Resource Allocation for Distributed Machine Learning at the Edge-Cloud Continuum

Ippokratis Sartzetakis*‡, Polyzois Soumplis*‡, Panagiotis Pantazopoulos*, Konstantinos V. Katsaros*, Vasilis Sourlas*, Emmanouel (Manos) Varvarigos*‡
*Institute of Communication and Computer Systems, Athens, Greece, ‡National Technical University of Athens, Athens, Greece
isartz@mail.ntua.gr, soumplis@mail.ntua.gr, ppantaz@iccs.gr, k.katsaros@iccs.gr, v.sourlas@iccs.gr, vmanos@mail.ntua.gr

***Abstract*—Edge computing has emerged as a paradigm for local computing/processing tasks, reducing the distances over which data transfers are made. Thus, an opportunity is presented for data transfer-intensive, distributed machine learning. In this paper we develop a solution for serving distributed Machine Learning (ML) training jobs at the edge–cloud continuum. We model the specific requirements of each ML job, and the features of the edge and cloud resources. Next, we develop an Integer Linear Programming algorithm to perform the resource allocation. We examine different scenarios (different processing and bandwidth costs) and quantify tradeoffs related to performance and cost of edge/cloud bandwidth and processing resources. Our simulations indicate that even though there are many parameters that determine the allocation, the processing costs seem to play on average the most important role. The cloud b/w costs can be significant in certain scenarios. Finally, in certain examined cases, significant monetary benefits can be achieved through the collaboration of both edge and cloud resources when compared to using exclusively edge or cloud resources.**

Keywords— cloud computing, distributed machine learning, edge-cloud continuum, edge computing, resource allocation

I. Introduction

Mobile phones, intelligent vehicles, energy meters and other Internet of Things (IoT) devices are improving and spreading into many areas of society empowering new digital services. The abundant edge devices generate enormous amounts of data. To tackle these developments, new computing paradigms (e.g., edge computing) arise to process these data. As a result, an opportunity for machine learning training arises to use the abundant processing infrastructure created and commoditize the related analytics services.

In distributed machine learning [1][2] the training is performed on dedicated edge or cloud resources. This means that powerful computation resources are employed. Moreover, the training of an ML job is divided into a number of ML tasks that are executed in parallel in different equipment [3][4]. In distributed ML performed over the edge-cloud continuum, an important challenge is to allocate the most appropriate resources to serve each ML job with a certain objective (e.g., minimize the monetary cost). The challenge is similar to computation offloading [5], but presents additional complications due to the requirements of distributed ML. First, a decision has to be made on whether a job will be served at the edge or at the cloud. The decision mainly depends on the cost requirements of each job and the corresponding parameters of the edge and cloud resources. Then the suitable number and type of resources should be allocated according to the needs of the ML job (e.g., amount of data, type of ML training algorithm). The problem requires modeling of the different contributors to the (bandwidth and processing) cost of a job that could be different at the edge and at the cloud. Moreover, there are different types of distributed ML architectures depending on, e.g., the type of parallelism, the communication architecture, and computation timing.

In this paper we investigate the resource allocation problem for distributed machine learning applications. More specifically, we consider a scenario where various devices are located at the edge of the network. The devices produce data that are used for ML training at edge or cloud resources. The goal is to assign the required resources (processing, memory, storage, bandwidth) for each machine learning job while optimizing certain metrics.

The main contributions of our work are:

i) We present an resource allocation model for serving distributed ML training jobs. We jointly take into account both edge and cloud resources, their computational performance, their bandwidth and processing monetary costs. The model allows for exploring insights on the various benefits and tradeoffs.

ii) We present an Integer Linear Programming algorithm that solves the edge/cloud joint resource allocation problem. The objective is to minimize the monetary cost to serve all the ML jobs. The formulation is versatile, and can be used to allocate resources for various variants of distributed ML applications, training (such as model or data parallelism, and also all-reduce or aggregation servers). The algorithm can provide a timely solution to large instances of the problem.

iii) We perform realistic simulations experiments to quantify the tradeoffs between edge and cloud resources for various b/w and processing costs of edge vs cloud.

II. Related Work

Our work is mainly related to two topics: distributed ML and computation offloading. Distributed ML is an active research topic. There are three main taxonomies of distributed ML [3][4] based on the: i) type of parallelism, ii) communication architecture, iii) computation timing. Their specific characteristics have to be taken into account to design a robust resource allocation formulation for each case. As far as parallelism is concerned, there is the model parallelism and the data parallelism. In model parallelism the model is divided into a certain number of segments that are executed in a respective number of workers. The training is performed using the same data in all workers. A main reason to adopt model parallelism is memory limitations. In data parallelism, the model is common to all workers, but the training data are separate. Each worker computes locally its model weights and communicates its values to the other workers to aggregate the results and update the common model. Regarding the

978-1-5386-8347-7/22/$31.00 ©2022 IEEE

communication architecture, a prominent variant is the parameter (aggregation) server. In this case the workers communicate their local computations to the server that aggregates the workers' weights. A different communication architecture is all-reduce. In this case the workers directly communicate with each other to average the model weights. Concerning computation timing, there are two main approaches: synchronous and asynchronous learning. In synchronous learning, the aggregation of the workers' model weights is performed simultaneously. This can incur inefficiencies, when some workers (stragglers) are performing worse than others. In asynchronous learning the workers are allowed to perform at their own pace. A pipelined architecture can be considered as in [4] to improve training throughput. It allows overlapping communication with computation time and reduces the required communication.

Computation offloading initially referred to moving computationally intensive tasks to the cloud where powerful and abundant resources were available. As technology evolved, new applications required low latency and high bandwidth. As a result, Mobile Edge Computing or Multi-access Edge Computing (MEC) emerged. MEC provides significant computation and b/w efficient capabilities at the network edge, close to the users. There is vast relevant research related to MEC computation offloading [5][6][7]. Various works target to minimize execution delay and energy consumption, optimize throughput-network costs, and find an optimal collaboration between edge and cloud resources. Even though the research on computation offloading is vast, still it cannot directly be applied to our case study. The reason is the specific resource allocation requirements of distributed ML that can vary depending on the specific type of ML training algorithm, the accuracy, time constraints, and the architecture.

A recent research topic is the intersection of distributed ML and computation offloading, which is the topic of this work as well. Regarding distributed ML at the edge, [8] studied the efficient utilization of the network's resources by analyzing the convergence rate of distributed gradient descent. The authors of [9] considered ML training of data from augmented reality edge devices. The work in [10] considered incremental offloading of a training model to edge devices. The authors of [11] compared the performance of federated learning to variants of edge and centralized learning. Ref. [12] jointly considered the data collection problem and the resource allocation to maximize the distributed ML throughput. The work in [13] investigated offloading of IoT deep learning applications in an edge computing environment.

To the best of our knowledge there is no previous work that combines realistic modeling of the resource allocation problem of distributed ML using *both* edge and cloud resources and accounting for different architectures. Moreover, an analytic comparison of various tradeoffs between b/w and processing costs of the edge and cloud seems to be missing from related work.

III. Problem Statement

A. Scenario Description

We consider various ML scenarios consisting of certain devices (e.g., vehicles, IoT) at the edge (Fig. 1). Each scenario could correspond to a different type of machine learning job, e.g., image recognition, anomaly detection, etc. We assume that there is an edge network close to the devices, and a more distant cloud. The ML jobs are served either at the edge or at the cloud, depending on specific requirements that will be discussed later. If the tasks of a job are served both at the edge and at the cloud then the large variance of the communication time for the exchange of the model weights may significantly impact the performance of the training algorithm. In the following we describe use cases in a variety of IoT and in Internet of Vehicles (IoV) environments.

In recent years IoT has spread across numerous applications and domains. Billions of devices are connected to the internet, gather data from their sensors and communicate with other devices. IoT applications range from smartphones, smart home devices and smart manufacturing in industries. IoT devices can have several different types of sensors: image, sound, environmental, etc. Depending on the application and the type of the sensor, the data could be generated in high volume and streaming fashion. According to [14], the data

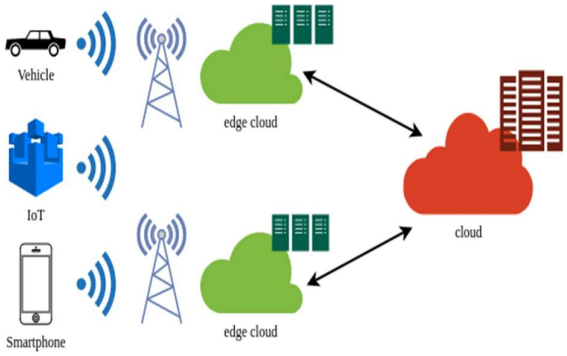

Fig. 1 The abstract architecture considered

produced by all the sensors of a smart city could be approximately 8 GB/day. The data can be used in a large range of machine learning applications: image-voice-activity recognition, and in sectors such as smart electricity grids and healthcare. In many of these scenarios, the continuous learning is important in order for the model to be adaptable to the environment and to other changes.

Automotive industry is in the dawn of a major transformation fueled by the developments in autonomous driving and the involvement of IT companies. IoV is a network of "smart" vehicles that are interconnected and exchange data to enhance traffic safety and efficiency and provide commercial infotainment. IoV has significant processing and communication capabilities, supported by edge and cloud resources. Future vehicles will have a large number of sensors (e.g., Light Detection and Ranging, sonar and cameras) that produce enormous amount of data. Each vehicle can generate 4 TB of data per day [15]. This amount of data will put significant stress on the network resources. The data of the vehicles can be leveraged in many different distributed ML scenarios (e.g., dynamic vehicle routing to avoid congestion, or object detection and classification).

B. Problem Formulation

In this section we will formally define the problem to tackle the aforementioned distributed ML scenarios. The formulation is generic in that it can be used for any of the above scenarios and for many different distributed ML architectures. More specifically, we consider a number of devices that continuously produce data. Each device u continuously produces data at a rate of λ_u samples/sec. A set of a certain number and type of devices and their data to be trained, form an ML training job j. The training of an ML job

j is divided into a set of distributed ML tasks T_j that are executed in respective workers. Each ML task t_{jk} is responsible for a subset d_{jk} of the entire dataset D_j of each job j. A device u_{jk} is related to the k^{th} ML task and belongs to the set of devices U_j of the j^{th} job.

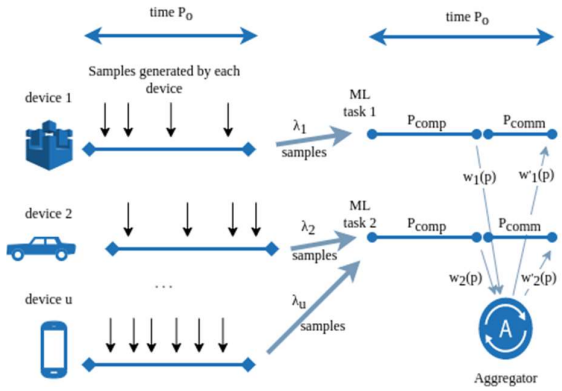

Fig. 2 An example of an ML job with the timings of the training

The time axis is divided in time periods of duration P_o, which are asynchronously defined for each resource. The devices feeding an ML job continuously upload their data to the network's resources that run a training ML algorithm on batches of data. During a period, each resource performs training by processing a batch of data, consisting of data received at it during the previous period (i.e., until the current period begins). Each resource continues to receive data from the devices that will be processed at the following period. At each time period P_o, device u produces and sends for training $s_u = P_o\lambda_u$ samples. Each ML task T_{jk} has a total $S_{jk} = \sum_{u\in U_{jk}} P_o\lambda_u$ samples that it has to process within period P_o. We assume all the ML tasks of a job will be trained within the same time period P_o. This includes both the computation time of the tasks and the communication of the weights (w) of the model from the nodes where the tasks are executed to the aggregation server(s) and back. Note that various devices may send data asynchronously, but computations take place synchronously at the next available time period P_o (Fig. 2). Let P_{comp} the total computation time required to finish the training of a batch of data. Also let P_{comm} the required communication time for the weights. Depending on the architecture of the distributed ML algorithm, there could be different relationships between P_{comm} and P_{comp}. Assuming that the workers first complete the computations and then they send the model weights for averaging (no pipelining, so no overlapping of communication and computation times), then $P_o = P_{comp} + P_{comm}$. Thus $P_{comm} = \omega P_{comp}$, ω<1. The actual communication overhead ω depends on parameters such as the hardware configuration and the specific ML model [4]. Assuming pipelining, the communication overhead can be reduced up to 95%, and a perfect overlap of computation and communication can be achieved [4]. Then, $P \approx P_{comp}$.

In order to perform the computations, each ML task needs certain resources. We assume that each ML task t_{jk} has processing (GPU based), memory, storage, bandwidth and aggregator requirements. These requirements are respectively described by a vector of resources $R^{jk}(Z) = [G^{jk}, M^{jk}, V^{jk}, B^{jk}, A^{jk}]$. The size of the required resources is proportionally dependent on the size of the samples of the task and on the type Z of the ML task. More specifically, an ML task has processing workload $G^{jk}(Z)S_{jk}$ which has to be completed within a time period P_o. Assuming perfect overlapping of computation and communication, the task requires processing rate $G^{jk}(Z)S_{jk}/P_o$. If the overlapping is not perfect, then the processing rate increases to account for the communication overhead. An ML task similarly requires memory $M^{jk}(Z)S_{jk}/P_o$ and storage $V^{jk}(Z)S_{jk}/P_o$. The number of aggregators depend on the assumed architecture. These values can be translated to resource units. Each resource can be assigned in units of predetermined granularity e.g., one unit of memory could correspond to 1GB RAM. The required bandwidth B^{jk} can be derived as follows. Let β the number of required bits to represent a sample. β depends on the nature of each sample (e.g., picture, sentence, etc.). Consequently, the required data rate for task T_{jk} is $B^{jk} = \beta \sum_{u\in U_{jk}} \lambda_{u_{jk}}$.

As far as the network is concerned, we consider an edge and a cloud network. The edge network consists of a set of nodes N. Each node has finite resources that can be used by the machine learning tasks. More specifically, each edge node n has a number of R_n^G GPU units, R_n^M memory units, R_n^V storage units, R_n^B incoming b/w units to receive the data from the devices and R_n^A aggregator units. The cloud network is assumed to have infinite resources. A major difference between the edge and the cloud are the respective processing and bandwidth monetary costs. The cost to use the processing units is defined as C_E^G at the edge and C_C^G at the cloud. The cost of b/w is defined as C_E^{bw} at the edge and C_C^{bw} at the cloud.

C. Resource Allocation Algorithm

In this subsection we present the ILP algorithm responsible for the allocation of the resources. The algorithm gets certain inputs, and using some related constraints aims to allocate the network's resources (the variables of the algorithm), while satisfying the objective. The formulation assumes that there is one aggregation server. It can be modified in a straightforward way for other architectures. Also, we assume that a job can be either entirely served at the edge or at the cloud. The formulation can be expanded to allow more flexible allocation of resources. The algorithm provides a solution for a given period P_o. Whenever the parameters change (e.g., a mobile dynamic scenario), the algorithm is re-executed to provide a new solution.

Inputs:

- N: Set of edge nodes, with n a specific node of the set,
- $R_n^G, R_n^M, R_n^V, R_n^B, R_n^A$: Total number of GPU, Memory, Storage, incoming B/w, Aggregator units at edge node n,
- J: Set of ML jobs, with j a specific job of the set,
- T: Set of ML tasks, with T_j the set containing all tasks of job j. The element t_{jk} is the k^{th} task related to the j^{th} job,
- $G^{jk}, M^{jk}, V^{jk}, B^{jk}, A^{jk}$: Required res. units of task t_{jk},
- C_E^G, C_C^G: The proc. monetary cost at the edge, cloud,
- C_E^{bw}, C_C^{bw}: The b/w monetary cost at the edge, cloud.

Variables:

- ξ_n^{jk}, ξ_C^{jk}:Binary variables (equal 1) if task t_{jk} of job j uses resource units at edge node n (ξ_n^{jk}), or at the cloud (ξ_C^{jk}),
- e_j, c_j: Binary variables (equal 1) if job j uses edge or cloud

Objective:

$$min\left(\sum_j\left(\begin{array}{l}\sum_n\sum_{T_j}\xi_n^{jk}(C_E^{bw}B^{jk}+C_E^GG^{jk})\\+\sum_{T_j}\xi_c^{jk}(C_C^{bw}B^{jk}+C_C^GG^{jk})\end{array}\right)\right) \quad (3)$$

Subject to:

- Each job and all its tasks are served at the edge or cloud:

$$\forall j\in J: e_j+c_j=1 \quad (4)$$

- For each job and each of its tasks, if it employs the edge ($e_j=1$), its tasks will be served once and in one node, since e_j is 1, and the tasks variables are summed:

$$\forall t_{jk}\in T_j: \sum_{n\in N}\xi_n^{jk}=e_j \quad (5)$$

- If a job employs the cloud, its tasks will be served once:

$$\forall t_{jk}\in T_j: \xi_c^{jk}=c_j \quad (6)$$

- Each edge node should have enough (#GPUs, memory, storage, bandwidth, aggregator) capacity to serve the assigned tasks. So, for all nodes we sum all the resources that a job could potentially use (ξ_n^{jk}), and this sum should be less than the capacity of each node:

$$\forall n\in N: \sum_j\sum_{T_j}\xi_n^{jk}G^{jk}\le R_n^G,\qquad \sum_j\sum_{T_j}\xi_n^{jk}M^{jk}\le R_n^M$$

$$\forall n\in N: \sum_j\sum_{T_j}\xi_n^{jk}V^{jk}\le R_n^V,\qquad \sum_j\sum_{T_j}\xi_n^{jk}B^{jk}\le R_n^B$$

$$\forall n\in N: \sum_j\xi_n^{jk}\le R_n^A \quad (7)$$

In Eq. 3 the objective is to minimize the total cost of serving all ML jobs. The first part of the equation refers to the cost of a job if it is served at an edge node n. The cost consists of the b/w units B^{jk} of each task times the cost of edge b/w C_E^{bw}, plus the processing units G^{jk} of each task times the cost of each processing unit C_E^G. The second part of the equation refers to the cost of a job if served at the cloud. It is similar to the calculation of the edge cost, only without the *n* nodes. The ILP algorithm can serve the ML jobs with the objective to minimize the total cost while satisfying the constraints and requirements. In the following section we examine various scenarios and evaluate the tradeoffs in each case.

IV. Simulation Results

To evaluate our proposed resource allocation framework and quantify the edge-cloud cost relationships, we performed a number of simulation experiments. For all the parameters we assigned values that we consider realistic. We assumed a 20-node edge network with finite resources. The network could correspond to the edge facilities of a megacity. For example, in the New York metropolitan area, Google currently operates 8 edge nodes [16]. There are also several other service providers in the area, with their own equipment. In the immediate future further expansion of these facilities is almost certain. Thus, an edge network of 20 nodes seems realistic. We also assumed an abstract cloud with infinite resources. Each edge node has 5 racks. One rack is comprised of 10 servers, and 1 server has 4 GPUs. Thus, each edge node has 200 GPUs. For each edge node we also consider a total of 25 GB RAM, 10 TB of storage, 10 Tbps incoming bandwidth and 6000 CPU physical cores for the aggregators. We assume a total of 100 image recognition ML jobs. This number of jobs could correspond to jobs from fleets of IoV coupled with jobs from networks of IoTs. Each job consists of either 3, 4 or 5 ML tasks, uniformly distributed over all jobs. The sum of the data production rate of the devices of each task ($\sum_{u\in U_{jk}}\lambda_{u_{jk}}$) is 10 samples/sec. We consider that the duration of the training period is $P_o=60$ seconds. Note that the exact number of the ML tasks per job, the sum of the $\lambda_{u_{jk}}$and P_o does not play an important role to the simulation and the resulting tradeoffs. They almost only affect the magnitude of the problem. The size S of each sample (image) of a job is chosen from the following set of integers: [1, 2, 3, 4, 5, 6] MBs and it is uniformly distributed across all jobs. The training performance Π of the GPU is $\Pi=566$ samples/sec. The respective cost at the cloud is \$0.91/hour. The training performance is based on [17]. The benchmarked GPU was NVIDIA V100 corresponding to 1 GPU. The b/w cost to transfer data to the cloud is \$0.02/GB. The respective pricings are taken from [18]. We assume that the training is fully pipelined, i.e., the computation and communication times fully overlap. To find the required number of GPU resources G^{jk} per task, we first multiply the duration of the training period (P_o) by the number of samples/sec of the task (S_{jk}) and by the number of epochs H_j. Then we divide by the performance in samples/sec Π. The (rounded) result is the number of required GPUs for the ML task. The required b/w of each task is derived as follows. We multiply the number of samples of a task by the size in bytes of the task. The calculation of the required storage and memory is trivial and does not play significant role in the allocation.

We examined a set of different parameters to evaluate the tradeoffs between processing-b/w cost at the edge and at the cloud. More specifically, we assumed different: i) edge vs cloud bandwidth costs, ii) edge vs cloud processing costs, iii) number of epochs. According to [19] the edge's b/w costs can be approximately 0.1 times the cloud's. We therefore assumed that the edge b/w cost could be [0.5, 0,25, 0.1] times the cost to transfer the data to the cloud. Moreover, according to [20] the edge processing costs can be approximately 1.5 times the cloud processing costs. We therefore assumed that the processing costs at the edge could be [1, 1.5, 2] times more than that of the cloud. The number of epochs affects the required processing units required to complete a training round within the time period P_o. The required number of epochs can depend on factors, such as the type of the ML algorithm, the layers of the Neural Network, the desired accuracy, whether the training is continuous or not, and many other parameters. According to [21], the number of epochs required for certain benchmarks to reach the required accuracy can vary from 5 to approximately 50 epochs. In other cases, a larger number of epochs may be required. In our problem statement, we assume continuous learning with different datasets. This means that each dataset can potentially employ low number of epochs. On a long enough timeline, the accuracy of each ML model will converge to the required. We assume that the number of epochs can be [1, 10, 20, 40, 80, 160]. For the simulations we used a desktop computer with a quad-core CPU at 4 GHz with 16 GB RAM. We used Python and Pyomo [22] to code the ILP, and IBM CPLEX to solve the problem. The running time of the ILP algorithm for the

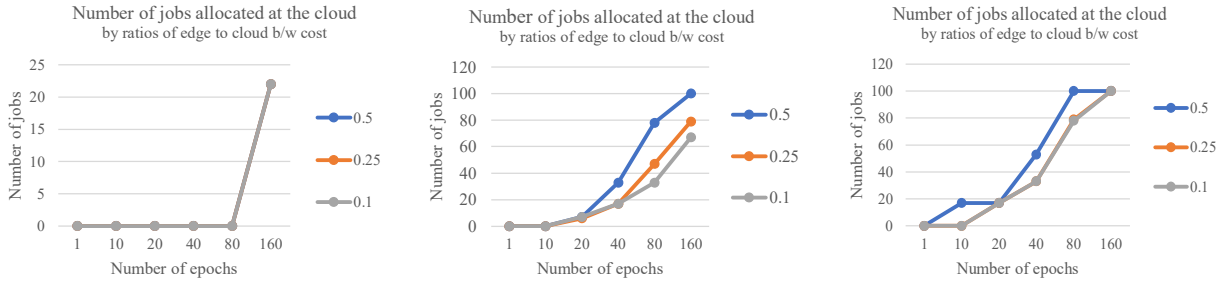

Fig. 3 Number of jobs allocated at the cloud for different b/w costs and ratios of edge to cloud processing costs: (a) 1, (b) 1.5, (c) 2

aforementioned parameters, was approximately 0.7 seconds to create the equations and 0.8 seconds to prepare the solver *and* find the solution (the total number of variables was approximately 11000). The optimality gap was always 0.00%. For a larger instance of 60 edge nodes and 200 jobs the running time in total was approximately 2 and 2.5 seconds respectively (61000 variables). The ILP can provide a timely solution even in large instances of the problem, and a heuristic is not necessary. Future variations of the problem, can be more computationally intensive, and a heuristic could be required.

TABLE I. SIMULATION PARAMETERS

Symbol	Value	Symbol	Value
N	20 nodes	P_o	60 sec
R_n^G, R_n^B	200 GPU, 10 Tbps	S	1-6 MBs
J	100 jobs	Π	566 samples/sec
$\|T_j\|$	3, 4, 5 tasks	C_C^G	\$0.91/hour
$\sum_{u \in U_{jk}} \lambda_{u_{jk}}$	*10 samples/sec*	C_C^{bw}	\$0.02/GB

A. *Edge vs Cloud allocation decisions*

In Fig. 3 we show the number of jobs allocated at the cloud as a function of the number of epochs and for different edge/cloud processing and b/w costs. For simplicity reasons we do not depict the allocation of the remaining jobs at the edge. In Fig. 3a the processing costs of edge and cloud are equal. For all the b/w edge/cloud cost ratios, the number of jobs allocated to the cloud is the same. When the number of epochs is small, all jobs are served at the edge, since the b/w costs are lower. Only after 160 epochs are some jobs served at the cloud. The increased number of epochs means that the total processing cost of a job play a more important role than the b/w cost to the allocation of the jobs. Note that as we will see in Fig. 4, the cloud serves smaller (in terms of Mbytes) jobs. In Figs. 3b, 3c the edge processing costs are more expensive than the cloud's. In Fig. 3b we notice that the allocation of jobs tips towards the cloud relatively quickly. The different b/w costs seem to play a role for the allocation of the jobs after 20 epochs. Until then all jobs are always served at the edge. In Fig. 3c more jobs are served at the cloud. At 160 epochs all jobs all always served at the cloud. Even though the edge processing costs are twice the cloud's, the edge is still more preferable until 40 epochs. Overall, from Fig. 3 we can conclude that the edge is more preferable to serve jobs with relatively low processing requirements. Also the different b/w cost ratios play a relatively small role in the allocation.

Fig. 4 depicts the mean size in GBs for 40 epochs of a job's task that is served at either the edge or cloud when the edge's processing costs are twice than the cloud's. Similar conclusions can be drawn for different epochs and processing costs (as long as some jobs are served at the edge and others at the cloud). The size of a task depends on the number of samples/sec λ_u of its related devices, the duration of P_o , and the size of each sample of a task. The first two variables are the same for all the jobs we considered. Thus, the differentiating factor is the size of a task's sample. Note also that we have assumed a random number of tasks per job. This means that the definite size of a job depends also on the exact number of the tasks. However, this does not significantly affect the decision on the allocated location of a job. The increased number of tasks not only means more data to transfer (hence increased b/w costs), but also means more samples to calculate (hence analogous increase on the processing requirements). Since we have assumed that the performance of a GPU in samples/sec is constant regardless of the size of a sample, the differentiating factor in whether a job will be served at the edge or at the cloud is the size of its tasks' samples. We notice that the edge tends to serve tasks with large size. It seems that in order for a task to be served at the

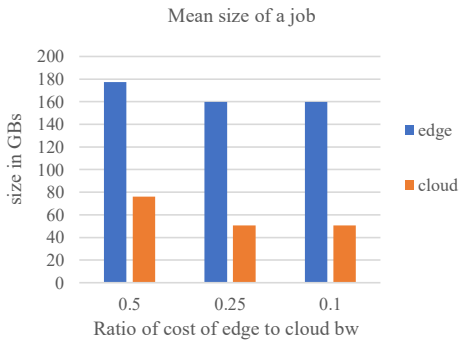

Fig. 4 Mean size of a job allocated in edge or cloud for 40 epochs and for different cost ratios of edge to cloud b/w

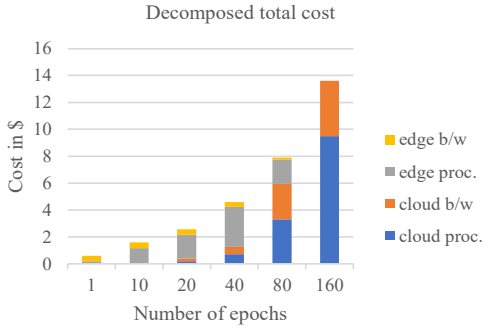

Fig. 5 Total cost of jobs decomposed to edge b/w, edge processing, cloud b/w, cloud processing costs

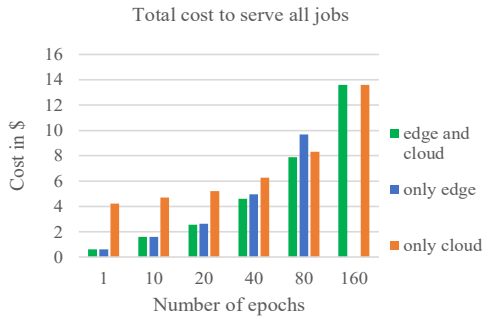

Fig. 6 Total cost to serve the jobs for number of epochs

cloud, it has to be significantly smaller than the tasks that are typically served at the edge. As the b/w cost decrease, the size of the jobs served to the cloud decreases respectively by 34%. The trends are similar for different edge processing costs (not shown here due to limited space). Also, when the number of epochs increase, the contribution of the processing costs due to the additional epochs is increased. Therefore, a job has to be larger to be served at the edge. Note that we considered an image recognition training scenario. Thus, each sample is relatively large. In different applications (e.g., ML translation) the size of the samples can be smaller. This means less b/w required, thus different job distribution at the edge and cloud.

B. *Monetary cost evaluation*

Fig. 5 shows the total cost to serve all jobs for one training round, decomposed to edge and cloud b/w and processing costs and for different number of epochs. Here we assumed an edge to cloud b/w cost ratio of 0.1, and edge to cloud processing costs of 2. Apart from the case of 1 epoch, the main contributor to the costs is processing costs. As the number of epochs increases, edge (and later cloud) processing costs play the most important role to the total cost of the jobs. For 160 epochs all jobs are served at the cloud. Overall, we notice that the *cloud* b/w costs are a considerable fraction (approximately 44% for 40 and 80 epochs) of the overall cloud costs. Thus, in lower number of epochs the edge is more preferable.

In Fig. 6 we present the total cost to serve all jobs to the edge/cloud for the same parameters as Fig. 5. We compare to the cost in case we had only cloud or only edge resources available. We notice that for low number of epochs the cloud is overall much more expensive than the edge, and therefore the cooperation of edge/cloud does not offer monetary benefits when compared to the edge. For 20 and 40 epochs the edge/cloud is 3% and 8% cheaper than the edge. For large training instances, this difference can be monetary significant. For 80 epochs where the processing costs are sizeable, the edge/cloud is 22.4% cheaper than the edge and 5.3% cheaper than the cloud. For 160 epochs all jobs are served to the cloud since the edge is more expensive overall. Also, in this case the edge does not have enough resources to serve all jobs. If it had (requiring 2000 GPUs/node), the total cost would have been $20.37. In either case, the cooperation of edge/cloud can result in significant monetary cost savings when compared to the isolated operation of either the edge or cloud.

V. Conclusions

In this paper we considered the resource allocation problem for distributed machine learning applications. We proposed a framework to allocate resources for training ML jobs at the edge–cloud continuum. We examined various optimization parameters pertained to processing costs and bandwidth costs in both edge and cloud resources. The results indicate that the processing costs play an important role in the allocation of a job to the edge or to the cloud. Significant cost savings were observed through the cooperation of edge and cloud resources when compared to the exclusive use of edge or cloud. Future work includes the allocation of inference jobs along with training, the modeling of energy consumption as well as trade-offs between computation time and accuracy.

Acknowledgment

This work was partially supported by the Horizon 2020 5G-IANA project (grant agreement: 101016427), and the Horizon 2020 SERRANO project (grant agreement: 101017168).

References

[1] Trishul Chilimbi, et. al., “Project adam: Building an efficient and scalable deep learning training system,” USENIX 2014.

[2] Mu Li, et. al., “Scaling Distributed Machine Learning with the Parameter Server,” USENIX 2014.

[3] M. Langer, et. al., “Distributed Training of Deep Learning Models: A Taxonomic Perspective,” IEEE Transactions on Parallel and Distributed Systems, vol. 31, no. 12, 2802-2818, 2020.

[4] A. Harlap, et. al., “PipeDream: Fast and Efficient Pipeline Parallel DNN Training,” arXiv:1806.03377v1, 2018.

[5] P. Mach, and Z. Becvar, “Mobile edge computing: A survey on architecture and computation offloading,” IEEE Comm. Surveys & Tutorials, 19(3), 1628-1656, 2017.

[6] Y. Mao, et. al., “A survey on mobile edge computing: The communication perspective,” IEEE Comm. Surveys & Tutorials, 19(4), 2322-2358, 2017.

[7] F. Saeik, et. al., “Task offloading in Edge and Cloud Computing: A survey on mathematical, artificial intelligence and control theory solutions,” Computer Networks, 195, 2021.

[8] S. Wang, et. al., “When edge meets learning: Adaptive control for resource-constrained distributed machine learning,” INFOCOM 2018.

[9] X. Ran, et. al., “DeepDecision: A mobile deep learning framework for edge video analytics,” INFOCOM 2018.

[10] H.-J. Jeong, et. al., “IONN: Incremental offloading of neural network computations from mobile devices to edge servers,” SoCC 2018.

[11] G. Drainakis, et. al., “On the Distribution of ML Workloads to the Network Edge and Beyond”, INFOCOM 2021.

[12] M. Chen, et. al., “Joint Data Collection and Resource Allocation for Distributed Machine Learning at the Edge,” IEEE Transactions on Mobile Computing 2020.

[13] H. Li, K. Ota, M. Dong, “Learning IoT in Edge: Deep Learning for the Internet of Things with Edge Computing,” IEEE network 32.1, 2018.

[14] A. Sinaeepourfard, et. al., “Estimating Smart City sensors data generation,” In IEEE Med-Hoc-Net, 2016.

[15] J. Zhang, L. Khaled, “Mobile edge intelligence and computing for the internet of vehicles,” Proceedings of the IEEE 108.2, 2019.

[16] “Mapping out edge computing: How dense is it?,” available online: https://www.lightreading.com/the-edge/mapping-out-edge-computing-how-dense-is-it/d/d-id/771128

[17] “Nvidia resnext performance,” available online: https://ngc.nvidia.com/catalog/resources/nvidia:resnext_for_tensorflow/performance

[18] “Amazon ec2 pricing,” available online: https://aws.amazon.com/ec2/instance-types/p3/

[19] “Edge computing and transmission costs,” available online: https://www.datacenterdynamics.com/en/opinions/edge-computing-and-transmission-costs/

[20] “The economics of edge computing,” available online: https://edgecomputing-news.com/2020/10/29/analysis-economics-of-edge-computing

[21] P. Mattson, et al., “MLPerf Training Benchmark,” ArXiv abs/1910.01500 (2020).

[22] W. Hart, et. al.,“Pyomo–Optimization Modeling in Python,” Springer, 2017.