Dynamic Edge/Cloud Resource Allocation for Distributed Computation under Semi-Static Demands

Ippokratis Sartzetakis, Panagiotis Pantazopoulos, Konstantinos V. Katsaros, Vasilis Sourlas, Emmanouel Varvarigos

***Abstract*—Edge computing is a recent paradigm where the processing takes place close to the data sources. It therefore reduces latency and saves bandwidth compared to traditional cloud computing. The latter can continue to play a supportive role. Edge-cloud computing provides benefits in many use cases including distributed computation algorithms, where the processing is divided into a number of tasks that are executed in parallel on different equipment. An important relevant challenge is to allocate the appropriate resources to process the data that are continuously generated from user devices. The issue becomes more complicated when we take into account the variations in the volume of the generated data as a function of time. In this paper we present a resource allocation algorithm for distributed computation with emphasis on machine learning algorithms. We consider that the resource requirements vary with time in a semi-static way that exhibits some daily pattern. We distinguish between periodic (expected) variations that occur during the day, and sporadic variations due to unexpected events. We propose an Integer Linear Programming algorithm to allocate the periodic resource requirements. To handle the non-periodic requirements, we consider a suitable prediction algorithm coupled with a reconfiguration algorithm that allocates the predicted required resources. Our results indicate that our proposal outperforms traditional allocation algorithms in terms of resource utilization, monetary cost and achieved accuracy.**

***Index Terms*—cloud and edge computing, distributed computing, distributed machine learning, resource allocation, prediction, dynamic.**

I. Introduction/Motivation

The applications and services that process data generated at Internet of Things (IoT) devices and mobile phones continuously evolve. The resulting job processing requirements places significant burden on the edge and cloud resource management. As a result, significant research effort has been exerted to develop algorithms to select the resources that serve the processing and storage demands [1] [2] [3] [4].

Another factor that complicates the management challenge is the dynamicity and periodicity of the requirements. Internet traffic volume in general, and the human generated data in particular, follow a periodic pattern of daily fluctuations due to the respective human activities patterns. Thus, the resource requirements of the resulting generated jobs are not constant but exhibit a semi-static structure. Moreover, there are certain events (e.g., football matches) that affect the volume of data generated and processed. These events are to a certain extent predictable. To this end, a number of previous works have focused on resource demand prediction and resource allocation at the edge and cloud [5] [6] [7] [8] [9]. It is an interesting topic that combines the resource allocation challenge with the requirement of accurate prediction using artificial intelligence among others.

A significant sub-category of work addressing the offloading challenge is related to the problem of resource allocation for distributed computation algorithms over a continuous time horizon. In this case, a number of devices at the edge continuously produce (large amounts of) data that are offloaded at the edge and cloud resources. The processing of the algorithm is divided into a number of tasks that are executed distributedly on different equipment. In our previous work [10] we specifically studied resource allocation algorithms to solve the aforementioned challenge. In this paper we extend our previous work by considering the dynamicity and partial periodicity of the resource requirements. More specifically, we assume that the resource requirements are to a certain amount periodical (known or estimated) during the day. For example, consider a distributed ML training scenario in an Internet of Vehicles setting. The daily vehicle traffic volume is periodical in a large extent. We adopt a semi-static model, where the day is divided in a number of sub-periods during which the resource assignments are close to constant, except for some random fluctuations. We plan in advance the assignment of resources to the demands exhibiting a periodic pattern. Additionally, we consider that certain events during the day significantly alter the planned periodic allocations. For example, a major event that results in mass people gatherings, typically disrupts among others the vehicle traffic volume. We consider that these events are estimated with some accuracy using a suitable prediction algorithm as we will discuss in the following sections. We present a suitable algorithm that takes as input the predicted requirements, and adjusts the current resource allocation to accommodate the altered demands.

To the best of our knowledge the combination of a planning and a prediction algorithm for the allocation of network resources has not been considered before at least in the specific context of distributed computation algorithms. We present interesting insights on the interplay of various parameters, such as the trade-off between accuracy and monetary cost under the aforementioned scenario. We compare our approach to the alternative protocol of incremental updates in the resource assignment to meet the current demands. A scenario where the demands are highly dynamic can be a topic of future work.

II. Related Work

This work is generally related to the topic of computation offloading. More specifically it is related to the topic of resource allocation for distributed computation with prediction of the resource requirements. Computation offloading and Mobile Edge Computing [1] [2] [3] [4] is a vast research area. The challenge is to decide about the locations where a given set of tasks will be executed and to allocate the related resources. In some cases, the execution of a task can be divided and be partially offloaded. A special subcategory of computation offloading is distributed computation [11], where the jobs comprising a certain task are executed in parallel and in many distributed locations. A major application of distributed computation is distributed machine learning (DML) [12] [13]. There are different variations/architectures of DML, such as the parameter server(s) (where certain servers are used to average the model's weights and send the updated values to the computing nodes) and all-reduce where the coordination is distributed. A number of works have focused on the allocation of resources for DML. In our previous work [10], we developed resource allocation algorithms and examined interactions between accuracy, monetary cost and delay. Other approaches include the (computation and storage) resource allocation to maximize the distributed learning throughput [14] and the development of scheduling algorithms to minimize the completion time [15]. The aforementioned research generally does not consider the dynamic (periodic and non-periodic) nature of the tasks. A number of works have considered predicting the demands to more efficiently allocate the appropriate resources [5] [6] [7] [8] [9] in the context of computation offloading. Typically, an (artificial intelligence) algorithm is used to estimate the resources in a given future time interval. Then a related algorithm allocates the resources for that interval based on the received input. In a similar direction, in this work we significantly expand our previous work [10] in the context of distributed computation. We consider both periodic and non-periodic variations of the data volume generated by the user devices that feed the distributed algorithm. We employ a traffic predictor to estimate the unexpected changes in requirements. We then demonstrate the advantages of our approach over other approaches.

III. Problem Statement

We consider a number of devices at the edge that continuously produce data. The data processing is performed at the edge network close to the devices, and at a more distant cloud. The edge network includes a set of nodes N with finite capacity that can be used by the ML tasks. The cloud network has infinite resources. A set of devices make for an algorithmic job j that processes their data. The processing of job j is divided into a set of distributed ML tasks $T_j = \{t_{j1}, t_{j2}, ..., t_{je}\}$ that are executed in parallel at respective worker nodes. There is a time window Γ in which the devices generate and send the data for processing, and a subsequent time window where the data are processed. Each ML task needs certain computation and network and possibly aggregation (in the case of parameter servers) resources during each time window. We assume that the vector of resource requirements remains constant throughout certain periods during a 24h daily cycle due to known periodic variations. For example, we can assume that there are three 8-hour periods during which the requirements remain constant.

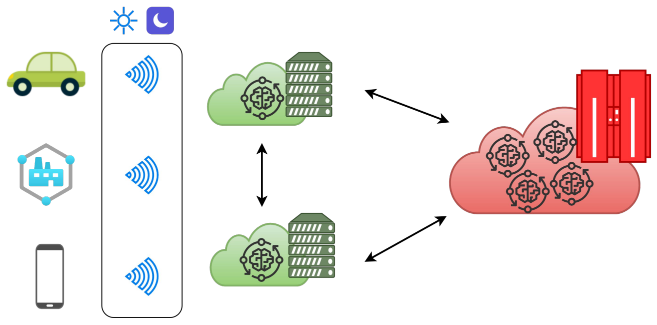

Fig. 1. The abstract architecture considered.

We denote: $R^{pjea} = [G^{pjea}, B^{pjea}, \Theta^{pjea}]$ where G, B, and Θ are parameters that reflect the amount of processing (in, e.g., Floating Point Operations - FLOP), number of bits communicated to the nodes and processing for weight aggregating purposes that each sample requires for the specific time period p, the ML task e of job (application) j and for a specific required computation accuracy a. Computation accuracy is a parameter that can be traded-off against resource requirements, when the resources are limited.

Additionally, we consider that during the day we can have temporary fluctuations in the resource requirements. These fluctuations could be due to major events that disrupt the normal city traffic or that attract too many people to a certain location. We denote with R_*^{pjea} the brief increase of the resource requirements that has to be met, where the indices p, e, j and a are as defined above.

To allocate the appropriate resources for the aforementioned scenarios, we employ three main algorithms:

- An Integer Linear Programming (ILP) planning algorithm. This algorithm is responsible for the resource allocation of the periodic requirements R^{pjea}.
- A traffic predictor that estimates temporary increases R_*^{pjea} in resource requirements.
- A reconfiguration algorithm that reconfigures the resource allocation according to the predictor's output.

IV. Resource Allocation Algorithms

A. ILP Algorithm

In this subsection we present the ILP algorithm. The goal of this resource allocation algorithm is to reserve the appropriate resources for the tasks, including the specific edge node where

TABLE I
NOTATION

Symbol	Description
J	Set of jobs
T_j	Set of tasks of job j
λ_{je}	Production rate of task je in samples/sec j
N	Set of node of edge network
R_n^G, R_n^B, R_n^Θ	Set of processing, b/w, aggregation resources of edge node n
$C_E^G, C_E^{bw}, C_C^G, C_C^{bw}$	Processing and b/w costs at the edge and cloud respectively
δ_c	Propagation delay of cloud
Δ_j	Acceptable prop. delay of job j
W	Weight to control optimization objective
A	Set of possible accuracies of ML jobs
a_j	An accuracy of a job j ranging from 0 to 1
a_j^{min}	The minimum acceptable accuracy of a job j
ξ_n^{pjea}	Binary variable equal to 1 if task je is served at node n, period p, accuracy a
ξ_c^{pjea}	Binary variable equal to 1 if task je is served at period p, accuracy a
k	The total monetary cost to serve all jobs
S	A set of jobs that must not migrate locations from one period to another
PC	A set of all possible combinations of successive periods p, p'
$m_{pp'}^{je}$	The migration cost of each task je from a period p to a period p'

each task will be processed), while minimizing certain objectives and satisfying all the constraints. A special parameter (migration cost) is used to control whether or not tasks can change execution location from one period to another. To avoid the related transferring (migration) overhead, tasks should be executed at a fixed place.

Inputs:
$N, R_n^G, R_n^B, R_n^\Theta, J, T_j, C_E^G, C_E^{bw}, C_C^G, C_C^{bw}, \delta_c, \Delta_j, W, A, A_j, \lambda_j$

Variables:
$\xi_n^{pjea}, \xi_c^{pjea}, k, a, m_{pp'}^{je}$

The symbolism in binary variable ξ_n^{pjeqa} means that there is a different variable for every different time period p, node n, job j, task e, and for every different accuracy level α. The symbol k represents the total monetary cost to serve all jobs. It is aimed to be minimized at the objective. Symbol a represents the mean accuracy of all the tasks to be served. It is aimed to be maximized at the objective.

Objective:

We have a multi-criterion optimization problem, as the objective is to minimize the total cost to serve the jobs, minimize the migration cost (tasks moving from one location to another) and maximize the accuracy.

The relative importance of each individual objective (that transforms each objective in a monetary cost, for example) is controlled by three respective weights w_1, w_2, w_3 with $w_1 + w_2 + w_3 = 1$. The cost of each job depends on the amount of processing and b/w and whether it is served at the edge or the cloud:

$$\min\left(w_1 k - w_2 a + w_3 \sum m_{pp'}^{je}\right) \quad (1)$$

Subject to:

- The cost to serve all jobs consists of the sum of the edge and cloud bandwidth (b/w) plus the edge and cloud processing cost for all the task jobs, for all the accuracy options and for all the periods:

$$k = \sum_j \sum_{t_{je}} \sum_a \sum_p \Big(\sum_n \xi_n^{pjea} \lambda_{je} (C_E^{bw} B^{pjea} + C_E^G G^{pjea}) + \xi_c^{pjea} \lambda_{je} (C_C^{bw} B^{pjea} + C_C^G G^{pjea})\Big) \quad (2)$$

- The mean accuracy of all tasks is defined as:

$$a = \sum_j \sum_{t_{je}} \Big(\sum_n \xi_n^{pjea} \alpha_j + \xi_c^{pjea} \alpha_j\Big) \quad (3)$$

- Each task of a job should be served once with one accuracy option, at the edge or at the cloud $\forall j, \forall t_{je} \forall p$:

$$\sum_{n\in N} \sum_a \xi_n^{pjea} + \sum_a \xi_c^{pjea} = 1 \quad (4)$$

- Edge capacity constraints:

$$\forall n \in N : \sum_j \sum_{t_{je}} \sum_a \sum_p \xi_n^{pjea} G^{pjea} \lambda_{je} \le R_n^G \quad (5)$$

$$\forall n \in N : \sum_j \sum_{t_{je}} \sum_a \sum_p \xi_n^{pjea} B^{pjea} \lambda_{je} \le R_n^B \quad (6)$$

$$\forall n \in N : \sum_j \sum_{t_{je}} \sum_a \sum_p \xi_n^{pjea} \Theta^{pjea} \lambda_{je} \le R_n^\Theta \quad (7)$$

- Cloud delay constraints for relevant jobs:

$$\forall j_i, \forall t_{je}, \forall p : \xi_c^{pjea} \delta_j \le \Delta_c \quad (8)$$

- The minimum required accuracy should be respected: $\forall j, \forall t_{je}, \forall p$:

$$\sum_{n\in N} \sum_a \xi_n^{pjea} \alpha_j + \sum_a \xi_c^{pjea} \alpha_j \ge \alpha_j^{min} \quad (9)$$

- Migration cost: the cost when a task moves from one location to the other. Variable $mc_{jepp'}$ increases in value if a job stays in the same place in subsequent periods.

$$\forall j, \forall t_{je} \notin S, \forall a, \forall p, p' \in PC : m_{pp'}^{je} >= \xi_n^{pjea} - \xi_n^{p'jea} \quad (10)$$

$$\forall j, \forall t_{je} \notin S, \forall a, \forall p, p' \in PC : m_{pp'}^{je} >= \xi_n^{p'jea} - \xi_n^{pjea} \quad (11)$$

The solution of the algorithm is the binary values of all the variables. For example, when $p = 1, n = 1, j = 1, e = 1, q = 1, a = 1$, and $\xi_n^{pjea} = 1$ then the first task of the first job is served at the first node of the edge network using the first GPU model, and the first choice of the available accuracy options. Even though the ILP algorithms are generally computationally intensive, the specific formulation can provide promptly a solution in realistic scenarios as we will demonstrate in the simulation section. Moreover, certain heuristic algorithms as shown in our previous work [10] can be used to accelerate the execution time.

B. Traffic prediction algorithm

The traffic predictor takes as input some previously observed data of either the sample production rate λ_{je} or the required resources per task R^{pjea}. Without having to be specific, as we are not interested in the details of the specific algorithm used, we assume that we employ some ML or other algorithm to predict the required resources, providing as output the predicted vector of required resources during future time steps $\lambda_{je}R^{pjea}_{*1}$, $\lambda_{je}R^{pjea}_{*2}$. The goal of a predictor is to find and fit a function to the available past data in order to estimate future data. An essential part of the problem is to select (mainly through trial and error) the suitable time estimation period (or equivalent) amount of data required to have accurate enough predictions. In our case we are interested in very short term prediction. The input could be three or four time steps, with one time step corresponding to 15 minutes, and the output could be given for two or three time steps. There is a large number of previous works related to this subject ranging from autoregression algorithms to specific ML and neural network architectures [16] [17] [18]. Autoregression uses a regression model that combines a number of historical values to estimate future values. One suitable model could be (weighted) linear regression. SVMs as well as neural networks (e.g. Long short-term memory or convolutional neural networks) have the ability to fit more complex functions to the data. An example architecture in the case of a predictor based on Convolutional Neural networks consists of a convolutional hidden layer and a pooling layer. Then a flatten layer is used to feed the subsequent dense fully connected layer that provides the output. In our case any algorithm that provides an accurate output for a small number of future time steps can be used to estimate the resource requirements. The prediction is given to the heuristic algorithm responsible for the reconfiguration of the network.

C. Heuristic resource allocation algorithm

The heuristic reconfiguration algorithm takes as input the estimated resource requirements for each task, and examines one task at a time. If the estimated demands cannot be served by the allocated resources, then the algorithm considers reconfiguring the appropriate resources. The algorithm first checks whether there are adequate free resources at the original location of each task that can satisfy the new requirements. If there are, then it allocates the appropriate resources. If not, there are two options: either move the entire task to a new location that can satisfy the requirements, or keep unchanged the previous configuration. All the decisions depend on the respective acceptable reconfiguration cost and the Service Level Agreement (SLA) constraints. More specifically, the reconfiguration cost can be defined as the number or the percentage increase of the processing resources, the additional monetary cost and the location of the possible reconfigurated task. Regarding the SLA, certain tasks could require they should not change location so that their execution will not be interrupted. In other cases the most appropriate decision could be to always match their requested resources.

After the requirements have returned to their normal planned values, the algorithm releases the additional resources. If the task has changed location, the algorithm preserves the new location (unless there is another policy) so that future increases can be easily accommodated. Detailed complexity analysis of the ILP and heuristics is ommitted due to space limitations, but a relevant discussion can be found in [10].

V. Simulations

In this section we present the results of a number of simulation experiments than can be an application of a distributed ML image training scenario. We assumed a 10-node edge network with finite resources. Each edge node has 5 racks, 1 rack has 10 servers, and 1 server has 4 low cost and 2 higher cost GPUs, for a total of 200 low cost and 100 higher cost GPUs per node. Each edge node has 10 Tbps incoming bandwidth and 6000 CPU physical cores (that could correspond to approximately 100 CPUs). We also assumed a cloud network with infinite resources. Without loss of generality, we considered a scenario A consisting of a total of 400 training image recognition ML jobs and a scenario B consisting of 600 jobs. The size B_j of each data sample (i.e., image in these experiments) of a job j is chosen uniformly from the following set of values: [0.4, 0.8, 1.2, 1.6, 2, 2.4] MBs / sample. The available GPU models q were NVIDIA DGX-1 with 1 (low cost) or 8 (higher cost) GPU V100 16G. The respective cost of these GPUs at the cloud is \$2.08/hour and \$16.7/hour [19] (p3.2xlarge or p3.16xlarge). The b/w cost to transfer data to the cloud is $0.01/GB$ [19]. The required b/w of each task t_{je} is derived by multiplying the generation rate l_{je} of samples/sec by the size B_j in MBs/sample and by the duration of the time window Γ in seconds. This figure equals to the amount of data that have to be transferred within one period. Similarly to our previous work [10], we considered that the edge's b/w costs are approximately 0.1 times those of the cloud's (as the data has to be transferred over shorter distances), and the edge processing costs are 1.5 times the cloud processing costs (as edge resources are more costly to host and operate). Each of the 100 training jobs consists of either 3, 4,…, 7 ML image recognition tasks, uniformly distributed. The sum of the sample production rates of the devices providing data to task t_{je} is 15 samples/sec. We consider that the duration of the training period is $\Gamma = 30$ seconds, yielding $S_{je} = 450$ total samples processed in each period. We also assume that each ML task could be served with two different accuracies $A = a^{good}, a^{low}$. The number of NVIDIA GPU units U^{pjeqa} required per period, per task and per time window Γ, is calculated as in [10] based on the number of samples S_{je} of each task t_{je} processed per time window and the accuracy a_j. The computational performance Pi^a_q of 1 GPU V100

16G unit for image recognition training according to [20] is $Pi^a_q = 166$ or 566 samples/sec for single-precision floating-point math – FP32 or mixed precision accuracy respectively. The performance of a resource type q consisting of 8 GPU V100 units is $Pi^a_q = 1210$ or 4160 samples/sec, respectively. We assumed 100 number of epochs. Similar results can be drawn for different epochs and processing costs as explained in [10]. We consider 3 periods within a day for which we will

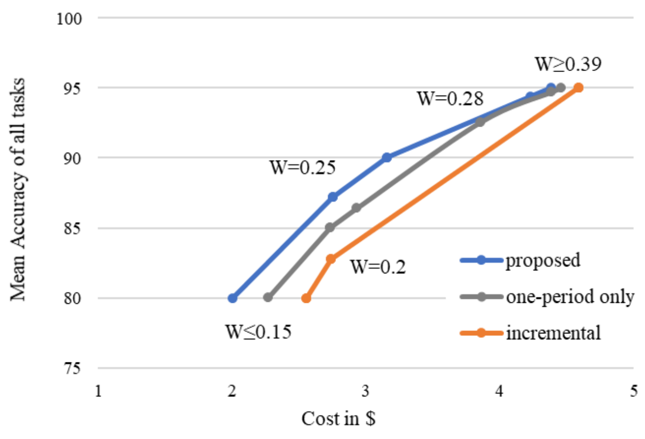

Fig. 2. Accuracy vs monetary cost comparison (Scenario A)

use the ILP to plan the allocation of resources. Based on [22], we assume that traffic volume ratios for the respective periods correspond to: 0.25, 1, 0.5. So, the second period volume is four times bigger than the first period, and the third period volume is twice than that of the first period. We assume that these ratios translate to respective ratios for the number of ML tasks. The simulations were executed in Python in a quad-core CPU. The running time of the algorithm for these parameters was approximately 2 seconds.

Regarding the unexpected fluctuations, we assume certain events can increase the normal traffic of the second period by 20% percent which is a realistic estimate [22]. We assumed that a traffic predictor as in [18] is used to estimate the differences. We compare to an alternative algorithm, where there is no planning to serve the demands over the three periods (named ”incremental”) and to predict the unexpected fluctuations. Instead, this algorithm serves (greedily) each demand one-by-one at the edge or the cloud by trying to minimize its individual cost objective. Whenever there is an increase at the demands, the algorithm serves one-by-one the related demands. Another alternative (named ”one-period”) assumes one planning period where the requirements of the most demanding period are planned in advance (as in [10]). The rest of the requirements are again served one-by-one by the aforementioned incremental algorithm.

In Figs. 2 and 3 we compare the accuracy vs monetary cost of our proposal compared to the alternatives for scenarios A and B, respectively. For simplicity we assumed only one weight W, for the monetary cost optimization, while the accuracy weight equals to $1 - W$ and there is no migration. For this set of values, we plot the resulting mean accuracy of all the jobs and their total monetary cost. The choice of the exact values of W depends on the parameters of the problem. For larger or smaller instances, different set of values may be required to acquire the respective accuracy to monetary cost relationship. For example, for smaller amount of jobs the total monetary cost could be smaller. Therefore, a larger value of W may be necessary to have the same accuracy. Generally,

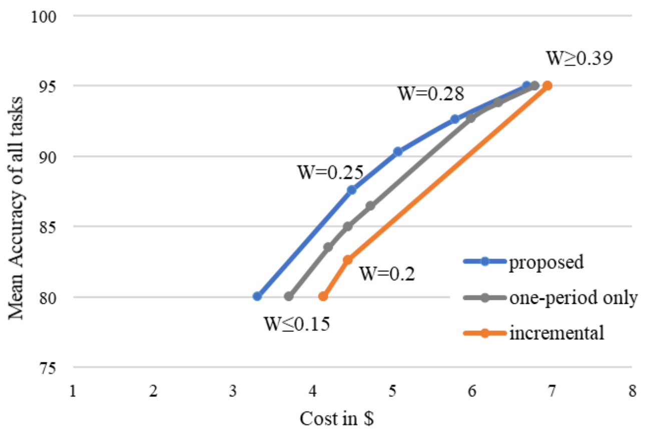

Fig. 3. Accuracy vs monetary cost comparison (Scenario B)

using this figure, the trade-off between accuracy and monetary cost can be deduced for a given set of parameters and jobs.

We observe that the proposed algorithm achieves the best accuracy coupled with the lowest monetary cost in both scenarios. For 80% accuracy, the related costs in scenario A are \$2, \$2.27 and \$2.6 for the proposed, one-period and traditional algorithm respectively. So, the ILP algorithm can serve the tasks with the same accuracy, but at 9.5% and 21.7% lower cost respectively. Moreover, for roughly the same cost (\$2.75) the proposed algorithm achieves 87.5% mean accuracy, compared to the 84.9% and 82.7% of the other algorithms. The reason that the algorithms that employ (even one period) planning perform better, is that they have a complete view of all the demands at the beginning of the allocation. Therefore, they can make optimized placing decisions by taking into account the best overall objective cost. In contrast, the heuristic algorithms that serve one by one the tasks take into account the individual objective cost of each task, which could lead to sub-optimal total objective cost. The proposed algorithm uses planning for all three periods, so it has the best performance among the examined algorithms. In the case of the best accuracies, the difference in monetary costs are less prominent (1.8% and 4.36% lower cost than the one-period and the incremental algorithms, respectively). The reason for this is that the best accuracy requires the most expensive allocation decisions. This leaves little room for cost improvements by the placement optimization of the jobs. In scenario B, the savings of our proposal are somewhat larger, by approximately 2%. The reason is that the additional number of jobs create more allocation possibilities, and more opportunities for the planning and the subsequent reconfiguration algorithm to better allocate the resources. In conclusion, the proposed algorithm provides benefits under a variety of loads.

In Figs. 4 and 5, we assumed a common target accuracy of 80% for all the compared algorithms. The objective is to serve the jobs at the lowest possible monetary cost. We examine the GPU utilization of the three algorithms. We notice that the proposed algorithm in scenario A requires 12.6% and 6.4% less GPU units to achieve the same results as the traditional incremental and the one-period algorithms respectively. Again, in scenario B the savings are slightly higher for the same aforementioned reasons. These results also translate to less required energy of the resources to produce the same output. An energy model as the one in [23] can be used to acquire the respective results and can be the topic of a future work.

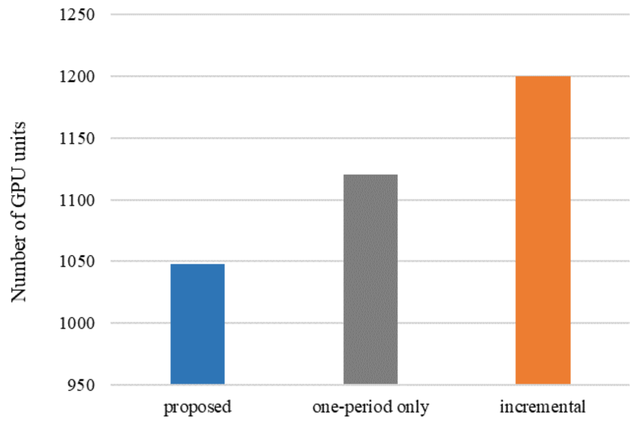

Fig. 4. GPU unit utilization comparison (Scenario A)

VI. Conclusions

In this work we investigated the problem of allocating resources for distributed computation applications in the context of (non) periodic demands. We presented a planning algorithm that serves the periodic semi-static demands. We also proposed a traffic predictor and a reconfiguration algorithm that serves the unexpected demands. We performed a number of simulation experiments. The results show that our proposal has a number of advantages when compared to the examined alternatives: a reconfiguration algorithm to serve one-by-one each demand, or a one period planning coupled with a reconfiguration algorithm to serve one-by-one the fluctuations. More specifically, our proposal can achieve approximately 20% and 10% reduction in monetary costs over the first and second alternative respectively.

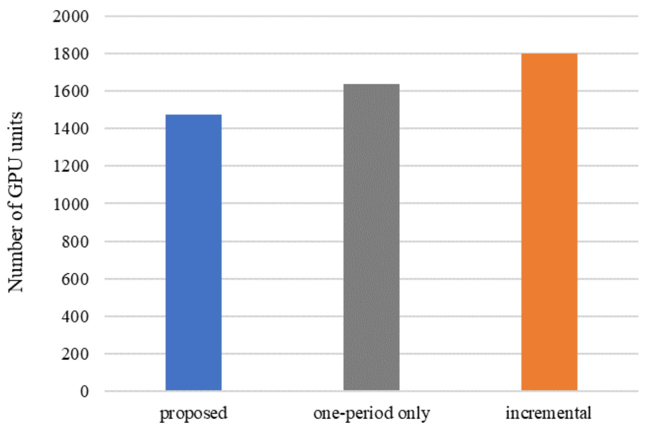

Fig. 5. GPU unit utilization comparison (Scenario B)

References

[1] P. Mach, Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," IEEE Comm. Surv. and Tutor., 19(3), 2017.
[2] 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.
[3] I. Sarrigiannis, et al., "Cost-Aware Placement and Enhanced Lifecycle Management of Service Function Chains in a Multidomain 5G Architecture," IEEE Trans. on Network and Serv. Management, 19(4), 2022.
[4] T. M. Ayenew, et al., "Demand-Aware Cooperative Content Caching in 5G/6G Networks With MEC-Enabled Edges," IEEE Networking Letters, 4(3), 2022.
[5] W. C. Chien, et al., "Dynamic resource prediction and allocation in C-RAN with edge artificial intelligence," IEEE Transactions on Industrial Informatics, 15(7), 2019.
[6] B. Bao, et al., "Resource allocation with edge-cloud collaborative traffic prediction in integrated radio and optical networks," IEEE Access, 11, 2023.
[7] X. Chen, et al., "Resource allocation for cloud-based software services using prediction-enabled feedback control with reinforcement learning," IEEE Transactions on Cloud Computing, 10(2), 2020.
[8] M. Chen, et al., "Intelligent traffic adaptive resource allocation for edge computing-based 5G networks," IEEE transactions on cognitive communications and networking, 6(2), 2019.
[9] L. A. Grieco, et al., "Ad-hoc, mobile, and wireless networks," Proceedings of the 19th international conference on ad-hoc networks and wireless, ADHOC-NOW, 2020.
[10] I. Sartzetakis, et al., "Edge/Cloud Infinite-time Horizon Resource Allocation for Distributed Machine Learning and General Tasks," IEEE Transactions on Network and Service Management, 21(1), 2024.
[11] D. Bertsekas, J. Tsitsiklis, "Parallel and distributed computation: numerical methods," Athena Scientific, 2015.
[12] J. Dean, et al., "Large scale distributed deep networks," in NIPS, 2012.
[13] T. Ben-Nun, T. Hoefler, "Demystifying Parallel and Distributed Deep Learning: An In-depth Concurrency Analysis," ACM Comput. Surv. 52, 4, Article 65, 2019.
[14] M. Chen, et al., "Joint Data Collection and Resource Allocation for Distributed Machine Learning at the Edge," IEEE Transactions on Mobile Computing, 21(8), 2020.
[15] R. Zhou, et al., "Online Scheduling Algorithm for Heterogeneous Distributed Machine Learning Jobs," IEEE Transactions on Cloud Computing, 11(2), 2023.
[16] A. S. Weigend, "Time series prediction: forecasting the future and understanding the past," Routledge, 2018.
[17] N. I. Sapankevych, S. Ravi, "Time series prediction using support vector machines: a survey," IEEE Computational Intellig. Mag., 4(2), 2009.
[18] Y. Hua, et al., "Deep learning with long short-term memory for time series prediction," IEEE Comm. Mag., 57(6), 114-119, 2019.
[19] "Amazon ec2 pricing," available online: https://aws.amazon.com/ec2/instance-types/p3/
[20] "Nvidia resnext performance," available online: https://ngc.nvidia.com/catalog/resources/nvidia:resnext_for_tensorflow/performance
[21] P. Mattson, et al., "MLPerf Training Benchmark," ArXiv abs/1910.01500, 2020.
[22] S. Batterman, et al., "Temporal variation of traffic on highways and the development of accurate temporal allocation factors for air pollution analyses," Atmospheric environment 107, 2015.
[23] G. Drainakis, et al., "On the Distribution of ML Workloads to the Network Edge and Beyond," in IEEE INFOCOM, 2021.