Fog Computing Task Offloading Optimization based on Deep Reinforcement Learning

Luís Ferreira

***Abstract*—Industry and society are constantly changing, and the tendency is to evolve into an ever more technology oriented system. The ubiquity of Internet of Things plays a big part in the technological future, connecting the surroundings through sensors and actuators to the Internet. Cloud Computing comes as a computing paradigm that can also leverage Internet of Things. However, it has its shortcomings regarding communication bandwidth and latency. Fog Computing comes as an extension of Cloud Computing in order to tackle the aforementioned problems, shifting partial computation closer to the edge and managing communications to the Cloud. To make this technology viable, it is first necessary to manage the resources present in the Fog. A proposed concept by academia is a choreography styled resource management, where each Fog node makes its own decisions based on requests from the end devices in a group cooperation to improve the overall performance. Therefore, it is necessary to implement an algorithm that makes the decisions with awareness of other nodes. Studies have been made to introduce Deep Reinforcement Learning as a viable decision maker. To create groundwork for standardized further studies, a simulator based on an open source Deep Reinforcement Learning tool is developed. Then, a Proximal Policy Optimization with an Advantage Actor Critic architecture algorithm is implemented and tested in a set of cases in the developed tool. In the developed scenarios, the choreography Proximal Policy Optimization algorithm demonstrated the capability to surpass the non intelligent baselines algorithms when optimizing based on a common reward. However, some limitations are found and should be addressed in future works.**

***Index Terms*—Fog Computing, Resource Management, Choreography, Reinforcement Learning, Deep Reinforcement Learning, Proximal Policy Optimization.**

I. Introduction

Cloud Computing (CC) is a computing paradigm where the resources utilized by a user are located on the cloud. This seemingly infinite computing power allows the user to make highly complex computational tasks with low effort. Internet of Things (IoT) takes great advantage in the use of CC, since it means smaller connected devices are still able to perform intricate operations as long as there is connection. However, CC has some shortcomings. First, some tasks might have a strict deadline that cannot be held because of a large unpredictable latency in the connection to the cloud. Second, with the growth in number of IoT devices and all the data they generate, the bandwidth required in communication becomes too large for a centralized process.

In order to tackle the constraints found in CC, many computing paradigms have been proposed. A study encompassing the most notorious computing paradigms is done in [1]. Among them, Mist Computing, Edge Computing, and Fog Computing (FC) take relevance. Mist Computing refers to a computing paradigm at the end devices, i.e. at IoT devices level. Edge Computing proposes a computing location near the end devices, as far as one hop, in small data centers at the edge. Although its definition depends on the literature, here FC is defined as a concept that seeks to complement all the computing paradigms placed between end devices and the cloud, to work as the network link with computing and storage capabilities. Due to its versatility and wide use-case range, FC, proposed by both industry and academia, is widely accepted as a new computing paradigm [2]. The FC proposal architecture was designed to support the real-time and bandwidth constraints of the ever-growing use of IoT applications.

It can be said that FC is a mid level architecture meant to bridge end-users with the cloud. It takes some toll from the network links to the cloud, helping to manage the resources allocated, possibly improving Quality of Service (QoS) in real-time applications and improving overall efficiency. Rather than replacing the current CC paradigm, it exists to complement it.

To achieve and deploy an FC network, it is necessary to solve the problem of resource management, in either a centralized or distributed system approach, in order to achieve optimal delay in task execution within physical constraints. This management is focused in computing, storage, and communication resources. It will translate into a problem of task offloading, i.e. allocate atomic tasks to another computing device rather than the current device. A problem of caching, i.e. defining local temporary storage by frequency of request in applications to reduce content delivery time. Plus, a problem of bandwidth management. All problems to be approached with multiple environment associated constraints.

A. Objective

As one of the problems presented in FC management, computing task offloading is the main focus of this work. Each of the aforementioned atomic computing tasks can be seen as a module in a single application. The more an application is divided into smaller modules, the greater the opportunity for distributed computing. However, it is necessary to manage it and associate the respective modules with a computing device, i.e. placement of a task in either an Edge, Fog, or Cloud node.

The traditional optimization techniques, often reaching optimal placement policies, lack in scalability in terms of computing power and require previous knowledge. On the other hand, the machine learning methods allow for a more flexible and model free approach, which can manage to converge into

an optimal policy as well. This way it is possible to translate from an offline to an online optimization problem.

In this work, the main goal is to create an optimized FC choreography algorithm based on the previous work done in the area, specifically by Baek et al. [3]. Improved Deep Reinforcement Learning (DRL) techniques are used to make a scalable and efficient choreography in this ubiquitous and heterogeneous environment.

B. Document organization

The outline of the document structured as follows: Section II describes the simulated FC environment; Section III explains the implementation of the used DRL algorithms; Section IV contains the experimental results in various test cases; Section V concludes the work and proposes further research.

II. Simulation Environment

An FC environment has heterogeneous characteristics: each device and connection may be different in its specifications. Entities can be divided into three layers: Cloud devices, Fog devices, and end devices. Each of these devices might be grouped or not, from an Software Defined Network (SDN) point of view, and considered a node, i.e. Cloud node, Fog node, and End node. Each node is somehow connected to the network through a network interface, translated to physical form in either optical cable, coaxial cable, or cellular network, among other possibilities, including multi-hop routing segments.

The work in [3] will be used as reference for the FC environment modelling.

The FC environment is set in an SDN point of view, where the data plane processes and forwards task while the control plane is in charge of decision-making. Furthermore, it is split in logical slices, which provides an extra level of abstraction allowing priority allocation or concise physical resource division per layer, among other possible operations in a sliced environment.

This problem is defined as a problem of choreography, where each node makes its own decisions. However, as nodes share their local rewards, a joint reward is used to improve the system as a whole, which includes a hint of orchestration.

For this Section notation will follow the Table I.

Each node is characterized by its resources $U_i = (U_i^c, U_i^m)$, CPU and memory respectively, transmission resources bandwidth BW_i and transmission power P_i. From the control layer point of view, the node buffer is divided in K slices, each of this slice partitions of the buffer is seen as a buffer itself with a maximum queue size of $\bar{b}_{i,k}$.

Different task types are assigned at different slices. In each slice a task is characterized by its packet size T_k, a delay constraint D_k^{max}, and resource requirements for processing the task $L_k = (L_k^c, L_k^m)$, CPU processing density and necessary memory respectively.

It is assumed that each allocation unit, η_i^c and η_i^m respectively for CPU and memory, is indivisible. Furthermore, it is assumed that a CPU allocation unit is a single computing core with frequency η_i^c. Another assumption is that each task of size T_k and computing density L_k^c is sequential, i.e. it can not be split to compute in parallel cores. With those assumptions, each task in slice k is assigned one core and the necessary memory when being processed:

$$L_k = \left(\eta_i^c, \left\lceil \frac{L_k^m}{\eta_i^m} \right\rceil\right) \tag{1}$$

TABLE I
Environment description notation

Notation	Unit	Description
I		The number of Fog nodes
K		The number of logical slices in the Fog
(β_1, β_2)		The path loss constant and exponent
N_0	dBm/Hz	Thermal noise density
P_i	dBm	Transmission power of fog node i
BW_i	MHz	Transmission bandwidth attributed to fog node i
U_i^c	GHz	The total amount of CPU resources at fog node i
U_i^m	MB	The total amount of memory resources at fog node i
η_i^c	GHz	The allocation unit of CPU resources at fog node i
η_i^m	MB	The allocation unit of memory resources at fog node i
r_i^c		The number of available units of CPU resources at fog node i
r_i^m		The number of available units of memory resources at fog node i
T_k	bit	Packet size of a task in slice k
D_k^{max}	ms	Maximum delay budget of a task in slice k
L_k^c	cycle/bit	CPU processing density demanded for a task in slice k
L_k^m	MB	Memory size demanded demanded for a task in slice k
$a_{i,k}$		A Boolean variable that indicates whether or not a task from slice k arrived at fog node i during a certain time step
$\bar{b}_{i,k}$		The maximum number of tasks from slice k in the buffer of fog node i
$b_{i,k}$		The number of tasks from slice k in the buffer of fog node i limited by $\bar{b}_{i,k}$
$b_{i,k}^e$		The number of tasks from slice k in the buffer of node i which have resources allocated to its execution limited by $b_{i,k}$
$f_{i,k}$		The offloading decision made by fog node i to where the task of slice k will be processed
$w_{i,k}$		The resource allocation decision done by fog node i that indicates how many tasks of slice k will have resources allocated to them
$\mu_{i,k}$		The service rate of the node i for slice k, calculated as the number of tasks leaving the buffer $b_{i,k}$ per time step
$\lambda_{i,k}$		The arrival rate at node i for slice k

With the assigned resources stated in Equation 1, a task from slice k processing in node i has a processing delay calculated with the following expression:

$$D^p_{i,k} = \frac{T_k \cdot L^c_k}{\eta^c_i}. \tag{2}$$

The communication is modelled in a full duplex fashion, with each node splitting its assigned bandwidth for the number of concurrent tasks being offloaded to another node. Another assumption is that it is a far field approach. To calculate the bit rate between node i and node j, offloading n tasks, the Shannon-Hartley theorem is used [4]:

$$v_{i,j,n} = BW_{i,n} \cdot log_2 \left(1 + \frac{\beta_1 {d_{i,j}}^{-\beta_2} \cdot P_i}{BW_{i,n} \cdot N_0}\right), \tag{3}$$

where $d_{i,j}$ is the distance between the two nodes, β_1 and β_2 are the path loss constant and exponent respectively, $BW_{i,n}$ is the bandwidth of node i evenly divided by the n concurrent tasks, P_i is the transmission power and N_0 is the thermal noise density.

With Equation 3, a transmission delay from node i to node j, with $i \neq j$, can be calculated as:

$$D^t_{i,j,k,n} = \frac{T_k}{v_{i,j,n}}, \tag{4}$$

with T_k as the packet size of a task in slice k.

Another delay that has to be taken into account for computation of the total delay is the waiting time in the queue, given by the following expression:

$$D^q_{i,k} = \frac{b_{i,k}}{\mu_{i,k}}, \tag{5}$$

with $\mu_{i,k}$ as the service rate of node i for the slice k.

The total delay of a task from slice k processing in the Fog can be computed, including transmission delay ($D^t_{i,j,k,n} \neq 0$) if there is an offloading ($i \neq j$), as the sum of all delays as follows:

$$D_{i,j,k,n} = D^p_{i,k} + D^q_{i,k} + D^t_{i,j,k,n}. \tag{6}$$

Limited by their delay constraints D^{max}_k, when a task reaches its maximum delay budget it is discarded, since it has lost its data value or necessity to be processed and considered unsuccessful. This is only done when a CPU unit and the necessary memory units can be assigned to the task in order to verify the delay constraint. Another limitation imposed is that, when a task arrives from the end device, the decision to offload or not the task is done in that time step and the task cannot be offloaded to a third node.

The time step is considered small enough for the event of a task arriving at a node i from a slice k to be unique. So, at each time step t, the task arrives following a Bernoulli distribution with the parameter $\lambda_{i,k}$.

A. Discrete Event Simulator

To simulate the environment previously described, a Discrete Event Simulator is used. A unique event queue sorts posted events and runs each event that modifies the representation of the physical system changing its state. Eight events were implemented as a subclass of a generic sortable event of the event queue:

- **Set Arrivals** - is an event that triggers a Task arrival from an end device based on a Bernoulli distribution with the parameter $\lambda_{i,k}$ for each node i and slice k. This event is called once in the beginning of the simulation and recursively posts itself in the queue for the next time step, it simulates the end devices requests.
- **Task Arrival (i, k, task)** - is an event that receives a task from slice k on node i. This event is called either from an Offload task event or a Set arrivals event.
- **Start Processing (i, k, w)** - is an event that sets $w_{i,k}$ tasks from slice k to process in node i, occupying necessary resources L_k as described in Equation 1; it posts a Discard task event if any attempted task exceeded its delay budget D^{max}_k or the Stop processing event if the task completes its processing within time constraint $D_{i,j,k,n} < D^{max}_k$. This event is posted from the control plane, made by the decision making algorithm.
- **Stop Processing (i, k, task)** - is an event that stops the processing of a specific task from the slice k in the node i, freeing all the taken resources L_k. This event is posted by the Start processing event.
- **Discard Task (i, k, task)** - is an event that discards a specific task from a slice k that attempted processing or is in the middle of processing in the node i, that has exceeded its delay budged D^{max}_k, returning all assigned resources L_k to the node pool of available resources. This event is called by the Start processing event.
- **Offload Task (i, j, k, n)** - is an event that offloads the task from a slice k that arrived in this time step in the node i to a node j, where $i \neq j$. It will only offload a recent arrival, since the decision of where a task will be processed, either locally or remotely, is done when the task arrives. To offload a task, this event splits the currently available bandwidth with the concurrent offloads n from other slices, calculates an arriving time and triggers a Task arrival event for each offloaded task. It also triggers a Start transmitting event for each offloaded task that will use the portion of the bandwidth to send the task. This event is called from the control plane, made by the decision making algorithm.
- **Start Transmitting (i)** - is an event that starts a transmission of a specific task in node i, occupying the given bandwidth $BW_{i,n}$, and posting the Stop transmitting event when the transmission ends. This event is posted by the Offload task event.
- **Stop Transmitting (i)** - is an event that stops a transmission in node i, freeing the previously occupied bandwidth $BW_{i,n}$ for a specific task. This event is posted by the Start

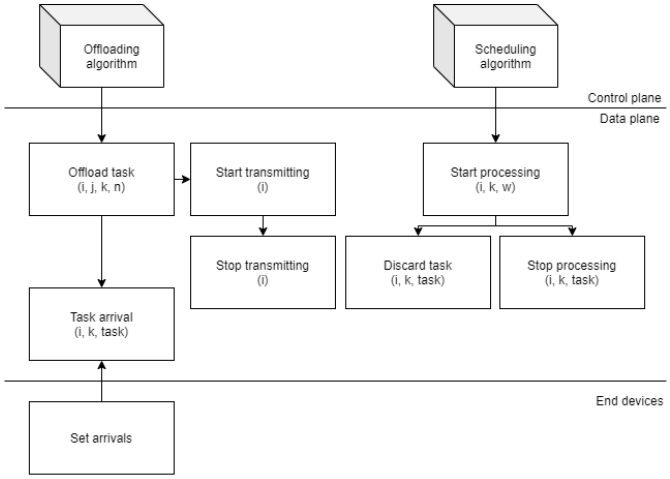

Fig. 1. Flow structure of the discrete events

transmitting event.

An illustration of the events flow is provided in Figure 1.

B. OpenAI Gym - Fog_env

In order to make a FC environment with an Reinforcement Learning (RL) approach, the OpenAI Gym base class *Env*, provided by OpenAI under the MIT Licence, is used as a super class of the implemented FC environment Fog_env. To do that it is first necessary to translate the environment to an Markov Decision Process (MDP), i.e. define a state, an action, a probability of transition and a reward function. However, each node only knows its own information, so the MDP is in fact the sum of the Partially Observed Markov Decision Process (POMDP) from each node. The local observation of the node in a time step is described, similar to the one in [3], as follows:

$$O_i(t) = (A_i(t), B_i(t), B_i^e(t), R_i(t)), \tag{7}$$

where $A_i(t)$ is the vector of $a_{i,k}(t)$, the boolean that indicates a task arrival from an end device, $B_i(t)$ the vector of $b_{i,k}(t)$, the number of tasks in the buffer for this time step t in each slice k, $B_i^e(t)$ the vector of $b_{i,k}^e(t)$, the number of tasks under processing in the buffer for this time step t in each slice k, and $R_i(t) = (r_i^c(t), r_i^m(t))$ are the available resource units from the resource pool in the Fog node i.

The action a node can take based on his observation is:

$$X_i(t) = (F_i(t), W_i(t)), \tag{8}$$

where $F_i(t)$ is the vector of $f_{i,k}(t)$, the decision to where the task that arrived at the time step t from an end device in node i from slice k will be processed, and $W_i(t)$ is the vector of $w_{i,k}(t)$, the number of tasks from slice k to be assigned resources in node i. The values of $f_{i,k}(t)$ are either $f_{i,k}(t) = 0$ if no task arrived, or $f_{i,k}(t) = j$ given a task from an end device. If the decision is to offload then $i \neq j$ else the task will be processed locally if $i = j$.

However, there are limitations in the action chosen by the node at a specific time step. These limitations, which can be implemented by the environment or by any action clipping before applying them to the environment, are the following:

$$f_{i,k}(t) = 0, if a_{i,k} = 0, \forall k \in K_i, \forall i \in I, \tag{9}$$

$$w_{i,k}(t) \leq b_{i,k}(t) - b_{i,k}^e(t), \forall k \in K_i, \forall i \in I, \tag{10}$$

$$\sum_{k \in K_i} w_{i,k}(t) \leq r_i^c, \forall i \in I, \tag{11}$$

$$\sum_{k \in K_i} w_{i,k}(t) \cdot \left\lceil \frac{L_k^m}{\eta_i^m} \right\rceil \leq r_i^m, \forall \in I, \tag{12}$$

where Equation 9 states that the node cannot make a decision on where to process a task that did not arrive, Equation 10 limits the number of tasks to assign resources to the number of idle tasks and Equations 11 and 12 limit the number of requested tasks depending on the free memory.

Lastly the reward of the POMDP is decided as an average of probable completion reward for each task arriving at the different slices, as modelled in [3],

$$\psi_i(O_i(t), X_i(t)) = \frac{1}{K} \cdot \sum_{k \in K_i} a_{i,k}(t) \cdot (S_{i,k} - O_{i,k}), \tag{13}$$

$$S_{i,k} = (-1)^{\mathbb{1}(D_k^{max} \leq D_{i,k})}, \tag{14}$$

$$O_{i,k} = \xi_k \cdot \mathbb{1}(b_{f_{i,k},k}(t + D_{i,k}^t(t)) \geq \bar{b}_{f_{i,k},k}), \tag{15}$$

where the first term (14) of the reward in (13) estimates whether the task will be completed successfully or it will exceed its time budget D_k^{max}, in which $D_{i,k}$ indicates the total delay estimate made by Equation 6, and the second term (15) with the constant weight ξ_k, as a tunable parameter, negatively impacts the reward if the task is expected to overflow the node $f_{i,k}$ assigned to process the task. The buffer size evaluation $b_{f_{i,k},k}(t + D_{i,k}^t(t)$ is done after the transmission $D_{i,k}^t(t)$ occurred for the offline optimization algorithms. An estimation $\hat{b}_{f_{i,k},k}(t + D_{i,k}^t(t)$ had to be used for online optimization algorithms at the moment of arrival t.

Given the POMDP for each node, in the Fog_env Gym the observation space is defined as a vector of all node partial observations $O(t) = (O_i(t), i \in I)$ and accordingly the action space is defined as $X(t) = (X_i(t), i \in I)$. Then the time step is also implemented in the Fog_env: it takes an action within the action space decided by the control plane as input, runs all the consequent events of an action until $t + 1$, and then returns the environment observation $O(t+1)$ and total reward $\psi = \sum_{i \in I} \psi_i(O_i(t), X_i(t))$.

It is noted that having multiple agents with a shared reward is a way to make a cooperative game, reaching an optimal Nash Equilibrium if the common reward is maximized. In this way there is no need to share large messages with the state, which might occupy the bandwidth, but rather a small reward signal that allows the cooperation to occur.

III. Algorithm Implementation

The DRL approaches to FC choreography, mean the RL algorithm is approximated with a Deep Neural Networks (DNN).

To achieve the best performance in the presented MDP in Section II-B, the function approximation for the RL algorithms is as proposed in [3], where the one dimensional Convolutional Neural Networks (CNN) provides the feature extraction from a sequence time steps and the Recurrent Neural Networks (RNN) allows a state memory to make necessary correlations between inputs, seeing that the FC environment has a time correlation between states.

In this section, the synchronous version of the Asynchronous Advantage Actor Critic (A3C) presented in [5], A2C, is implemented with a PPO loss actor step update as proposed in [6] for the MDP presented in section II-B. Since it is a multi-agent scenario with a common reward, the critic is a common estimator as proposed in [7] to allow for a better convergence. In short, a PPO with an A2C architecture DRL algorithm for a multi-agent scenario.

The loss function for the actors is the PPO loss function as proposed by the authors in [6], with the ϵ clipping parameter set to the value of 0.2. The advantage estimator used was the Generalized Advantage Estimator as proposed in [8], and used in [6] for the PPO algorithm.

The training is done with parallel short trajectories to collect information, and then apply the gradients in minibatch to the network, as popularized in [5] and [6] specially for training of RNN. A short description of the training process is presented in algorithm 1.

Algorithm 1: PPO with A2C architecture adaptation for multi-agent scenario

Create N agents and a common critic;
for *i = 1,2, ...* **do**
  for *env = 1,2, ..., E* **do**
    Run policy $\pi_{\theta old}$ in environment env for T timesteps;
    Compute advantage estimates $\hat{A}_1, ..., \hat{A}_T$;
  end
  Optimize actors surrogate L wrt θ, with L epochs and minibatch size $M \leq ET$;
  $\theta_{old} \leftarrow \theta$;
  Optimize critic surrogate L wrt w, with minibatch size M;
  $w_{old} \leftarrow w$
end

Seen that this method gives an online update of the algorithm, the reward calculation as referred in Equation 13, Section II-B, uses an estimation of the buffer size for the second term, rather than the complete information only obtained at time $t + D^t_{i,k}(t)$.

Another issue related with the online update is, since the states depend on the actions taken, successive misplaced actions might bring the environment to a pitfall. To avoid this, it is either necessary to define a failure parameter or to restart the environment periodically during training. The second option was chosen for the presented results. Which in turn allows for the algorithm to explore more courses of actions and not to over-fit the DNN to a stable reward case.

A. Baseline algorithms

For the comparative baselines decision to construct the action vector as described in Equation 8 of Section II-B, the decision was split between an offloading algorithm and a scheduling algorithm.

The offloading algorithms are the following:

- Nearest Node: is a threshold offloading with nearest node selection strategy, as used in [3] for the baselines, where each node will only offload tasks if it is above a defined threshold Th, and will offload it to the nearest node possible, minimizing both transmission time and energy consumption to incur in less costs.
- No Offload: will not offload any task, assuming each node would be better on its own and could handle tasks assigned to itself.
- Random Offload: is a threshold offloading with random node selection, will select a random node to offload tasks after the buffer capacity is above a defined threshold Th. This baseline works to demonstrate, together with Nearest Node, the impact of choosing or not the offload target in each test case.

For the scheduling algorithm, two different baselines were used, with improvements relative to the ones described in [3]:

- Round Robin: where the algorithm, based on the available resource pool R_i, schedules a tasks per slice in turns by increments of the action value, $w_{i,k}$, providing a reasonably spread action vector $X_i(t)$ to the POMDP interface.
- Priority Queuing: where the algorithm, based on the available resource pool R_i, schedules tasks starting in a higher priority slice, and only moving to lower priority if all higher priority buffers have their tasks under processing, by increments of the action value $w_{i,k}$ for each slice. Here, the priority is here determined by the delay budget.

The purpose is that the baseline algorithms also interact with the environment through the MDP, or more specifically for each node the POMDP. This way, in each time step, the algorithm in each node will receive a system observation $O_i(t)$ and build the action vector to apply in the next step of the environment $X_i(t)$.

IV. Experimental Results

For the particular FC environment, as described in Section II-B, the used base values in the tests are defined in table II.

Each of the I nodes is located within the plane of the defined Area, with assigned bandwidth BW_i, transmission power P_i, a CPU speed sampled from [5GHz, 6GHz, 7GHz, 8GHz, 9GHz, 10GHz], and a memory size sampled from [2400MB, 4000MB, 8000MB]. The allocation units defined in a node are η^c_i (1 GHz) and η^m_i (400 MB) for CPU and memory resources respectively. The surrounding environment has the path loss constant and exponent (β_1, β_2) and thermal noise density N_0.

TABLE II
SIMULATION CONFIGURATION.

Variable	Value(s)
Area	100 × 100 m^2
I	5
K	[1, 2, 3]
P_i	20 dBm
BW_i	1 MHz
η_i^c	1 GHz
η_i^m	400 MB
$\bar{b}_{i,k}$	10
$\lambda_{i,k}$	[0.6, 0.7, 0.8, 0.9]
(β_1, β_2)	(1, -4)
N_0	-174 dBm
T_k	5 Kbit
CPU	[5GHz, 6GHz, 7GHz, 8GHz, 9GHz, 10GHz]
Memory	[2400MB, 4000MB, 8000MB]
Th	0.8
ξ_k	[0.2, 2.0]

With the number of slices K defined either for an only offload problem ($K = 1$) or offload and scheduling ($K = 2$ or $K = 3$), the task packet size is constant in every slice with T_k = 5 Kbit and the arrival rate $\lambda_{i,k}$ is set by default to 0.6, i.e. every time step there is a 60% chance of a task arriving at a node slice. To test increased load impact the arrival rate was increased to the values of 0.7, 0.8 and 0.9.

The baseline offloading threshold Th is 0.8, meaning only when 80% of the slice buffer is occupied will the tasks begin to be offloaded. Whereas the tunable overflow penalty ξ_k is either set to 2.0 in test cases with no time constraint or 0.2 when there is time constraint.

Different test cases have to be created to verify the algorithm performances. The cases have two levels of heterogeneity: in terms of resources and in terms of delay constraints. To formulate the test case resource demands, it is necessary to take into account that the number of tasks that can be concurrently processed by a node is limited. Similar to Equations 11 and 12, the total number of concurrent tasks processing in each node, $W_i = \sum_{k \in K_i} b_{i,k}^e$ is constrained as follows:

$$\sum_{k \in K_i} b_{i,k}^e \leq \frac{U_i^c}{\eta_i^c}, \forall i \in I, \tag{16}$$

$$\sum_{k \in K_i} b_{i,k}^e \cdot \left\lceil \frac{L_k^m}{\eta_i^m} \right\rceil \leq \frac{U_i^m}{\eta_i^m}, \forall \in I. \tag{17}$$

Also, for a node to be able, on average, to handle tasks given to it, it is necessary that the number of tasks arriving is the same as the number of tasks being completed. However, the Fog is a resource constrained environment, where often the unbalanced load has to be balanced by a controller that does not necessarily provide a guaranteed 100% success rate.

The time constraint D_k^{max} provides the second level of heterogeneity. It can exemplify the good management of the resources, since a wrong distribution of the resources that hypothetically could complete all tasks might incur a task in missing its time constraint. Since the processors are limited to one unit per task, another constraint is posed on the delay,

$$\frac{L_k^c \cdot T_k}{\eta_i^c} \leq D_k^{max}, \tag{18}$$

meaning a task has to be able to be processed in the node where it is placed.

The test cases formulation are presented with the respective results, with the soft constraints provided by the Equations 16 and 17, and a hard constraint provided by Equation 18.

For the PPO algorithm, the tunable training parameters default values are presented in Table III.

TABLE III
PPO TRAINING PARAMETERS.

Parameter	Value
T	64
E	3
L	3
M	32

A. Performance metrics

To evaluate the performance of an algorithm in the test cases two metrics were used, and another one can be derived by the former two metrics. The first metric is the success rate, indicating the percentage of tasks assigned to the FC environment that was successfully completed and returned. The second metric is the overflow rate, which, in turn, indicates the number of tasks that tried to be offloaded to the FC environment and were refused due to a full buffer. The implied metric is the number of tasks that fail to meet their delay total constraint D_k^{max}, that were nor successful nor refused.

To evaluate the DRL algorithm convergence, a moving average of the reward over training iterations is used, since the purpose of the algorithm is to maximize the long term reward. However, it is to be noticed that, to maximize the evaluation metrics, it is necessary to formulate a reward that takes them into account, i.e., maps the objectives of the algorithm into a numeric function $R(s, a, s') \rightarrow \mathbb{R}$. A maximized reward will not guarantee maximized metrics.

B. Test environments scenarios

For the simple test cases, two specific test environments were created to demonstrate the performance of the algorithm. The developed scenarios were designed with the CPU clock speed and memory sizes of the nodes sampled from the values given in Table II, and within the specified space constraint in the same table.

On the first scenario, *Env1* as illustrated in blue in Figure 2, the node configurations were set with a smooth spacial

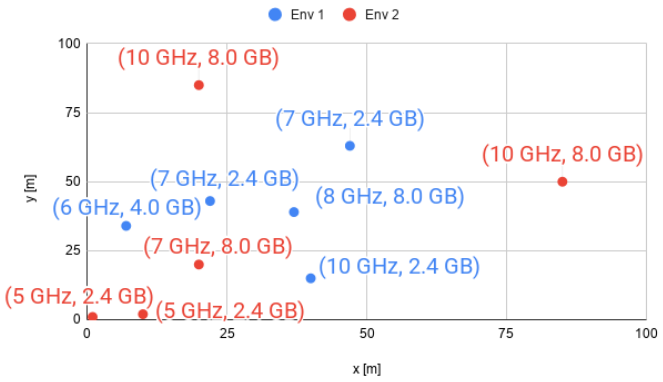

Fig. 2. Test environments. In blue (*Env1*) a favourable for Nearest Node offload strategy and in red (*Env2*) a case with no exact optimal strategy.

distribution but uneven resources, causing offloading to be necessary.

On the second scenario, *Env2* as illustrated in red in Figure 2, the nodes were spread unevenly on the plane and computing autonomy was given to two nodes. Here, the unsuccessful tasks are mainly due to the limited resources in the nodes closer to the origin.

C. Simple test cases

First, the algorithms were tested in a simple set of test cases with only one abstract slice (K=1), reducing the action space to two values per node, one to offload and another to indicate the number of tasks to schedule in the only existing slice. With a regular setup of 5 nodes and resources enough to concurrently process all tasks in the buffer, it translates to 60 different actions on each node.

For these simple test cases, the scheduler of the baseline algorithms was set to schedule all tasks in the buffer, according to their arrival, whenever resources are available. With that in mind, the algorithms that will be compared are:

- Nearest Node (NN);
- Random Offload (RO);
- No Offload (NO);
- PPO, with an A2C architecture, DRL algorithm with schedule and offload decision making.

The purpose of this section is to demonstrate the offload strategy learning capability of the proposed algorithm. In spite of the single slice, it also shows the learning capability of the algorithm to choose actions that involve scheduling, since the algorithm is not constrained in a lower bound, i.e., it can opt not to schedule a task.

TABLE IV
OFFLOAD TEST CASES PARAMETERS.

	D_k^{max} (ms)	L_k^c (cycles/bit)	L_k^m (MB)
Case 1:	1000	1200	800
Case 2:	15	1200	800

The first offload case, detailed in Table IV, was set to experiment if the algorithm could offload to a node with enough resources to take care of a task before buffer overflow, without actually constraining the task deadline. Whereas, a second offload case was created, similar to Offload Test Case 1, but now with a delay constraint on the number of milliseconds a task can take to complete.

Running the algorithms in the two test environments for the first test case, the results obtained are presented in Table V.

TABLE V
PERFORMANCE IN TWO DIFFERENT ENVIRONMENTS FOR THE FIRST OFFLOAD CASE.

	Algorithm:	NN	RO	NO	PPO
Env1	Success rate (%)	99.8	94.0	88.5	99.5
	Overflow rate (%)	0.2	6.0	11.5	0.5
Env2	Success rate (%)	93.4	93.3	93.5	99.9
	Overflow rate (%)	6.6	6.7	6.5	0.1

The resources constraints of the first trial, in *Env1*, with this specific case lie on the memory capability to have concurrent tasks, so the Nearest Node offload strategy has inherent advantage in the pairing happening between the memory resource rich nodes with memory resource constrained nodes in the first scenario. The PPO algorithm also managed to optimize the offloading strategy, showing a success rate of 99.5%. It is also assumed that it managed to schedule tasks successfully, choosing the right action from the scheduling action space, which is limited by the number of resources (Equations 11, 12).

However, it is to be noticed that, with such a high delay constraint defined on Offload Test Case 1, the reward function defined in Equation 13 will be highly unlikely to miss its delay constraint, so the reward function is mostly limited to the second term of overflow. The PPO algorithm rather than opting for an offload strategy within the delay constraint, will minimize the overflow penalty, and therefore maximize the success rate in this case where almost no task will be discarded for not meeting its delay constraint. With a simple reward like this, convergence to an optimal case is simpler to achieve. In a more complex case, convergence is not guaranteed to go to the maximum value, since it can fall on a local maximum or even diverge.

Given more resources for each node, in *Env2*, scheduling whenever there is a task and resources available with No Offload strategy results in a total success rate of 93.5%, with the buffers overflowing with the other 6.5% of the tasks. With the Random Offload strategy, which offloads tasks when the threshold is reached, didn't improve the success rate, keeping it at 93.3%. Whereas the Nearest Node offloading strategy fell into a trap where the two most resource limited nodes are the nearest to each other, failing to improve the success rate, which is kept at a value of 93.4%. Using the offload strategy trained with the PPO algorithm the nodes learn to offload for

a node with more resources increasing total success rate of a test run to 99.9%.

Now with added time constraints, running the algorithms in the two test environments for the second test case detailed in Table IV, the results obtained are presented in Table VI.

TABLE VI
PERFORMANCE IN TWO DIFFERENT ENVIRONMENTS FOR THE FIRST OFFLOAD CASE.

	Algorithm:	NN	RO	NO	PPO
Env1	Success rate (%)	95.5	79.6	67.3	86.9
	Overflow rate (%)	0.1	1.8	4.0	0.9
Env2	Success rate (%)	81.2	78.3	80.2	83.9
	Overflow rate (%)	1.4	2.7	1.9	1.9

The Nearest Node offload strategy will always opt for the best strategy possible in the first scenario, due to the distance pairing of the resource constrained nodes with resource rich nodes. The PPO algorithm manages to find a reward convergence during training, but it is not the best achievable solution with a success rate of only 86.9% and an overflow rate of 0.9%.

On the other hand, the PPO algorithm, although converging to a sub-optimal reward, demonstrated a stable performance in the second scenario with a success rate of 83.9% and an overflow rate of 1.9%. This hints towards a more flexible approach to an FC environment. While simple algorithms can outperform the algorithm in favorable situations, the PPO algorithm, with a well formulated MDP, can keep a decent and more flexible performance.

D. Increased load

The Offload Test Case 1 was run with the second scenario *Env2*, which has less resource constraints as presented in Figure 2, with a changing the arrival rate adopting the values of 0.7, 0.8 and 0.9. This was done to test the impact of an increased load for the algorithm training. The results are presented in Table VII.

TABLE VII
SUCCESS RATE ON THE SECOND SCENARIO WITH OFFLOAD TEST CASE 1 OVER INCREASING ARRIVAL RATE.

Arrival rate	NN	RO	NO	PPO
0.7	88.0%	89.2%	88.0%	93.5%
0.8	84.5%	85.1%	84.5%	89.0%
0.9	82.3%	82.2%	82.3%	86.5%

As observed in Table VII, even with increasing resource constraints, the PPO algorithm can manage to outperform the baselines.

E. Reward function impact

To test the influence of the MDP reward formulation, tests were made with different formulations for the same case. The used case is the Offload Test Case 2 presented in Section IV-C, using the test environment *Env1* illustrated in Figure 2. It is done so, to demonstrate that a poor PPO performance can be due to its MDP formulation.

Based on the idea that the global objective is to maximize the success rate, shorter delays would mean a better capacity to process tasks and therefore maximize the success rate. With that rationale, a new reward, based on Equation 13, is presented, where the term described in Equation 14 is altered to:

$$S_{i,k} = \beta \cdot (-1)^{\mathbb{1}(D_k^{max} \leq D_{i,k})}, \tag{19}$$

where β, the new term inserted, would scale the reward within the delay constraints to minimize delay. That is done with the following equation:

$$\beta = \begin{cases} \left(1 - \dfrac{D_{i,k}^q + D_{i,j,k,n}^t}{D_k^{max}}\right) & \text{if } D_k^{max} \geq D_{i,k}, \\ 1 & \text{otherwise.} \end{cases} \tag{20}$$

The Equation 20 takes into account that the delay to process a task $D_{i,k}^p$ is constant in this specific environment, due to core attribution imposed limit, so unnecessary to minimize. It is also to be noticed that the equation always rewards whenever a tasks is completed within its delay constraint, i.e., it doesn't have a zero value.

Another formulation for the reward was done to only penalize the system. The purpose is to decrease the probability of the worst actions enough, for them not to become a favourable exploring path. With that intent, the reward formulation is a simple penalty whenever a task is not completed, either by being refused in an overflow situation or by failing to meet its deadline the penalty is calculated as follows:

$$\psi_i(t) = (-1) \cdot (f + o), \tag{21}$$

where f is the number of tasks that missed its deadline and o the number of tasks that overflow in time step t.

A third reward formulation was build with the objective of maximizing the success rate and minimizing the overflow. It rewards whenever a task is completed and penalizes whenever a task is discarded due to a full buffer:

$$\psi_i(t) = (+1) \cdot s + (-1) \cdot o, \tag{22}$$

where s is the number of successfully completed tasks and o the number of tasks that overflowed in time step t.

TABLE VIII
PERFORMANCE ON THE FIRST TRIAL OF OFFLOAD TEST CASE 2 FOR DIFFERENT REWARD FUNCTIONS.

	NN	PPO (13)	PPO (19)	PPO (21)	PPO (22)
SR (%)	95.5	86.9	90.2	93.9	99.5
OR (%)	0.1	0.9	0.8	0.2	0.1

As it can be observed in Table VIII, the MDP reward formulation can have a great impact on the algorithm performance. This is a key point in applying the DRL algorithms to FC choreography, where the objective function has to

be modelled in order to provide ease of convergence and maximization of performance. The Nearest Node strategy, since it previously had the best results, was outperformed with the correct performance optimization.

Different rewards can be more or less robust. The designed reward in Equation 8 reaches convergence smoothly. The modification to the original reward done in Equation 19 increased the performance while also keeping a smooth convergence. The sparse reward of Equation 21, although improving the results for this specific test case, has difficulties in converging since it leans heavily towards exploration. The final reward function, in Equation 22, managed to converge and provide exceptional performance for this case, however, the training hyper-parameters had to be tuned since it proved likely to converge to a sub-optimal value if not given enough exploration options.

F. Complex test case

The addition of a second slice (K=2) in the FC environment increases the complexity of the test cases exponentially, increasing the action space to four different values and allowing task heterogeneity throughout the slices. With just 5 nodes and enough resources to process all tasks in each buffer, the limit action space size for each node is 2500.

This addition requests the baselines to use the scheduling algorithms, as presented in Section III-A. Along with the offload strategy, the baselines are named as follows:

- Nearest Node Priority Queue (NNPQ);
- Nearest Node Round Robin (NNRR);
- Random Offload Priority Queue (ROPQ);
- Random Offload Round Robin (RORR).

With two different slices, a resource heterogeneity is offered in the first test case:

TABLE IX
COMPLEX TEST CASE 3 PARAMETERS.

Slice K	D_k^{max} (ms)	L_k^c (cycles/bit)	L_k^m (MB)
1	1000	400	1200
2	1000	1200	400

A test run is made on the *Env2*, with the results presented in Table X.

TABLE X
COMPLEX TEST CASE 3 PERFORMANCE RESULTS COMPARISON WITH BASELINES.

Algorithm	PPO	NNPQ	NNRR	ROPQ	RORR
SR (%)	93.1	87.1	92.3	87.9	93.8
OR (%)	6.9	12.9	7.7	12.1	6.2

An explanation for the bad performance overall in Priority Queue scheduling is that, with no delay constraints, the priority attributed to a higher computational demanding slice might greatly increase the overflow rate. On the other hand, the PPO algorithm managed to match the best baseline for this specific case. Here, the reward used was the one described in Equation 13 since it provided the best results without the need of hyper-parameter tuning. This hints towards a possible approach to choreography in a complex environment based on DRL algorithms.

A second test case with two slices is presented. In this case, both resource and delay constraint heterogeneity are offered:

TABLE XI
COMPLEX TEST CASE 4 PARAMETERS.

Slice K	D_k^{max} (ms)	L_k^c (cycles/bit)	L_k^m (MB)
1	15	400	1200
2	50	1200	400

A test run is made on the *Env1*, with the results presented in Table XII.

TABLE XII
COMPLEX TEST CASE 4 PERFORMANCE RESULTS COMPARISON WITH BASELINES.

Algorithm	PPO	NNPQ	NNRR	ROPQ	RORR
SR (%)	91.9	97.1	96.1	89.6	86.7
OR (%)	2.9	0.5	0.6	3.9	4.1

In this specific case, the Priority Queue scheduling has an inherent advantage, since prioritization coincides with a less computing demanding task, which results in an overall better success rate. Since the *Env1* favours the Nearest Node strategy, the best baseline is the Nearest Node Priority Queue with 97.1% success rate.

The PPO algorithm is trained with the reward presented in Equation 22. It fails to achieve the best optimization, scoring a 91.9% success rate in this test case, but demonstrates learning ability. However, with further testing the algorithm was shown not to be robust enough. Hyper-parameter tuning was necessary to achieve a reasonable performance, and it was not always the best. Its performance limitations are due to the non-linear approximation used, the nature of the PPO algorithm itself, the multi-agent scenario, and the MDP formulation itself.

G. Limitations in DRL for Fog Computing

The first limitation comes from the DNN approximation used in DRL. It is possible that the network proves to correctly approximate a sub-set of states, i.e. has a low loss for the sub-set, and therefore does not perform any major changes in the network. This cripples the exploration policy and the algorithm might reach convergence with a sub-optimal performance.

A second limitation comes from the PPO algorithm itself. As any on-policy algorithm, the bias-variance duality is an issue that influences the convergence value. Whereas short trajectories allow for a smaller variance in the reward, and therefore the algorithm can adapt its actions easier to converge to a maximum, the bias to the total expected reward might be big enough for the convergence to be for a local maximum.

On the other hand, a long trajectory would have a smaller bias to the total expected reward, but a a high variance, i.e. small change in the actions could have big impacts in the final reward, causing the reward convergence to be harder to find. For the presented test cases, the PPO algorithm managed this trade-off with a trajectory length of 64 time steps, performing an n-step TD target bootstrap.

A third limitation is the multi-agent scenario, which creates a non-stationary environment with the changing policies of other agents. In this case, a cooperation game is required and, in order to ease convergence in training, a common critic is implemented as proposed in [7]. However, the construction of the critic comes as an extra variable for the convergence.

A fourth source of performance limitations comes from the MDP itself. Firstly, the information in the state is not exhaustive, making it so that the algorithm has to extrapolate the information by itself. However, this is not the main problem of the MDP, since the model free approach of this algorithm can easily adapt. The main limitation of the MDP is the reward function definition. For an easier convergence, it has to be designed so that the state-action relation with the reward is coupled. A sparse reward might increase the training time to unreasonable duration or provide poor performance, as discussed in [9] for Montezuma's Revenge. For a good performance, the reward has to translate the metrics of the optimization objective, and possibly have a concave shape to improve convergence for a global maximum.

Lastly, the reward designed in Equation 13, has a term that has to be estimated in an on-policy method. Only an off-policy method could store the reward for later completion before inserting it in a replay buffer - in order to tightly correlate the state-action-reward. Another note is that the total reward of this system is heavily dependent on the resources and placement of the nodes, as it should be. But this will translate into a more complex function to converge, dependent on the test case, so original training parameters are not guaranteed to converge for all cases.

V. Conclusions and Future Work

In a specific sub-set of DRL, the actor-critic algorithms, it was shown that the PPO algorithm can adapt to different situations. It managed to obtain the optimal offload strategy when it was known, for example in the Nearest Node, and when it was unknown. It also managed to schedule tasks for completion based on a combined offload-schedule action. This demonstrated a flexible algorithm that can achieve a good performance without the need of human interaction, which, in turn, implies that there is no the need to choose a different algorithm depending on the scenario. However, there are limitations found to the demonstrated performance. First, in the non-linear DNN approximation used for the PPO algorithm, where an over-fit to a subset of sates cripples the policy exploration. Secondly, in the PPO algorithm itself, where the bias-variance duality influence the reward value to which the algorithm converges. Therefore, to improve performance there is the necessity to tune hyper-parameters, especially the trajectory length. Thirdly, the application of the PPO algorithm in this multi-agent scenario implies a cooperation game, where convergence to a common maximum might be hard to achieve. For this, other techniques should be explored to improve or replace the common critic for training. Lastly, the MDP formulation can deeply impact the performance of the PPO algorithm. A design that manages to both represent the real system, in the state and the action, and to represent the performance objective, in the reward, has to be accurately created depending on the situation.

Nevertheless, DRL should be considered as path to explore in FC resource management, since, after training to an optimal strategy, it can map complex operations in a relatively short time and memory space.

A. Future steps

To enable a deliverable DRL algorithm for FC resource management it is still necessary to investigate further. Future steps of research include improving and expanding the presented FC simulation environment, build a comprehensive test set to be used as standard comparison with other algorithms, implement other DRL algorithms to compare in this environment and keep up with the DRL research for multi-agent scenarios, and lastly to explore the orchestration side of resource management with the inclusion of DRL.

References

[1] A. Yousefpour, C. Fung, T. Nguyen, K. Kadiyala, F. Jalali, A. Niakanlahiji, J. Kong, and J. Jue, "All one needs to know about fog computing and related edge computing paradigms: A complete survey," *Journal of Systems Architecture*, 08 2018.

[2] O. Consortium, "Openfog reference architecture for fog computing," 2017, [Online] Available on https://www.iiconsortium.org.

[3] J. Baek and G. Kaddoum, "Heterogeneous task offloading and resource allocations via deep recurrent reinforcement learning in partial observable multi-fog networks," *IEEE Internet of Things Journal*, pp. 1–1, 2020.

[4] C. E. Shannon, "Communication in the presence of noise," *Proc. Institute of Radio Engineers*, vol. 37, no. 1, pp. 10–21, 1949.

[5] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *Proceedings of The 33rd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. F. Balcan and K. Q. Weinberger, Eds., vol. 48. New York, New York, USA: PMLR, 20–22 Jun 2016, pp. 1928–1937. [Online]. Available: http://proceedings.mlr.press/v48/mniha16.html

[6] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *ArXiv*, vol. abs/1707.06347, 2017.

[7] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," 2020.

[8] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," 2018.

[9] V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. Bellemare, A. Graves, M. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–33, 02 2015.