side, an outstanding characteristic of Flower is that the server is unaware of the nature of connected clients, enabling model training across heterogeneous client platforms and implementations. For example, clients can be based on Virtual Client Engine (VCE), which enables the virtualization of Flower clients to maximize utilization of the available hardware, or can run on real devices (e.g. Android phones, Raspberry Pis, ARM microcontrollers, Nvidia Jetson devices, etc.), general purpose computers, or virtual machines and virtual devices deployed on a cloud or edge provider.

The experimental testbed used in this work is based on a cloud/edge environment, so that both the Flower server and the Flower clients run on virtual machines deployed on different cloud or edge providers. To emulate cross-device and cross-silo FL platforms, we use virtual machines of different capacity. For example in a cross-device FL platform, where training models run on end-devices, clients are implemented as VMs with limited capacity (e.g. 2 vCPUs and 4 MB RAM). On the other hand, in a cross-silo edge-based FL platform, where training models run on edge nodes, clients are implemented as VMs with higher capacity (e.g. 16 vCPUs and 30 MB RAM). These virtual infrastructures are deployed and managed using the OpenNebula platform.

5.2. Testbed and Results

In this section, experiments and results are presented with the provision of clusters on 3 different locations of Equinix Metal using KVM as hypervisor. Physical servers of `c3.small` machine type are used, with an Intel Xeon E-2278G CPU with 8 cores (16 threads) at 3.40 GHz and 32 GB of RAM.

The OpenNebula frontend is deployed on AWS in Paris. A VM of `t2.large` instance type is used, with 8 GB of RAM and an Intel Xeon Scalable CPU with 2 virtual CPUs at up to 3.0 GHz. All provisions and management operations are done from this server, which also runs the FL server. Table 1 lists the Equinix Metal locations used in the experiments along with the average Round-Trip Time (RTT) measured from the frontend.

Location	RTT (ms)
Frankfurt (Germany)	11
Amsterdam (Netherlands)	14
Madrid (Spain)	20
New York (US)	76
Washington DC (US)	83
Dallas (US)	108
Singapore (Singapore)	173
Sidney (Australia)	282

Table 1. Locations used in the experiments.

We use the CIFAR-10 dataset (162.6 MiB), which is commonly used for evaluating vision models. The dataset consists of 60,000 images (50,000 for training and 10,000 for testing) from 10 different object classes. The images are in RGB format, and are 32 x 32 pixels in size. Partitioning functions implemented in Flower are used to split a dataset across the different clients in a user-defined way.

For training we use a simple 2D Convolutional Neural Network (CNN) in Keras on top of TensorFlow, as shown in Figure 6, with four convolutional layers, two max pooling layer (for downsampling), three dropout layers (to prevent overfitting), and finally flattening out the network to densely-connected layers to make predictions. FedAvg is used as the FL strategy. The global model (5 MiB) is sent after every round of FL and 12 rounds are performed in total. For the first round, each client starts with a random model.

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, 3, padding='same',
            input_shape=(32, 32, 3), activation='relu'),
    tf.keras.layers.Conv2D(32, 3, activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Dropout(0.25),

    tf.keras.layers.Conv2D(64, 3, padding='same', activation='relu'),
    tf.keras.layers.Conv2D(64, 3, activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Dropout(0.25),

    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(10, activation='softmax'),
])
```

Figure 6. Learning model.

We consider two scenarios, training on the device (cross-device FL) and training on the edge (cross-silo FL), with parameters shown in Table 2. We compare performance, traffic, quality and cost.

Parameter	Device	Edge
Clients per location	6	1
vCPUs per client	2	16
RAM per client (GB)	4	30

Table 2. Configuration for the experiments.

Figures 7 and 8 show the deployment and execution times for training on the device and on the edge with 3 locations. The provision time consists of the deployment time by the provider (usually less than a minute in Equinix Metal, unless resources have to be deprovisioned) and the configuration time by OpenNebula. The host enablement time includes enabling the host in OpenNebula and receiving monitoring information until the host is ready. Then VMs are instantiated, including the transfer of the VM image and the boot process. Then, using the OpenNebula contextualization mechanism, the VM is prepared by installing the required software and transferring the needed data. Finally, the FL client is started, which waits for the rest of clients to start the execution.

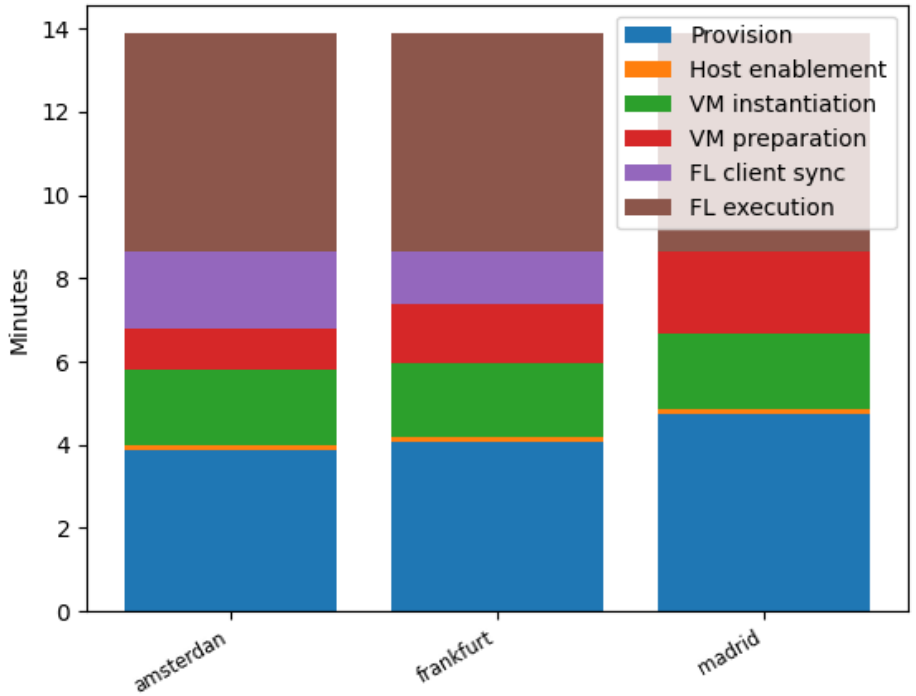

Figure 7. Deployment and execution times for training on the device with 3 locations.

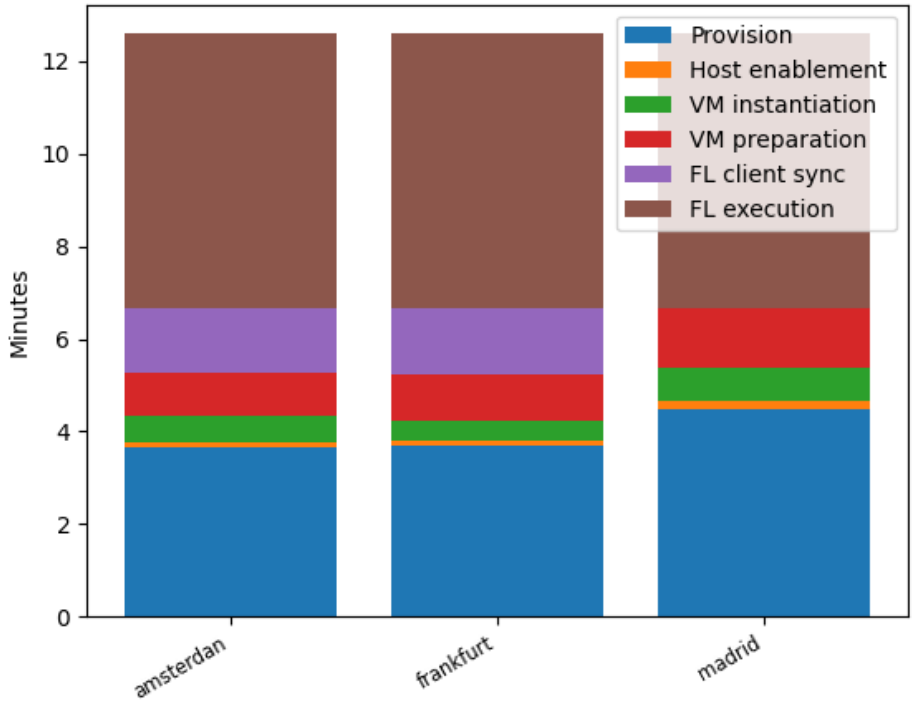

Figure 8. Deployment and execution times for training on the edge with 3 locations.

We can see that provision times are quite consistent in both deployments, with higher provisioning times in Madrid, because resources had to be deprovisioned first. The main difference is in the VM instantiation time, since in the first case 6 VMs were instantiated while only one was in the second case. Regarding the execution time, it is lower when learning in the device, because of the smaller input data and higher data parallelism. However, in terms of quality, learning on the edge provides higher accuracy and lower loss, as shown in Table 3. This is mainly because more data is processed on each client, increasing the chances to extract patterns from similar data.

Metric	Device	Edge
Time	314 s	356 s
Loss	1.47	0.86
Accuracy	0.46	0.70

Table 3. Execution time and quality of the model with 3 locations

To validate the approach at a larger scale, another experiment was performed for training on the edge with 8 locations. The deployment and execution times for this experiment are shown in Figure 9.

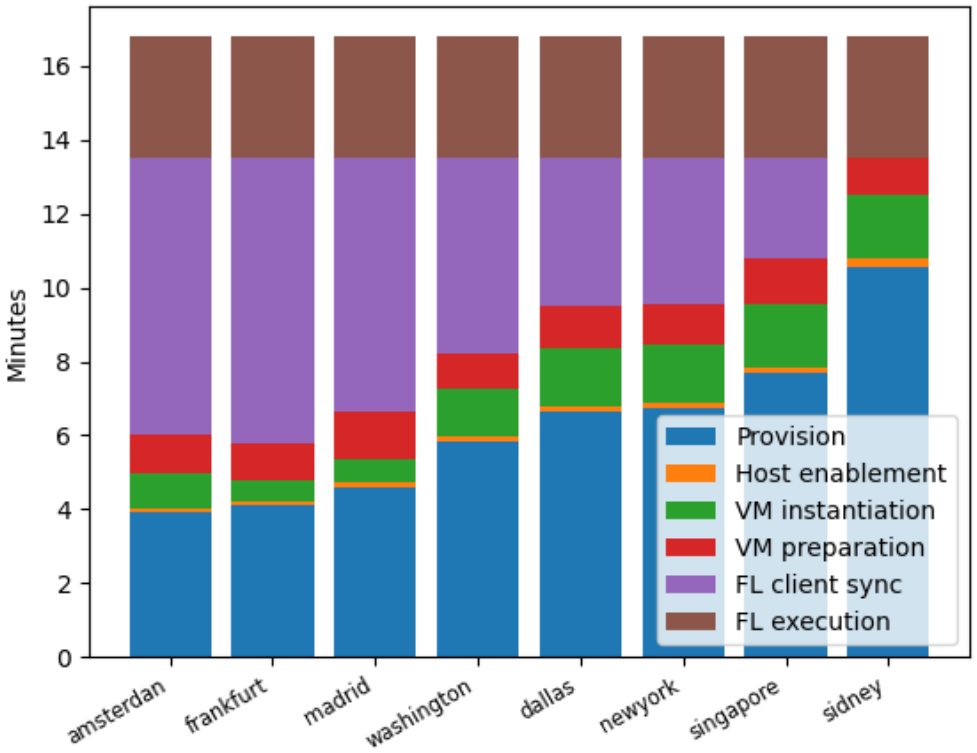

Figure 9. Deployment and execution times for training on the edge with 8 locations.

In this case, we can see that provision and VM instantiation times increase with the latency. On the other hand, FL execution time decreases to 197 s, since the training data is distributed to more clients. As discussed before, the number of clients also determines the quality of the resulting model. In this case, with 1.19 loss and 0.58 accuracy, the quality is worse than with 3 locations on the edge, but still better than with 3 locations on the device.

Finally, Table 4 summarizes the cost of each element of the infrastructure (not including the frontend) and calculates the cost per hour of each experiment. It can be seen that the cost of the experiments is affordable, even more if we take into account that each experiment takes about 15 minutes. Therefore, we demonstrate that the presented approach is feasible for benchmarking and validation experiments on highly distributed edge computing infrastructures, both in terms of performance and cost.

Element	Unit Cost	Training on the device in 3 locations	Training on the edge in 3 locations	Training on the edge in 8 locations
`c3.small` in Europe	$0.83/h	3	3	3
`c3.small` in US	$0.75/h	0	0	3
`c3.small` in Asia	$0.5/h	0	0	1
`c3.small` in Australia	$0.9/h	0	0	1
Public IPv4	$0.005/h	18	3	8
Outbound bandwidth	$0.05/GB	Almost 0	Almost 0	Almost 0
Total cost		$2.58/h	$2.51/h	$6.18/h

Table 4. Cost of the infrastructure.

Acknowledgments

This work was supported by the Ministerio de Ciencia, Innovación y Universidades through the EdgeCloud research project (RTI2018-096465-B-I00) and by the Comunidad de Madrid through the EdgeData research program (P2018/TCS4499).

References

[Aa15] M. Aazam, E. N. Huh. “Fog Computing Micro Datacenter Based Dynamic Resource Estimation and Pricing Model for IoT”. Proc. 2015 IEEE 29th Intl. Conf. Advanced Information Networking and Applications, pp. 687–694, 2015.

[Abr22] H.G. Abreha, M. Hayajn, M.A. Serhani. “Federated Learning in Edge Computing: A Systematic Survey”. Sensors. 2022; 22(2):450. https://doi.org/10.3390/s22020450

[Beu20] D.J. Beutel, T. Topal, A. Mathur, X. Qiu, T. Parcollet, P.P. de Gusmão, N.D. Lane. “Flower: A friendly federated learning research framework”. arXiv preprint arXiv:2007.14390, 2020

[Bo12] F. Bonomi, R. Milito, J. Zhu, S. Addepalli. “Fog computing and its role in the internet of things”. In Proc. First Edition of the MCC Workshop on Mobile Cloud Computing, pp. 13–16, 2012.

[Fen21] C. Feng, Z. Zhao, Y. Wang, T. Q. S. Quek and M. Peng, "On the Design of Federated Learning in the Mobile Edge Computing Systems," in IEEE Transactions on Communications, vol. 69, no. 9, pp. 5902-5916, Sept. 2021, doi: 10.1109/TCOMM.2021.3087125.

[IC19] R. Moreno-Vozmediano, E. Huedo, R. S. Montero, I. M. Llorente. “A Disaggregated Cloud Architecture for Edge Computing”. IEEE Internet Comput. 23(3): 31-36 (2019).

[Jarar16] Y. Jararweh, A. Doulat, A. Darabseh, M. Alsmirat, M. Al-Ayyoub and E. Benkhelifa, "SDMEC: Software Defined System for Mobile Edge Computing," 2016 IEEE

International Conference on Cloud Engineering Workshop (IC2EW), 2016, pp. 88-93, doi: 10.1109/IC2EW.2016.45.

[JGC21] E. Huedo, R. S. Montero, R. Moreno-Vozmediano, C. Vázquez, V. Holer, I. M. Llorente. “Opportunistic Deployment of Distributed Edge Clouds for Latency-Critical Applications”. J. Grid Comput. 19 (1): 2 (2021).

[Kai19] P. Kairouz, H. B. McMahan, B. Avent et al. "Advances and Open Problems in Federated Learning", Foundations and Trends® in Machine Learning 14 (1-2): 1-210 (2021). http://dx.doi.org/10.1561/2200000083

[Lim20] W. Y. B. Lim et al. "Federated Learning in Mobile Edge Networks: A Comprehensive Survey". IEEE Communications Surveys & Tutorials 22 (3): 2031-2063 (2020), doi: 10.1109/COMST.2020.2986024.

[Liu20] L. Liu, J. Zhang, S. H. Song and K. B. Letaief, "Client-Edge-Cloud Hierarchical Federated Learning," ICC 2020 - 2020 IEEE International Conference on Communications (ICC), 2020, pp. 1-6, doi: 10.1109/ICC40277.2020.9148862.

[MEC] ETSI. “Multi-access Edge Computing (MEC)”. Available online: https://www.etsi.org/technologies/multi-access-edge-computing (accessed October, 2022).

[MoRo17] R. S. Montero, E. Rojas, A. A. Carrillo, I. M. Llorente, “Extending the Cloud to the Network Edge”, IEEE Computer 50 (4): 91-95, 2017.

[Sa09] M. Satyanarayanan, P. Bahl, R. Cáceres, N. Davies. “The case for VM-based Cloudlets in mobile computing”. IEEE Pervasive Computing 8: 14-23 (2009).

[Sa17] M. Satyanarayanan. “The emergence of edge computing”. Computer 50 (1): 30-39 (2017). http://dx.doi.org/10.1109/MC.2017.9

[Shi16] W. Shi, S. Dustdar: The Promise of Edge Computing. Computer 49(5), 78–81 (2016)

[Wa20] N. Wang, B. Varghese, M. Matthaiou and D. S. Nikolopoulos: ENORM: a framework for edge NOde resource management, IEEE Trans. Services Computing13 (6), 1086–1099 (2020).

[Wan21] C. Wang, X. Wu, G. Liu, T. Deng, K. Peng, S. Wan. “Safeguarding cross-silo federated learning with local differential privacy”, Digital Communications and Networks (2021). https://doi.org/10.1016/j.dcan.2021.11.006

[Wan21] Z. Wang, H. Xu, J. Liu, H. Huang, C. Qiao and Y. Zhao, "Resource-Efficient Federated Learning with Hierarchical Aggregation in Edge Computing," IEEE INFOCOM 2021 - IEEE Conference on Computer Communications, 2021, pp. 1-10, doi: 10.1109/INFOCOM42981.2021.9488756.

[Xia21] Q. Xia, W. Ye, Z. Tao, J. Wu, Q. Li. “A survey of federated learning for edge computing: Research problems and solutions”. High-Confidence Computing 1: 100008 (2021).

[Ye20] Y. Ye, S. Li, F. Liu, Y. Tang, W. Hu. “EdgeFed: Optimized Federated Learning Based on Edge Computing”. IEEE Access 8: 209191-209198 (2020).

[Zha20] C. Zhang, S. Li, J. Xia, W. Wang, F. Yan, Y. Liu. “BatchCrypt: efficient homomorphic encryption for cross-silo federated learning”, Proc. 2020 USENIX Conference, Article 33, pp. 493–506.