A Spark-based Task Allocation Solution for Machine Learning in the Edge-Cloud Continuum

Loris Belcastro, Fabrizio Marozzo, Aleandro Presta, Domenico Talia
DIMES
University of Calabria
Rende, Italy
{lbelcastro, fmarozzo, apresta, talia}@dimes.unical.it

***Abstract*—The widespread utilization of Internet of Things (IoT) devices has resulted in an exponential increase in data at the Internet's edges. This trend, combined with the rapid growth of machine learning (ML) applications, necessitates the execution of learning tasks across the entire spectrum of computing resources – from the device, to the edge, to the cloud. This paper investigates the execution of machine learning algorithms within the edge-cloud continuum, focusing on their implications from a distributed computing perspective. We explore the integration of traditional ML algorithms, leveraging edge computing benefits such as low-latency processing and privacy preservation, along with cloud computing capabilities offering virtually limitless computational and storage resources. Our analysis offers insights into optimizing the execution of machine learning applications by decomposing them into smaller components and distributing these across processing nodes in edge-cloud architectures. By utilizing the Apache Spark framework, we define an efficient task allocation solution for distributing ML tasks across edge and cloud layers. Experiments on a clustering application in an edge-cloud setup confirm the effectiveness of our solution compared to highly centralized alternatives, in which cloud resources are extensively used for handling large volumes of data from IoT devices.**

***Index Terms*—Machine learning, Internet of Things, edge computing, cloud computing, edge-cloud continuum, Apache Spark**

I. Introduction

Machine learning (ML) has become an essential tool for tackling a wide range of everyday challenges that were once considered insurmountable [1]. Despite their long-standing presence in academic literature, machine learning techniques have achieved widespread adoption, largely driven by the utilization of extensive datasets during training, which has significantly fueled the evolution of machine learning solutions. The training and global deployment of these systems owe much to large distributed computing systems, which enable the execution of advanced algorithms on vast datasets, harnessing their substantial computing and storage power [2].

This work was supported by the research project "INSIDER: INtelligent ServIce Deployment for advanced cloud-Edge integRation" granted by the Italian Ministry of University and Research (MUR) within the PRIN 2022 program and European Union - Next Generation EU (grant n. 2022WWSCRR, CUP H53D23003670006) and by the European Union under the Italian National Recovery and Resilience Plan (NRRP) of NextGenerationEU, partnership on "Telecommunications of the Future" (PE00000001 - program "RESTART").

While traditional distributed systems like cloud or high performance computing (HPC) systems have demonstrated effectiveness in handling advanced machine learning tasks [3], they may not be well-suited to meet the low-latency requirements of real-time applications associated with the Internet of Things (IoT). The proliferation of IoT devices, such as smart cameras, wearables, and smartphones, has resulted in a substantial increase in data generation at the network edge. Transmitting data from edge devices to centralized servers for collection, processing, and analysis incurs high communication costs, which can significantly impact latency times and, consequently, influence the functionality of the application itself.

This paper focuses on the execution of machine learning algorithms in the edge-cloud continuum, with a specific emphasis on their implications from a distributed computing perspective. We explore the integration of traditional ML algorithms, leveraging the benefits of edge computing such as low-latency processing and privacy preservation, along with the computational and storage capabilities offered by cloud computing. This research aims to provide insights into optimizing the execution of machine learning applications by decomposing them into smaller components that can be distributed across processing nodes in edge-cloud architectures. Specifically, a machine learning application can be conceptualized as a sequence of steps, encompassing activities such as data collection, compression, preparation, and local learning. Alternatively, it can be viewed as a sequence of data transformation tasks, including operations like map, filter, and reduce, which can be executed on the data.

This paper presents a novel task allocation solution for machine learning algorithms in the edge-cloud continuum. Our solution considers the multi-layered architecture of the edge-cloud continuum, also taking into account factors such as data locality and computational resource constraints. To assess the effectiveness of our solution, we leveraged Apache Spark, a largely-used framework for parallel task distribution in traditional distributed systems but lacking native capabilities to address the distinctive challenges presented by IoT systems. In response, we guide the Spark scheduler to efficiently manage pools of resources allocated across different layers of the edge-cloud continuum architecture, depending on the tasks to be performed. As a case study, we implemented a

distributed clustering application, specifying at which layer (edge or cloud) certain data transformation operations should be performed. Experimental results demonstrated the effectiveness of our solution on an edge-cloud architecture, showcasing reduced computation times and minimized data exchange over the network compared to a centralized cloud-only solution.

The structure of the paper is as follows. Section II analyzes the related machine learning techniques and solutions of distributed ML on HPC systems, ML on resource-constrained edge devices, and federated learning. Section III discusses how a machine learning application can be performed on an edge-cloud continuum architecture. Section IV shows the architecture based on Apache Spark and the task allocation solution. Section V describes the execution of a clustering application on our architecture. Finally, Section VI concludes the paper.

II. Related work

While previous studies have focused heavily on distributed deep learning models for edge environments, there is a need for further exploration and adaptation of traditional machine learning algorithms, which are commonly used in AI-based applications [4]. Solutions and technologies for deep neural network model training and inference, as well as learning in resource-constrained edge computing environments, have been investigated in different survey works [5], [6]. However, while deep learning was extensively covered, the other machine learning algorithms were only hardly mentioned. Since the goal of this paper is to describe how a generic machine learning algorithm can be effectively deployed on edge-cloud continuum architectures, below we describe the main machine learning techniques and solutions in three specific domains of distributed computing [7]: (i) distributed machine learning on HPC; (ii) machine learning on resource-constrained edge devices; and (iii) federated learning. Table I presents brief descriptions for each domain along with an overview of the main frameworks and libraries that can be used.

A. Distributed machine learning on HPC

Machine learning algorithms demand high-performance machines for optimal execution. However, with the expansion of training data and machine learning models, relying on a single machine becomes impractical [8]. This challenge is effectively addressed through distributed computing, where multiple computing nodes collaborate to simultaneously train a model. Two primary collaboration approaches are used: *data distribution* among worker nodes, where the same algorithm is executed on distinct data partitions, and *model distribution*, with nodes processing the same data but handling different model sections, followed by aggregation of local results. These strategies may employ either a centralized architecture, involving iterative learning with parameter updates and synchronization on a central server, or a decentralized architecture, where communication occurs among worker nodes and model aggregation happens without a central node. In both approaches, distributed learning eliminates the need to accumulate extensive data on a single machine [9]. Various strategies have been proposed to enhance traditional machine learning algorithms on distributed HPC infrastructures, leveraging big data analysis frameworks like Apache Hadoop and Spark [2]. Examples include parallelized versions of Support Vector Machines (SVM) [10] and K-Means [11] using MapReduce, while Spark has been used for distributed implementations of Random Forest [12] and DBSCAN [13]. Over the years, libraries have emerged as essential tools, providing primary implementations of distributed machine learning algorithms. Apache Mahout [14], an open-source library that leverages the Hadoop ecosystem to facilitate the development of scalable machine learning algorithms, such as recommendation mining, clustering, classification, and frequent itemset mining. Conversely, MLlib [15] is the Apache Spark machine learning library, including a set of distributed machine learning algorithms, like classification, regression, clustering, and collaborative filtering. Despite these libraries being vital for machine learning implementations, they are specifically tailored for uniform distributed computing environments like clusters or HPC systems. To effectively adapt to heterogeneous distributed computing environments characterized by varying computation/storage capacities, hardware configurations, and software stacks, these libraries require appropriate modifications; otherwise, their usability is compromised.

B. Machine learning on resource-constrained edge devices

Implementing machine learning at the edge offers significant potential for real-world applications, leveraging low-latency benefits from on-device training and inference near data sources. However, challenges arise due to limitations in computational power, memory, and energy resources, along with hardware heterogeneity, and security concerns in IoT devices. Intensive learning tasks on edge devices primarily focus on deep learning, often using the gradient-descent method. Wang et al. [16] proposed training deep learning models at the edge using local gradient descent on multiple devices. They send local models to an aggregator that computes a weighted average, transmitted back to all edge devices for subsequent iterations. An alternative approach involves training large models on high-performance machines, followed by compression techniques like low-rank approximation and pruning to reduce size. However, downsized models may sacrifice accuracy, necessitating a careful trade-off consideration. To tackle these challenges, the EdgeML library [17] provides open-source algorithms for constructing machine learning models directly on edge devices, with lower memory requirements. Trained models, including tree-based classifiers [18], k-nearest neighbors (kNN) classifiers [19], and recursive neural networks (RNNs) [20], enable rapid and accurate predictions on edge devices like IoT devices and sensors. Projects like TinyML [21] and Tensorflow Lite Micro [22] specifically optimize deep learning inference for memory-limited edge devices, such as microcontrollers, facilitating efficient AI deployment in edge contexts. However, these solutions primarily focus on learning directly on end

TABLE I: Main machine learning techniques and solutions in three specific domains of distributed computing.

Distributed computing domain	Description	Example of frameworks/libraries
Distributed machine learning on HPC	Libraries designed to run algorithms on uniform distributed computing environments (e.g., clusters or HPC systems).	Apache Mahout Apache Spark MLlib
Machine learning on resource-constrained edge devices	Solutions designed to perform learning directly on the end devices with pruned models and do not allow the use of models distributed across devices, edges and clouds	EdgeML TinyML TensorFlow Lite
Federated machine learning	Solutions for the distribution of training/inference operations across edge and cloud levels, they cannot be applied to all classes of ML algorithms (some algorithms need to be properly optimized).	PySyft, FATE, FedML, Flower, Tensorflow Federated (TFF)

devices with pruned models, lacking support for global and complex models distributed across devices, edges, and clouds.

C. Federated machine learning

Traditional distributed machine learning approaches often overlook privacy and security concerns, relying on centralized data management. This becomes problematic, particularly when edge devices with limited defense capabilities are involved, or sensitive data needs to be sent to remote servers. To address this, federated learning has emerged as a paradigm, allowing centralized model training while keeping data distributed across devices, preserving privacy and reducing latency. For example, Kumar et al. [23] proposed Federated Averaging for distributed K-Means, ensuring privacy and reducing latency without sending data from edge devices to a centralized node. Other efforts applied federated learning to ensemble techniques, particularly Random Forest [24], eliminating the need to exchange raw data. Popular federated learning libraries include PySyft, FATE, FedML, Flower, and Tensorflow Federated (TFF). PySyft [25], developed by OpenMined, supports federated learning across various setups, providing secure model weights with features like differential privacy and encrypted computation. However, evolving APIs and outdated documentation present challenges for users. FATE [26] is an industrial-grade framework maintained by the Linux Foundation, employing secure computation protocols and supports various ML algorithms. FedML [27] facilitates federated learning algorithm development across on-device training, distributed computing, and single-machine simulation. Flower [28] is a recent language-agnostic framework that offers high-level abstractions and easy experimentation for federated learning, with a focus on mobile clients and wireless devices. TensorFlow Federated (TFF)[1], developed by Google, integrates tightly with TensorFlow, offering both high-level and low-level interfaces for federated implementations. While these solutions enable distribution of operations across edge and cloud levels, they may not apply universally to all machine learning algorithms. Some algorithms require specific optimization for efficient execution across edge-cloud continuum architectures, spanning device, edge, fog, and cloud levels.

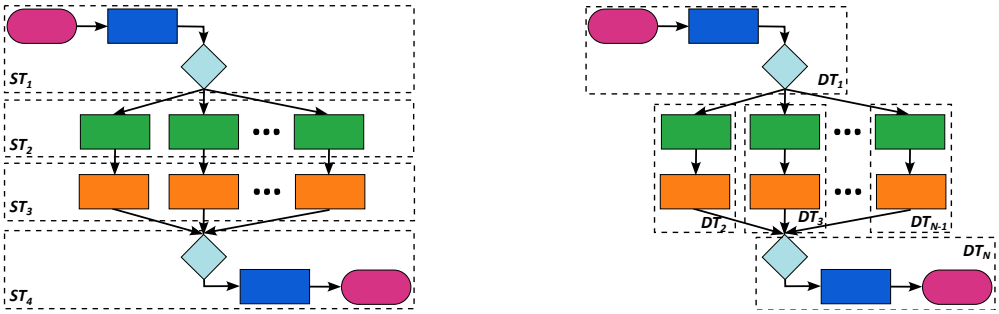
[1]https://www.tensorflow.org/federated

III. Distributed machine learning at the edge-cloud continuum

In this section, we explore the efficient deployment of machine learning applications throughout the different layers of an edge-cloud continuum architecture. Specifically, a machine learning application can be split according to different levels of granularity [29]. It can be viewed as a flow of *steps*, encompassing activities like data collection, compression, preparation, and local learning. Alternatively, it can be represented as a flow of *data transformation tasks*, including specific operations on data like map, filter, and reduce.

a) As a flow of steps: The deployment process of a machine learning application begins with data selection, where relevant datasets are chosen from various sources based on the goals of the analysis. Subsequently, data preprocessing is carried out to ensure data quality, integration, transformation, and reduction. After data preprocessing, the transformed data is subjected to ML techniques, applying algorithms such as clustering, classification, or association rule mining to extract meaningful patterns. Different learning approaches can occur, such as: *local learning* by enabling partial learning on device data; *aggregated learning* by merging and analyzing data from multiple sources; or *global learning* by leveraging large-scale datasets for comprehensive analysis and modeling. The steps of a machine learning application can be distributed on the layers of an edge-cloud continuum architecture as follows:

- At the *device layer*, data can be collected, filtered according to application requirements, and compressed in order to improve storage efficiency, conserve bandwidth, and enhance data transfer time. Preprocessed data can be used to train local learning models that can later be employed in federated learning tasks. This layer is critical for preserving data privacy, reducing latency, and conserving network bandwidth.
- At the *edge layer*, local models from IoT devices can be aggregated to facilitate collaborative learning and enhance predictive capabilities. By aggregating the local models, valuable insights and knowledge can be derived from larger and more diversified datasets, leading to more accurate and robust predictions. Additionally, at the edge layer, data can potentially be cached, allowing for quicker access and reducing the need for frequent data transfers to the cloud or fog servers.



(a) Flow of steps.

(b) Flow of data transformation tasks.

Fig. 1: Tasks of a machine learning algorithm distributed on the layers of an edge-cloud architecture.

- At the *fog layer*, data and models from multiple edge devices can be aggregated, consolidating information to gain collective intelligence. Aggregation allows for collaborative learning and the extraction of comprehensive insights. Moreover, meta-learning can be leveraged to improve the learning capabilities of the architecture.
- At the *cloud layer*, thanks to its high computational capabilities that are not available in other layers, large-scale training, advanced analytics, and large data storage can be performed. It can be leveraged for global learning by aggregating data and models from multiple sources, allowing for a broader understanding of global trends and patterns.

The levels described above differ in their proximity to the data source and computational capabilities, highlighting how the network architecture of edge and cloud computing can be particularly intricate. Figure 1(a) shows how a complex application, composed of several parts, can be split into a sequence of steps.

b) As a flow of data transformation tasks: Machine learning applications can be also represented as a sequence of data transformation tasks, which aim at converting, restructuring, or modifying data from one format, structure, or representation to another. This approach is fostered by the widespread use of functional programming, in which a sequence of common operations are used to manipulate collections of data in a declarative and concise manner. As an example: a filter operation is applied on input data to select specific elements, reducing the size of processed data; then a map operation is used to manipulate data format; finally, a reduce function is used to aggregate data and produce the final result. In particular, data transformation tasks can be categorized as: narrow transformation and wide transformation. The key difference lies in how data is shuffled and in the level of parallelism exploited during their execution. *Narrow transformation* tasks operate on a single partition at a time, not requiring data to be shuffled or rearranged between partitions (e.g, filtering and mapping). These tasks can be executed in parallel across partitions without the need for communication. In contrast, *wide transformation* tasks, which include operations like joining and grouping, require the data to be shuffled or redistributed across multiple partitions. Such tasks require coordination to address potential dependencies among partitions, which can reduce parallelism, and typically incur higher network overhead compared to narrow transformations.

In an edge-cloud architecture, narrow transformation tasks find optimal application at both the device and edge layers, since tasks focus on local data processing within a specific partition and can be performed in parallel without extensive communication. Since data is often generated by devices and naturally organized into partitions, analyzing it at the source reduces latency and network congestion. This approach facilitates faster response times and upholds enhanced privacy preservation. Conversely, wide transformation tasks, which involve data shuffling, are more efficiently executed at the fog or cloud layers. Such tasks leverage increased computational and storage capabilities, enabling the efficient handling of complex, data-intensive operations that require communication and coordination across partitions. Figure 1(b) illustrates how an application composed of several parts can be decomposed into a series of data transformation tasks.

IV. Architecture

As illustrated in Figure 2), our edge-cloud architecture is composed of three layers: i) the device layer, handling data preprocessing and compression; ii) the edge layer, which performs local processing and analysis on data from a group of devices; and iii) the cloud layer, which is dedicated to global data processing and analysis.

To efficiently manage computing nodes, allowing rapid deployment and scaling, the proposed solution has been built using Docker containers. Specifically, a certain number of containers have been allocated as follows: *M* containers were deployed on the edge layer and *N* containers on the cloud layer. The device layer operates independently, generating and compressing data before dispatching them to edge nodes. The containerization approach enhances flexibility and scalability, also facilitating coordination and resource allocation.

Apache Spark serves as the runtime orchestrator for the edge-cloud system, overseeing task allocation on containerized nodes. Spark enables access and reuse of a rich set of code written in Scala, Python, or Java, which can seamlessly run

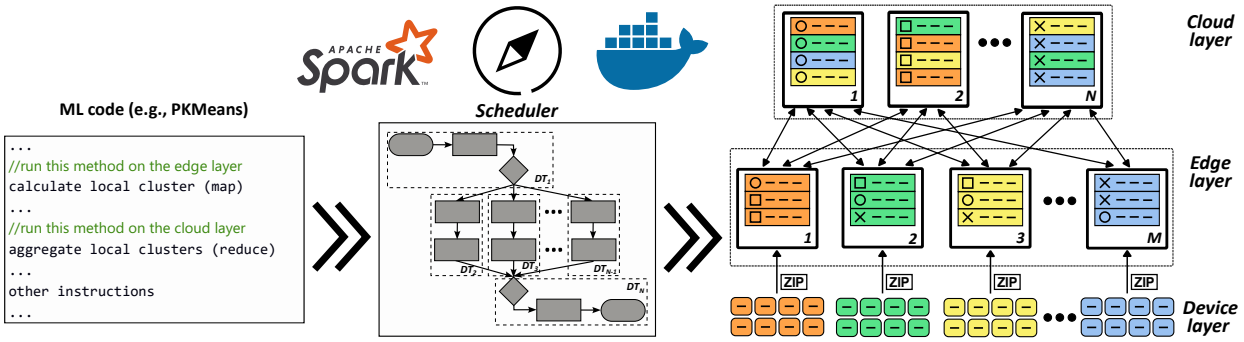


Fig. 2: Execution flow and proposed architecture.

in a distributed environment. Additionally, the Spark MLlib library provides distributed implementations of some popular machine learning algorithms.

The scheduler of Spark is natively designed for distributing tasks across a homogeneous environment, like a cluster, leveraging information about data locality and available resources (i.e., CPUs and memory). Furthermore, the scheduler organizes an application into stages based on operations performed on data, distinguishing between narrow and wide transformations. Specifically, a new stage is initiated when a wide transformation, which requires data shuffling, is encountered. Tasks on data partitions are generated within each stage and executed across cluster nodes, optimizing their execution leveraging in-memory caching. This process is visually represented as a directed acyclic graph (DAG). This scheduling behavior is not capable of covering the needs posed by IoT systems, characterized by limited and heterogeneous computing capacities, energy consumption constraints, latency issues, and diverse technologies and software stacks.

Recognizing the distinctive requirements of IoT environments, it is necessary to enhance the scheduler to ensure efficient resource utilization within the multilayer edge-cloud architecture. To this end, we propose a solution adept at managing pools of resources (nodes) allocated across different layers of the architecture (cloud and edge). Based on the specific data operation required, this approach tries to allocate tasks as follows:

- Narrow transformations, such as *map*, *flatMap*, *filter*, *union*, should be executed on edge nodes, using data received from associated devices.
- Wide transformations, such as *reduceByKey*, *join*, *groupBy*, should be performed on cloud nodes, leveraging aggregated data from devices.

Given a Spark application, our solution comprises: (i) defining distinct pools of compute nodes for each layer of the architecture; (ii) determining data distribution among nodes by defining Resilient Distributed Datasets (RDDs) on a specific layer; and (iii) executing data transformation tasks directly on the newly created RDDs within the designated layer. It is crucial to understand that the scheduler has the responsibility of selecting the optimal node within the chosen layer for task execution, recognizing that this decision might involve moving the data to the selected node. More implementation details are available in the next section.

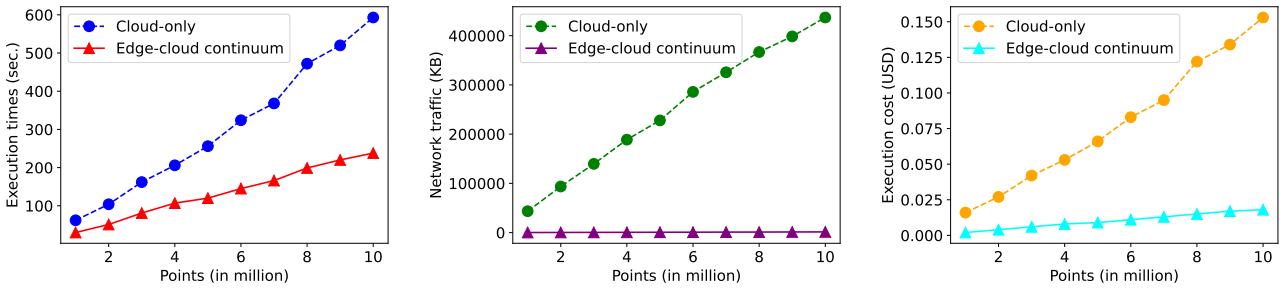
V. A Case Study: A Clustering Application in Spark on Edge-Cloud Continuum

Here we present and discuss a data clustering application exploiting a parallel version of the popular K-Means algorithm, namely PKMeans [11], which is based on the MapReduce paradigm. Specifically, the clustering is performed through the following sequence of data transformation tasks, which are repeated until convergence is achieved:

1) A *map* task that, given a data partition and K initial centroids, assigns each data point to the closest centroid.
2) A *local combine* task, running in parallel on each node, calculating the partial sum of all data points assigned to each centroid. This task minimizes the amount of data shuffled between worker nodes.
3) Finally, a *reduce* task combines the partial sums calculated by the different nodes to compute a global sum for each centroid. The global sum and the total count of data points assigned to each centroid are then used to calculate the new centroid.

To optimize the execution of the algorithm on the edge-cloud continuum, we propose the following scheduling approach: (i) map and combine tasks are executed on the edge layer, aggregating data generated from devices directly on the reference edge node; (ii) the results obtained by edge nodes are then combined at the cloud layer to compute new cluster centroids. This strategy avoids direct device-to-device communication and minimizes communication between the edge and cloud: in this distributed setting, sum vectors are transmitted from each edge node to the cloud, only followed by the back transmission of new centroids to the edge nodes.

Listing 1 shows a concise pseudo-code written in Scala for an application that executes a parallel K-Means clustering algorithm in a distributed Spark environment. The application takes as input the number of clusters (k=10) and the maximum number of iteration (*maxIterations*=30) (line 2). As suggested in [30], the *maxIterations* parameter has been set to 30, which should ensure convergence of the algorithm in most practical situations. Then, the computation layers are configured by setting up an array of preferred locations (IP addresses of



(a) Execution time. (b) Volume of exchanged data. (c) Execution cost.

Fig. 3: Performance comparison between the edge-cloud continuum and cloud-only for the clustering application.

nodes) for both the edge and cloud layers (lines 5-6). After creating a configuration for the Spark application with specified parameters (line 9), the initial centroids are generated (line 12). The algorithm then iterates for a specified number of times (lines 14-24). Within each iteration: (i) data processing starts with a *map* and *local combine* tasks on edge nodes using the *createRDD* method (lines 17-18); (ii) subsequently, a *reduce* phase occurs on cloud nodes using the results from the edge computations (lines 21-22); and (iii) centroids are updated based on the processed data and sent to edge nodes (line 23). From an implementation perspective, the allocation of specific data transformation tasks to a particular layer is achieved by creating a new RDD (Resilient Distributed Dataset) on a set of nodes within the chosen layer.

```
1  //Parameters used by KMeans
2  val (k, maxIterations) = (10, 30)
3  ...
4  //Parameters used to configure the computation layers
5  val preferred_locations_edge = [ip_addr_edge_1, ip_addr_edge_2, ...]
6  val preferred_location_cloud = [ip_addr_cloud_1, ip_addr_cloud_2, ...]
7  ...
8  //Create Spark environment
9  val conf = new SparkConf().set(...)
10 ...
11 //Generate the initial centroids
12 val centroids : Array[(Double, Double)] = ...
13 //Start the iterative process
14 for (i <- 1 to maxIterations) {
15         ...
16         //map and local combine (edge nodes)
17         val edgeRDD: RDD[Array[(Int, (Double, Double))]] =
               createRDD(sc = sc, input = "...", k = k, centr =
               centroids, nodes = edgeNodes)
18         ... //data transformations on edgeRDD
19
20         //reduce (cloud nodes)
21         val cloudRDD: RDD[Array[(Int, (Double, Double))]] =
               createRDD(sc = sc, data = edgeResults, nodes =
               cloudNodes)
22         ... //data transformations on cloudRDD using edge results
23         ... // update centroids and notify them to edge nodes
24 }
25 ...
```

Listing 1: Scala Application

To assess the performance of the proposed approach on an edge-cloud continuum architecture, we carried out a set of experiments to evaluate the behavior of our solution in comparison to centralized approaches that leverage only the cloud. Specifically, using Docker, we set up an execution environment with $M = 10$ containers for the edge level and $N = 4$ instances for the cloud level. Each container was based on Linux Alpine, a lightweight distribution known for low resource usage (both in terms of memory and disk), making it particularly suitable for resource-constrained environments such as in cloud-based or edge computing scenarios. Each container at the edge level was provisioned with 8GB of RAM and 4 CPU cores to emulate the capabilities of a Raspberry Pi 4. Conversely, the cloud instances were configured with 16 CPUs and 64GB of RAM to simulate a server instance m7a.4xlarge provided by AWS. In our experiments, we considered a fixed number of devices, uniformly associated with the different edge nodes, which overall generate a variable amount of data, ranging from 1 to 10 million points.

Figure 3 shows a performance comparison between the edge-cloud continuum and a cloud-only execution for the clustering application. As depicted in Figure 3(a), the edge-cloud continuum outperforms the cloud-only approach in terms of execution time, resulting in a significant reduction in the overall completion time of the application. This improvement becomes more pronounced as the volume of data to be processed increases. In Figure 3(b), we depict the network traffic between the edge and cloud layers, representing the wide area network (WAN) traffic. The internal communication from devices to the edge is not included in the analysis, as it constitutes internal traffic. The results again highlight the superiority of the hybrid approach over the cloud-only solution. This is mainly due to the fact that the algorithm calculates centroids locally on each edge node, exchanging only a small amount of data with the cloud. In contrast, the centralized solution necessitates moving all data from devices to the cloud for analysis, leading to longer execution times and increased data exchange. Furthermore, in Figure 3(c) shows an estimation of the execution cost in the two different architectures. Specifically, for the cloud, the cost of an m7a.4xlarge instance on AWS is 0.93 USD per hour. For the edge devices, we consider the electricity cost of a Raspberry Pi 4 equipped with one cooling fan and one USB flash drive, estimated at 0.009 USD per hour (based on the average electricity price

in the US of 0.139 USD per kWh). In this chart as well, the edge-cloud continuum shows a significantly lower cost compared to the cloud-only version, demonstrating the cost-effectiveness of fully utilizing local edge computing resources. Overall, the hybrid edge-cloud solution significantly reduces execution time, cost, and network utilization by efficiently leveraging local resources available on the edge layer. This advantage becomes even more pronounced as the volume of input data increases.

VI. Conclusion

The widespread use of IoT devices led to an increasing demand for scalable machine learning applications at the network edge, capable of leveraging the advantages of edge computing such as low-latency processing and privacy preservation, in conjunction with the vast resources offered by cloud computing.

This paper presented a novel task allocation solution for optimizing the execution of machine learning applications in the edge-cloud continuum. This solution aimed at decomposing an application into smaller data transformation operations that could be distributed across processing nodes situated at different layers of the architecture (edge and cloud).

We leveraged Apache Spark as the runtime orchestrator for the edge-cloud system, overseeing task allocation on containerized nodes. In particular, we drive the Spark scheduler to efficiently manage pools of resources allocated across different layers of the architecture, encompassing both cloud and edge environments. Testing our approach on a distributed clustering application within an edge-cloud continuum affirmed its effectiveness compared to highly centralized alternatives. This underscores the practicality and advantages of our method in achieving optimal performance and resource utilization in edge-cloud systems.

As future work, we aim to explore new runtimes specifically optimized for dynamic environments, capable of efficiently managing geographically distributed containers (on-premises, near/far edge, or cloud). This avenue of research will further enhance the adaptability and robustness of edge-cloud systems in evolving technological landscapes.

References

[1] I. H. Sarker, "Machine learning: Algorithms, real-world applications and research directions," *SN computer science*, vol. 2, no. 3, p. 160, 2021.

[2] L. Belcastro, R. Cantini, F. Marozzo, A. Orsino, D. Talia, and P. Trunfio, "Programming big data analysis: Principles and solutions," *Journal of Big Data*, vol. 9, no. 4, 2022.

[3] D. Talia, P. Trunfio, and F. Marozzo, *Data Analysis in the Cloud: Models, Techniques and Applications*. Elsevier, October 2015, iSBN 978-0-12-802881-0.

[4] L. Belcastro, F. Marozzo, A. Orsino, D. Talia, and P. Trunfio, "Edge-cloud continuum solutions for urban mobility prediction and planning," *IEEE Access*, vol. 11, pp. 38 864–38 874, 2023.

[5] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1738–1762, 2019.

[6] M. S. Murshed, C. Murphy, D. Hou, N. Khan, G. Ananthanarayanan, and F. Hussain, "Machine learning at the network edge: A survey," *ACM Computing Surveys (CSUR)*, vol. 54, no. 8, pp. 1–37, 2021.

[7] F. Marozzo, A. Orsino, D. Talia, and P. Trunfio, "Edge computing solutions for distributed machine learning," in *2022 Intl Conf on Cloud and Big Data Computing (CBDCom)*, 2022, pp. 1–8.

[8] C. Savaglio, P. Gerace, G. Di Fatta, and G. Fortino, "Data mining at the iot edge," in *2019 28th International Conference on Computer Communication and Networks (ICCCN)*. IEEE, 2019, pp. 1–6.

[9] J. Qiu, Q. Wu, G. Ding, Y. Xu, and S. Feng, "A survey of machine learning for big data processing," *EURASIP Journal on Advances in Signal Processing*, vol. 2016, no. 1, pp. 1–16, 2016.

[10] N. K. Alham, M. Li, Y. Liu, and M. Qi, "A mapreduce-based distributed svm ensemble for scalable image classification and annotation," *Computers & Math with Applications*, vol. 66, no. 10, pp. 1920–1934, 2013.

[11] W. Zhao, H. Ma, and Q. He, "Parallel k-means clustering based on mapreduce," in *IEEE international conference on cloud computing*. Springer, 2009, pp. 674–679.

[12] J. Chen, K. Li, Z. Tang, K. Bilal, S. Yu, C. Weng, and K. Li, "A parallel random forest algorithm for big data in a spark cloud computing environment," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 4, pp. 919–933, 2016.

[13] G. Luo, X. Luo, T. F. Gooch, L. Tian, and K. Qin, "A parallel dbscan algorithm based on spark," in *2016 IEEE International Conferences on Big Data and Cloud Computing (BDCloud)*. IEEE, 2016, pp. 548–553.

[14] "Apache Mahout," https://mahout.apache.org//, accessed May 2023.

[15] "Apache Spark's MLlib," https://spark.apache.org/mllib/, accessed May 2023.

[16] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, "When edge meets learning: Adaptive control for resource-constrained distributed machine learning," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, 2018, pp. 63–71.

[17] Dennis, Don Kurian et al., "EdgeML: Machine Learning for resource-constrained edge devices."

[18] A. Kumar, S. Goyal, and M. Varma, "Resource-efficient machine learning in 2 kb ram for the internet of things," in *International Conference on Machine Learning*. PMLR, 2017, pp. 1935–1944.

[19] C. Gupta, A. S. Suggala, A. Goyal, H. V. Simhadri, B. Paranjape, A. Kumar, S. Goyal, R. Udupa, M. Varma, and P. Jain, "Protonn: Compressed and accurate knn for resource-scarce devices," in *International Conference on Machine Learning*. PMLR, 2017, pp. 1331–1340.

[20] A. Kusupati, M. Singh, K. Bhatia, A. Kumar, P. Jain, and M. Varma, "Fastgrnn: A fast, accurate, stable and tiny kilobyte sized gated recurrent neural network," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[21] "MIT Tiny ML project," https://hanlab.mit.edu/, accessed May 2023.

[22] R. David, J. Duke, A. Jain, V. Janapa Reddi, N. Jeffries, J. Li, N. Kreeger, I. Nappier, M. Natraj, T. Wang *et al.*, "Tensorflow lite micro: Embedded machine learning for tinyml systems," *Proceedings of Machine Learning and Systems*, vol. 3, pp. 800–811, 2021.

[23] H. H. Kumar, V. Karthik, and M. K. Nair, "Federated k-means clustering: A novel edge ai based approach for privacy preservation," in *2020 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*. IEEE, 2020, pp. 52–56.

[24] H. Yao, J. Wang, P. Dai, L. Bo, and Y. Chen, "An efficient and robust system for vertically federated random forest," *arXiv preprint arXiv:2201.10761*, 2022.

[25] A. Ziller, A. Trask, A. Lopardo, B. Szymkow, B. Wagner, E. Bluemke, J.-M. Nounahon, J. Passerat-Palmbach, K. Prakash, N. Rose *et al.*, "Pysyft: A library for easy federated learning," *Federated Learning Systems: Towards Next-Generation AI*, pp. 111–139, 2021.

[26] Y. Liu, T. Fan, T. Chen, Q. Xu, and Q. Yang, "Fate: An industrial grade platform for collaborative learning with data protection," *The Journal of Machine Learning Research*, vol. 22, no. 1, pp. 10 320–10 325, 2021.

[27] C. H. et al., "Fedml: A research library and benchmark for federated machine learning," *CoRR*, vol. abs/2007.13518, 2020.

[28] D. J. B. et al., "Flower: A friendly federated learning research framework," *CoRR*, vol. abs/2007.14390, 2020.

[29] F. Marozzo, A. Orsino, P. Trunfio, and D. Talia, "Scaling machine learning at the edge-cloud: a distributed computing perspective," in *2023 19th International Conference on Distributed Computing in Smart Systems and the Internet of Things (DCOSS-IoT)*, Pafos, Cyprus, 2023, pp. 761–767.

[30] A. Broder, L. Garcia-Pueyo, V. Josifovski, S. Vassilvitskii, and S. Venkatesan, "Scalable k-means by ranked retrieval," in *Proceedings of the 7th ACM international conference on Web search and data mining*, 2014, pp. 233–242.