Real-time and Energy-aware Scheduling for Edge-to-Cloud Continuum based on Reinforcement Learning

Andrea Panceri, Gabriele Proietti Mattia, Roberto Beraldi
Department of Computer, Control and Management Engineering “A. Ruberti”,
Sapienza University of Rome,
ORCID: 0009-0005-2438-5982, 0000-0003-4551-7567, 0000-0002-9731-6321

Abstract—**By spreading out computing workloads over multiple levels, the Edge-to-Cloud continuum paradigm improves the performance of applications that are sensitive to latency. However, real-time scheduling is difficult on the Edge computing layer since it consists of a variety of nodes with varying uptime. In this paper, we address this issue by proposing an online and adaptive scheduling algorithm based on a continuous learning Reinforcement Learning. Our algorithm determines each work request individually, optimizing scheduling policies to meet real-time application requirements while taking environmental energy and battery limits into account. We validate the efficacy of our approach in dynamically assigning tasks, particularly in scenarios where Edge nodes exhibit variable speeds and unpredictable failures, while efficiently managing energy resources and battery constraints through extensive simulations and comparisons with static scheduling strategies.**

I. Introduction

In the contemporary digital context, the necessity for real-time processing extends beyond mere entertainment applications, such as augmented reality and virtual reality (AR and VR). Tasks that mandate immediate processing, including but not limited to industrial automation and healthcare monitoring, are ill-suited for cloud solutions due to the constraints imposed by network latency. To bridge this gap, fog computing, also referred as edge computing, has emerged as a viable solution by positioning an intermediary layer in closer proximity to end users, thereby reducing latency [1]. However, challenges persist, particularly in relation to the diverse nature of edge nodes.

In the intermediary layer of the Edge-to-Cloud continuum, nodes exhibit a wide range of hardware and software capabilities. This diversity complicates the task of job allocation, a complexity that is further exacerbated by the inherent constraints on resources, particularly in terms of power consumption and battery management, where resources are inherently limited [2]. The efficient assignment of jobs in such an environment necessitates a careful consideration of these factors, acknowledging the intricate interplay between hardware capabilities, software configurations, and resource constraints. This study addresses the complex problem of task scheduling and battery management within the context of the Edge-to-Cloud continuum. Our proposed approach incorporates multi-objective optimization, which encompasses power management in addition to real-time processing demands, to ensure both performance and sustainability. Our methodology takes into account the heterogeneity of nodes and dynamically organizes jobs to meet user-specified processing requirements. This is achieved through the utilization of online Reinforcement Learning, a machine learning technique that enables the system to learn and adapt to the dynamic environment in real-time.

The computing environment under discussion is shown in Figure 5, which consists of clusters with different numbers of worker nodes and scheduler nodes. We take a simple but suitable situation for our experiment: only one cluster of nodes with a single scheduler and a set of worker nodes. We deploy a learning agent tasked with observing the state of worker nodes and making scheduling decisions in response to task execution requests from end users. These decisions may involve executing tasks on specific worker nodes, in the cloud, or rejecting the request altogether. The learning agent receives positive rewards contingent upon our prioritization criteria. Specifically, the reward structure depends on whether we prioritize extending the lifespan of batteries or meeting task deadlines. This prioritization is achieved through the introduction of a parameter, denoted as α, which ranges from 0 to 1. This parameter serves to weight the contribution of each component of the reward, allowing us to dynamically adjust the emphasis placed on battery lifespan preservation versus adherence to task deadlines.

The primary contributions of our work are as follows:

- **Development of a reinforcement learning-based online scheduling algorithm** for the computing continuum, capable of handling node heterogeneity, meeting user-defined processing frame rate requirements, and optimizing battery lifespan while maintaining balanced battery loading.
- **Presentation of simulation results** showcasing the performance of our proposed algorithm in various settings, including a case where we have only workers and a case where we have also cloud. Our simulator replicates fine-grained delays encountered during job execution paths.

The structure of the remainder of this paper is outlined as follows: Section II provides an overview of related works, in Section III, we elaborate on the proposed reinforcement learning approach. Following this, Section V offers insights into the simulation results of the proposed protocol, and finally, in Section VI, we draw the final conclusions.

II. Related Work

Unlike episodic methods, we use a Reinforcement Learning (RL) algorithm designed for continuous learning in our work. In particular, our approach incorporates Differential Semi-Gradient Sarsa (DSG Sarsa), as described in [3], which we used in this previous work [4] that was exclusively concerned with optimizing frame rates. While DSG Sarsa tackles frame rate issues, our new algorithm goes beyond its limitations to include multi-objective RL factors, with a focus on battery balancing. This update is in line with the increasingly complex edge and fog computing environments, which need for careful consideration of all factors when allocating resources in order to maximize efficiency. [5] propose a rl algorithm that improve batteries management, respect to our work focus only on one energy aspect. Because Deep Q Learning (DQL) is good at handling high-dimensional state spaces, it has been widely used in the literature to tackle similar job scheduling problems in edge or fog computing scenarios. As an illustration, in [6], DQ is utilized for resource allocation in Mobile Edge Computing (MEC) systems, deviating from our focus on battery balance and giving priority to execution time minimization through time slice-based allocation. In a similar vein, [7] looks into base station selection in ultra high-density networks and emphasizes the improvements in performance that come from using their suggested DQL-based approach. In contrast to our online scheduling method, [8] focuses on task scheduling in edge computing and uses DQL for task execution ordering and machine assignment, but in an offline environment. [9] offers an RL-based caching solution for edge environments, further examining RL applications. It optimizes stochastic allocation policies using simulation-based evaluations. As for mobile device management in cellular networks, [10] explores this area by using RL in conjunction with stochastic gradient descent for real-time system improvement, tackling issues like power distribution and uplink transmission. While there are no hard job deadlines, the work closely complies with [11], emphasizing online scheduling with time differential learning techniques. Furthermore, [12] highlights the complexities of resource allocation in dynamic computing settings by providing a thorough analysis of task placement throughout the edge-to-cloud continuum.

III. System Model and Problem Definition

This study approaches the online scheduling problem by formalizing it as a Markov Decision Process (MDP) and employing Reinforcement Learning (RL) for its solution. Adopting a model-free approach eliminates any prior assumptions regarding the underlying mathematical model. The learning agent observes the current environment state and takes actions based on its accumulated knowledge (exploitation) or through random exploration. These actions represent scheduling decisions, dictating where tasks should be executed. Subsequently, upon task completion, a reward signal is received, crucial for driving the learning process. The entire framework is implemented within a delay-focused discrete events simulator, leveraging the Simpy library in Python [13]. The solution framework remains adaptable for real-world applications that conform to the presented task model. This section proceeds to delineate the entities integral to the problem domain: the environment, task and delay models, state representation, and reward system. Additionally, we introduce the energy balancing aspect, a vital consideration for sustainable computing systems.

A. Environment

A conceptual framework of a computing continuum environment, with the learner agent positioned at the edge layer, is shown in figure 5. To be more precise, our configuration consists of a single cluster that houses one scheduler node, called h, and a predetermined number of worker nodes, called $W = \{w_1, w_2, ... \}$. The scheduler node's responsibility is to handle requests for task execution coming from the clients (end users). It chooses whether to reject a task or where it will be executed: locally in the cluster (by identifying the particular worker node) or remotely in the cloud. Every second, the scheduler node receives λ_i requests, each of which triggers a scheduling decision based on reinforcement learning (RL) algorithms. The worker nodes, on the other hand, have a fixed-size queue, K, and can only process one task at a time. A task is denied if it is assigned to a node and the queue's current capacity is K or greater. These worker nodes are noteworthy for their heterogeneity; each is correlated with an execution speed S_i, which denotes a time extension factor for tasks carried out on that specific node. A task with a notional time of 15 ms, for example, would take 25 ms to execute on worker node 3 if S_3 = 0.6. The idea of execution speed is modeled after the actual situation in which a worker node's available CPU time is allotted to task execution, which may vary over time. However, we assume the execution speed as constant for the purposes of this study. Every worker node has a battery as well. We also take into consideration the power used for transmission power, CPU utilization for job execution, and power consumption during idle times, all expressed in watt-hours (WH). As a result, if a worker node's battery runs out from excessive use or inadequate charging, the node will not function.

B. The Agent

The agent's primary objective is to learn an effective scheduling policy, denoted as π, which is a function of the current state:

$$\pi : S \to A \tag{1}$$

In this context, the policy π maps a given state $s \in S$ to an action $a \in A$, where A represents the set of all possible actions. Initially, in our experimentation phase, actions involve either task rejection or assignment exclusively to worker nodes. Thus, the action set for node i is described as:

$$A = \{reject\} \cup W_i \tag{2}$$

In a subsequent phase (Section V-D), we introduce the option of forwarding tasks to the cloud, expanding the set of available actions to:

$$A = \{reject, cloud\} \cup W_i \tag{3}$$

It's important to note that only the scheduler node is responsible for receiving task requests and making scheduling decisions. Furthermore, in our multi-objective reinforcement learning framework, we incorporate a parameter $\alpha \in [0, 1]$. This parameter allows for the adjustment of reward criteria, providing flexibility in optimizing various objectives simultaneously.

C. State representation

The possible states of the environment are all included in the set S. In order for the agent to decide what to do, it must have access to an environment representation with all the information it needs. Here, the only data that is available is the quantity and kind of jobs that are currently scheduled on each worker node, together with the normalized battery lifespans of each one. This limitation results from the requirement that every task go to the scheduler. The matching task-type counter for worker j is increased upon arrival by a task of type i given to worker j, and it is decreased upon completion. As said in the introduction, we still cannot determine the node speeds even though we know the task type when we arrive. A scheduler node with three worker nodes and two job kinds is shown in Figure 1 as an example of the state representation. An integer (corresponding to number 1) designates each job type, and is followed by tuples that indicate how many of each type of task are in the queue for each worker node. Furthermore, it comprises the normalized battery lifespans for every worker node, with the highest lifespan equal to 1 and the minimum lifespan to 0, and intermediate values for the other worker. However, the raw state representation is not directly utilized in

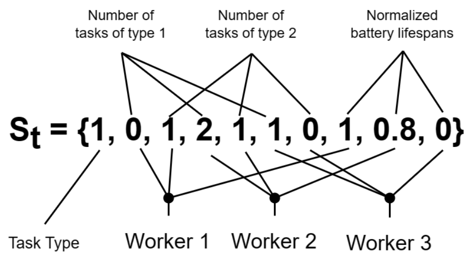
Number of tasks of type 1
Number of tasks of type 2
Normalized battery lifespans
S_t = {1, 0, 1, 2, 1, 1, 0, 1, 0.8, 0}
Task Type
Worker 1
Worker 2
Worker 3

Fig. 1: State representation of the scheduler node at time t

the learning process. Instead, we employ the tiling technique [14] to map the vector into a 24-dimensional vector space. Furthermore, the task type, defined by the user, encapsulates all task characteristics and traffic flow attributes, including arrival and desired execution rates. While the lifespan value encapsulates all energy characteristics, including battery capacity and battery consumption.

D. Reward

The determination of the reward is pivotal in achieving the desired outcomes of meeting user Quality of Service (QoS) constraints while optimizing the utilization of available resources. In our investigation, we direct our attention towards specific applications wherein frames are generated by devices and sequentially processed by a back-end server. The outcome of this processing is then displayed to the user on a screen, assuming synchronization between the screen refresh rate and frame generation. Notably, the back-end server operates under an energy constraint due to limited battery capacity, thus necessitating a balanced consideration between frame per second (fps) and battery lifespan. Our primary objective is to ensure minimal lag at the client's end while accommodating a minimum acceptable response frame rate from the server. Additionally, we seek to optimize battery lifespan, maintaining equilibrium across all batteries during simulation. To establish an optimal reward framework, we draw upon methodologies employed in our prior research [4] concerning FPS performance. To maximizing service time, our focus lies on prolonging the lifespan of each worker node. This entails calculating the lifespan of each worker based on battery capacity and current power consumption. In our proposed reward formulation, we take into account both the frames per second (FPS) performance and the battery lifespan of the devices. The reward is defined as follows:

$$\text{reward} = \text{reward\_fps} \times \alpha + (1 - \alpha) \times \text{reward\_batteries} \tag{4}$$

The parameter α plays a crucial role in this formulation as it enables the prioritization between the two performance measures. By selecting an appropriate value for α, we can optimize the results and achieve the best tradeoff between FPS performance and battery lifespan.In the subsequent sections, we will demonstrate how the selection of α can significantly impact the results. We will also identify the optimal value of α that yields the best tradeoff between the two performance measures. Finally The reward_fps is defined as follow:

$$R_{fps}(s, a) = \begin{cases} 2 & \text{if } d_t \leq \frac{1}{\omega_n} \\ 1 & \text{if } \frac{1}{\omega_n} < d_t \leq \frac{1}{\omega_m} \\ -1 & \text{if } d_t > \frac{1}{\omega_m} \\ -4 & \text{if } action \ = \ REJECT \end{cases} \tag{5}$$

Where ω_n and ω_m are particular to the given traffic flow, and s is the status as observed by the scheduler upon arrival of the frame and a is the chosen scheduling action. While the reward_batteries is defined as follow:

$$R_B(s, a) = \begin{cases} 2 \cdot \Gamma[action] & \text{where } \beta \in \Gamma, \ \beta \in [0, 1] \\ 1 & \text{if } action \ = \ CLOUD \\ -4 & \text{if } action \ = \ REJECT \end{cases} \tag{6}$$

Where Γ is the list of normalized values of lifespans of each worker, where each element corresponds to a worker, indexed sequentially.

E. Performance Parameters

In order to evaluate the performance of our reinforcement learning-based scheduling algorithm, we present the following performance metrics to provide a comprehensive evaluation of our algorithm:

- The variance of the remaining watt-hours (Wh) of each battery is a performance metric that we aim to minimize in order to achieve balanced utilization of the batteries. The optimal value for this metric is zero.
- Another important performance metric is the maximum-minimum difference of the remaining watt-hours (Wh) of the batteries when the first worker dies. This metric provides an indication of the fairness of the algorithm in terms of device utilization. The optimal value for this metric is zero, indicating that all batteries are being utilized evenly.
- The minimum and maximum lifespan of the devices is an important performance metric that provides an indication of the overall device utilization. The minimum lifespan refers to the shortest amount of time that a device is able to operate before its battery is depleted, while the maximum lifespan refers to the longest amount. A large difference between the minimum and maximum lifespan indicates that some devices are being utilized more heavily than others, which can lead to an imbalance in the overall system.
- The ratio of the number of tasks that meet their deadline to the total number of tasks, which provides an indication of the algorithm's ability to meet the QoS requirements. This metric is further broken down into three categories based on the job type (0, 1, and 2). We aim to obtain
- Also important is the total service time, that represent the sum of all service time of all workers. The algorithm aim to reach the greatest value possible. We can use as baseline the maximul lifespan algorithm.

IV. Online Scheduling Decisions with RL

The objective of the reinforcement learning agent in this paper is to learn a scheduling policy, denoted as π, that maximizes the long-term reward in an online task scheduling environment. Since the decisions are made in a continuous and ongoing manner, the problem is treated as a continuing learning task. In such a task, it is more effective to consider the current average reward, rather than discounting future rewards, in order to make the right decision. Given a state $s \in S$ defined in Figure 1, the agent performs an action $a \in A$, and obtains an immediate reward r, with the next state being $s' \in S$. The optimal policy, which maximizes the long-term reward, is defined by the optimal q_* function, as shown below:

$$q_*(s,a) = \sum_{r,s'} p(s',r|s,a) \cdot \left[r + \max_{\pi} r(\pi) + \max_{a'} q_*(s^{'},a^{'})\right] \tag{7}$$

The Sarsa algorithm is used to learn the policy, and at a certain time t, the differential form of the error, δ_t, is expressed as shown in Equation 8. This form can be applied to any function approximation algorithm for estimating the q^*, and in this paper, the tiling technique is used.

$$\delta_t = R_{t+1} - \overline{R}_{t+1} + \hat{q}(S_{t+1}, A_{t+1}, \vec{w}_t) - \hat{q}(S_t, A_t, \vec{w}_t) \tag{8}$$

It is important to note that in the setup of this work, the reward_fps is never immediate, as it is only known after a task has been executed or rejected and returned to the client. Therefore, a window size of Z tasks is set, and the weights are updated for all the tasks in the window, only if the window is reached and all the tasks in the window have been executed or rejected. We can obtain the reward_batteries immediatly after the scheduler choose an action, but for compute the total amount need both part of reward 4. Algorithm 1 of [4] is executed by the scheduler, whenever a new task to be executed arrives. The task is appended to the array of pending tasks, and the state is computed, as described in Section III. The best action to perform, given the current q(s, a, $\vec{w}$), is then retrieved. If the action is 0, the task is rejected, if it is 1, the task is forwarded to the cloud (only in the case where we have also the cloud), and otherwise, the action number is used to derive the index of the worker of the cluster to which the task is forwarded. In the case where the task is scheduled to be executed in a worker node, the current queue length is checked, and the task is rejected if it is equal to or exceeds the limit K. Algorithm 2 of [4] is executed every time a task completes its execution. The task reward is recorded, and the array of pending tasks is iterated over to check if the first ζ tasks of the array are finished. If this is not the case, the function returns, and otherwise, the information about the first ζ tasks is retrieved by popping them from the array. This information is used to train the weights vector $\vec{w}$ using the semi-gradient differential Sarsa algorithm. The significant modification we have made in comparison to previous work is the development of a multi-objective algorithm. This allows us to adjust the desired outcome by varying the value of the parameter α, enabling us to focus on the specific objective of interest.

V. Results

In this section, we present the results of our proposed reinforcement learning (RL)-based scheduling algorithm in both edge-only and edge-to-cloud continuum environments. All experiments can be reproduced using the public repository available on CodeOcean [1]. In all experiments, we consider the battery awareness of the devices and incorporate it into the RL algorithm to ensure that the devices are not over-utilized and their battery life is maximized. We derive the execution speeds of the workers in the Edge node and the maximum queue length from the technical parameters of real devices. The service rate is normalized with respect to the highest clock speed in the group. For example, the service rate of the Asus Tinker (1.8 GHz) is 0.9. The cloud, on the other hand, always runs at a speed equal to 1.0.

[1]The code is temporarily available on the https://github.com/Pancio-code/Simulator while awaiting publishing from the CodeOcean capsule.

A. Complexity Analysis

In our work, we have chosen to use the Differential Semi-Gradient (DSG) Sarsa [3] algorithm. This choice was motivated by the need for a reinforcement learning algorithm that has a low computational overhead and is capable of fast convergence. The use of tiling and hash table in the implementation of the DSG Sarsa algorithm allows for a constant time complexity, $\mathcal{O}(1)$, for both the training and execution of the algorithm. This is a significant advantage in the context of edge nodes, where the availability of computational resources is often limited, and the need for fast and efficient task scheduling is crucial. In conclusion, the use of the DSG Sarsa algorithm, implemented using tiling and hash table, in our online task scheduling algorithm is motivated by the need for a fast and efficient reinforcement learning algorithm that is capable of handling the complex and dynamic nature of the edge nodes, while also being mindful of the limited resources available.

B. Adaptability to change

In order to demonstrate the adaptability of our proposed RL-based scheduling algorithm to changes in the environment, we conduct a series of experiments. Specifically, we use the same setting as in the case of V-D, where the cloud is available. In these experiments, we simulate a crash failure of the most powerful worker node (w_1) at timestamp 4000 (s). The node remains in a failed state until timestamp 8000 (s), after which it returns to normal operation. It is important to note that during the failed state, the node is still consuming power, albeit in an idle state. This is an important consideration for the management of the batteries in the edge nodes. We have selected a timestamp of 4000, as the algorithm has already converged to the desired results by this point in the simulation (We can see it from the reward graph 2). Furthermore, we explore the performance of our algorithm in two different scenarios. In the first scenario, we set the value of the parameter α to 0, with the goal of maximizing the lifespan of the service while maintaining a balanced balance of the batteries in the edge nodes. In the second scenario, we set the value of α to 1, with the goal of satisfying the frames per second (FPS) requirements of the clients. We present the results of the first scenario in Fig. 2, where the value of α is set to 0. The results (Fig. 2) show that the algorithm quickly reacts to the failure of the worker node by offloading the jobs to the cloud in order to maintain a balanced balance of the batteries in the edge nodes. This is an important consideration for the management of the batteries in the edge nodes. It is worth noting that, during the failed state of the worker node, the jobs are still able to meet the frames per second (FPS) requirements of the clients. However, this is not the primary focus of the algorithm in this scenario. The results also show that the variance in the balance of the batteries in the edge nodes grows only slightly during the failed state of the worker node. At timestamp 8000, the worker node returns to normal operation and the algorithm immediately shifts the load back to the worker nodes. The results show that the variance of

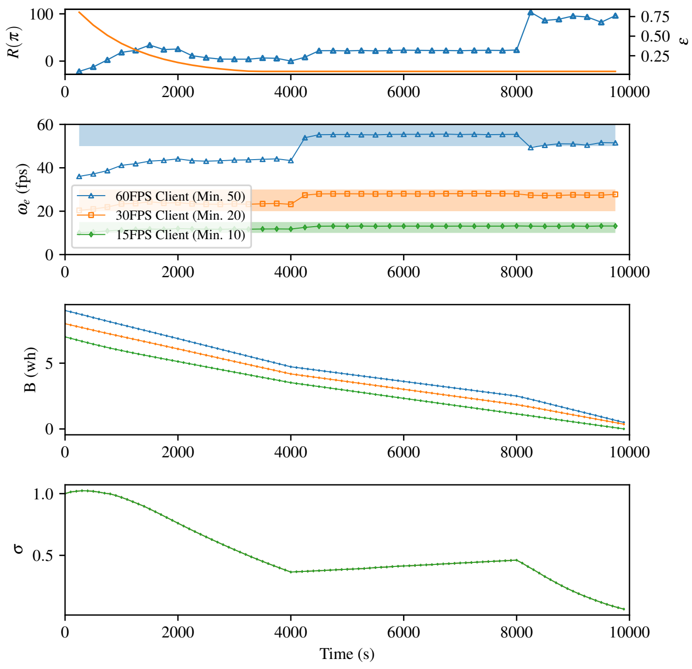

Fig. 2: Results of the simulation of a single cluster and three worker nodes with cloud, regarding, from top to bottom, the reward, the effective frame rate ω_e, the batteries remaining wh and the variance of batteries. We assume that node #1 fails at time 4000. $\alpha = 0.0$

the batteries in the edge nodes quickly approaches zero and, at the end of the simulations, the batteries in the edge nodes are all balanced and the lifespan of the service is very good. We now present the second part, where the value of α is set to 1: The results of the second scenario (Fig. 3), where the value of the parameter α is set to 1, demonstrate the ability of the proposed scheduling algorithm to quickly adapt to changes in the environment and maintain the desired frames per second (FPS) requirements of the clients. In particular, the results show that, for a brief period after the simulated crash failure of worker 1, the algorithm is unable to meet the FPS requirements of the clients. However, the algorithm quickly adapts by offloading the load to the other two worker nodes and the cloud. This allows the algorithm to return to the previous level of performance until timestamp 8000, at which point worker 1 returns to normal operation. However, the increased load on the other two worker nodes causes them to run out of power, resulting in another anomaly in the system. Once again, the algorithm quickly adapts by splitting the tasks between worker 1 and the cloud, allowing it to meet the FPS requirements of the clients. In conclusion, the proposed scheduling algorithm quickly adapts to changes in the environment and maintains the desired FPS requirements of the clients, even in the presence of multiple anomalies.

C. Only Workers

In this setting, we consider an edge node with three workers and a single scheduler. The task can either be executed by

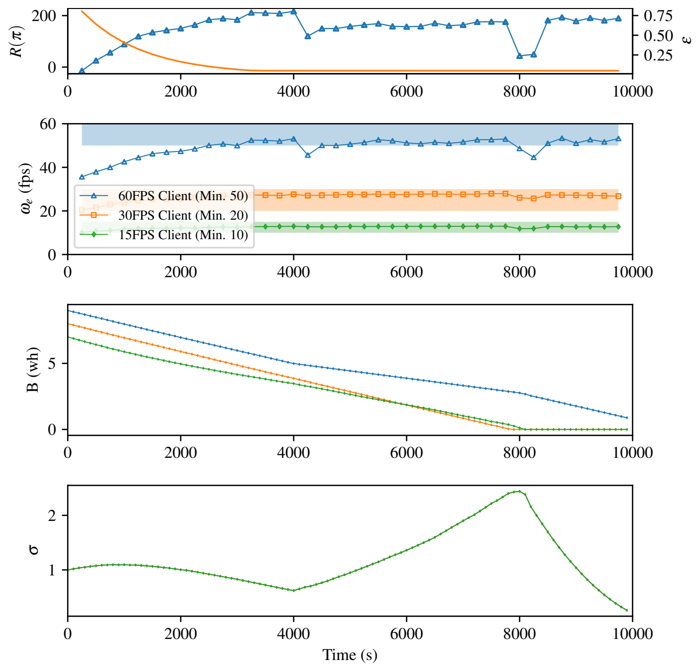

Fig. 3: Results of the simulation of a single cluster and three worker nodes with cloud, regarding, from top to bottom, the reward, the effective frame rate ω_e, the batteries remaining wh and the variance of batteries. We assume that node #1 fails at time 4000. $\alpha = 1.0$

the workers or rejected. We present this simpler case to compare our algorithm with several baselines. Specifically, we compare our algorithm with the least-loaded algorithm, which is the best-known algorithm for meeting deadline requirements. Additionally, to establish a baseline for lifespan, we use the longest lifespan algorithm, which assigns tasks to the worker with the longest expected lifespan. Without cloud involvement, this setting provides a fair comparison. We simulate our algorithm for all values of α between 0 and 1, with a step size of 0.1. Our goal is to demonstrate the efficacy of our algorithm and identify the α that achieves the best trade-off. As another baseline, we also use a random algorithm that selects a worker for each task at random. The performance metrics are illustrated in III-E. Additionally, the values used for this experiment are B_1 = 9 Wh, B_2 = 8 Wh, and B_3 = 7 Wh. The machine speeds are different from those used in 5; specifically, S_1 = 1.8, S_2 = 1.7, and S_3 = 1.4, as without cloud support, it would be impossible to meet FPS requirements with the settings used. In the table, we use the following notations:

- α: Alpha value.
- σ: Variance of battery Wh difference.
- δ: Max Wh - Min Wh when the first node discharges.
- M: Maximum lifespan, m: Minimum lifespan.
- γ : Percentage of jobs meeting requirements.
- γ_i : Percentage of jobs meeting requirements of $type_i$.
- ts: Total service time of all workers.
- LL: Least Loaded algorithm.
- MLIF: Maximum Lifespan algorithm.
- RAND: Random algorithm.

α	$\sigma\downarrow$	$\delta\downarrow$	$m\uparrow$	$M\uparrow$	$\gamma\uparrow$	$\gamma_0\uparrow$	$\gamma_1\uparrow$	$\gamma_2\uparrow$	ts $\uparrow$
0.00	**0.08**	**0.00**	8055	8056	70.9	18.5	94.2	**99.9**	24166
0.20	0.08	0.10	7870	7957	82.2	49.3	98.5	**99.9**	23784
0.30	0.10	0.24	7711	7912	91.3	74.5	**99.4**	**99.9**	23533
0.40	0.14	0.49	7541	7941	91.8	76.7	98.8	**99.9**	23415
0.50	0.14	0.48	7548	7942	94.5	84.9	98.7	**99.9**	23385
0.60	0.14	0.47	7534	7926	**95.6**	**88.7**	98.4	99.9	23325
0.70	0.22	0.88	7348	8077	92.7	84.8	94.2	99.2	23224
0.80	0.31	1.22	7175	8202	90.1	80.4	90.6	99.2	23120
0.90	0.33	1.25	7197	8259	89.0	78.3	89.2	99.5	23097
1.00	0.37	1.38	7110	8306	88.5	78.3	88.3	98.9	23054
LL	0.41	1.48	7167	8384	84.67	64.8	89.2	99.9	23347
MLIF	0.12	**0.00**	**9452**	**9452**	41.1	0.3	25.0	98.1	**28356**
RAND	1.130	2.62	7155	8804	39.39	0.00	34.92	83.25	15959

TABLE I: Results of the experiment in the case of only workers with batteries values B_1 = 9 Wh, B_2 = 8 Wh, and B_3 = 7 Wh.

The results of our experiments are quite intriguing and provide valuable insights into the performance of our proposed scheduling algorithm. In particular, we can observe the significant impact of the parameter α on the results. Our algorithm outperforms the least loaded algorithm for values of α greater than 0.3. This can be attributed to the fact that the least loaded algorithm is not energy-aware and tends to quickly drain the batteries of the worker nodes with more resources. In this case, the value of α equal to 0.3 is the most suitable choice, as it allows for a good balance of the batteries in the edge nodes throughout the simulation, as well as a good lifespan of the service. Additionally, the algorithm is able to maintain the frames per second (FPS) requirements of the clients. It is worth noting that the worker nodes, in this case, die at approximately the same time. While our algorithm does not match the performance of the maximum lifespan algorithm, it offers better balancing with respect to maximizing lifespan. The second case, as shown in II, is even more interesting. In this case, the worker node with the smallest battery is assigned to the most powerful worker: In this case, our algorithm performs exceptionally well, significantly outperforming LL algorithm. This improvement is particularly notable because worker 1 depletes its battery at 6669, failing to meet the

α	$\sigma\downarrow$	$\delta\downarrow$	$m\uparrow$	$M\uparrow$	$\gamma\uparrow$	$\gamma_0\uparrow$	$\gamma_1\uparrow$	$\gamma_2\uparrow$	ts $\uparrow$
0.00	**0.08**	**0.00**	8070	8070	63.4	4.3	86.1	99.9	24210
0.10	**0.08**	**0.00**	8049	8050	63.3	3.7	86.1	99.9	24149
0.20	**0.08**	**0.00**	8019	8020	63.2	3.2	86.6	99.9	24058
0.30	**0.08**	**0.00**	7996	7996	64.5	4.0	89.4	99.9	23988
0.40	**0.08**	**0.00**	7950	7951	71.1	17.4	95.8	**100**	23851
0.50	**0.08**	0.06	7812	7862	89.9	70.3	**99.4**	99.9	23534
0.60	0.24	1.08	7177	8036	92.0	78.6	97.4	99.9	23124
0.70	0.23	1.05	7208	8046	**92.4**	**80.1**	97.2	99.9	23155
0.80	0.39	1.52	6935	8149	90.5	74.7	96.9	99.9	23068
0.90	0.39	1.55	7029	8238	87.7	73.3	90.3	99.3	23066
1.00	0.51	1.75	6968	8363	86.9	72.7	89.5	98.6	23060
LL	0.86	2.47	6669	8574	77.9	53.6	81.5	98.7	22872
MLIF	0.12	**0.00**	**9433**	**9434**	37.7	0.3	16.3	96.4	**28300**
RAND	0.42	1.31	7863	9197	46.5	0.0	46.7	92.9	25864

TABLE II: Results of the experiment in the case of only workers with batteries values B_1 = 7 Wh, B_2 = 8 Wh, and B_3 = 9 Wh.

frame per second (fps) requirements from that point onward due to the lack of battery consideration. The results for lower values of alpha are not as favorable as before because the task allocation to slower workers is aimed at balancing battery usage. However, for alpha values greater than 0.5, the performance in terms of fps is excellent, and the battery usage is well balanced. The figure 4 clearly demonstrates that the fps requirements are met as long as the workers remain operational, providing a good service time. Additionally, the workload is almost perfectly balanced among the workers.

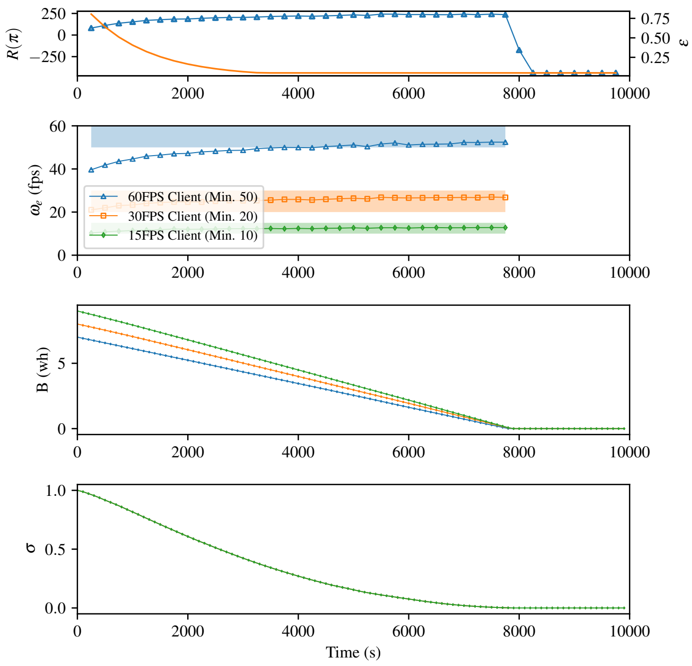

Fig. 4: Results of the simulation of a single cluster and three worker nodes, regarding, from top to bottom, the reward, the effective frame rate ω_e, the batteries remaining wh and the variance of batteries. We assume that $\alpha = 0.5$

D. Edge-to-Cloud continuum

Now, we present the results in the target environment where tasks can be sent to the cloud in addition to being assigned to workers. The task flow and worker specifications are illustrated in Figure 5. In the first setting the batteries are the same of first expirement of before, now the LL algorithm chage to LLAC (Least loaded aware cloud), because is aware of cloud and send longest task to it. We can see the result in table III In this scenario, the results of our algorithm are remarkable. It surpasses both baselines—Least Loaded for FPS and Maximum Lifespan. Notably, the best performance is achieved at $\alpha = 0.3$, where the service time is maximized due to extensive use of the cloud while still meeting FPS requirements. For a detailed visualization, refer to Figure 6. We can appreciate the variance curve and the battery load at the bottom of the plot. Additionally, it is evident how the algorithm remains very stable after reaching convergence. As observed previously, it is quite intriguing how switching the batteries between the

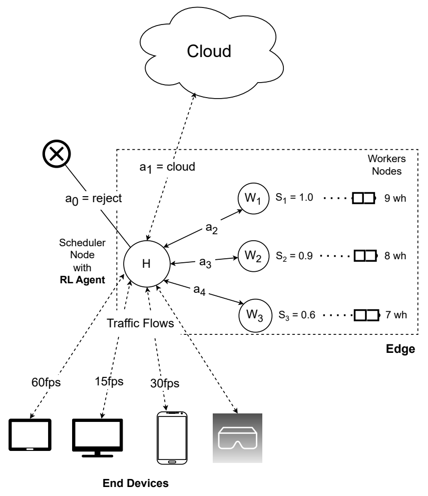

Fig. 5: The setting of the of the experiments on a single cluster with three workers nodes, cloud, and four traffic flows V-D.

α	$\sigma\downarrow$	$\delta\downarrow$	$m\uparrow$	$M\uparrow$	$\gamma\uparrow$	$\gamma_0\uparrow$	$\gamma_1\uparrow$	$\gamma_2\uparrow$	ts ↑
0.00	0.13	**0.00**	8545	8546	65.2	25.2	81.9	90.6	25636
0.10	0.14	**0.00**	8877	8877	72.6	45.3	83.0	89.4	26631
0.20	0.16	**0.00**	9200	9201	83.7	62.1	91.4	97.2	27602
0.30	0.17	0.01	**9462**	**9471**	92.8	80.4	**97.8**	99.9	**28403**
0.40	0.17	**0.00**	9380	9385	**93.2**	**81.9**	**97.8**	**100**	28149
0.50	0.17	**0.00**	9076	9077	92.3	79.4	97.6	100	27230
0.60	0.15	**0.00**	8715	8717	90.9	75.6	97.3	99.9	26147
0.70	0.15	0.20	8261	8436	88.9	71.2	95.6	**100**	25069
0.80	0.21	0.57	8064	8577	89.5	72.4	96.3	99.8	24961
0.90	0.29	1.08	7590	8652	92.0	79.1	97.1	99.9	24727
1.00	0.32	1.19	7517	8671	90.6	75.1	96.8	99.8	24389
LLAC	0.20	1.1	6272	7081	88.3	70.9	93.9	99.9	20221
MLIF	**0.11**	**0.0**	9218	9219	16.0	0.0	0.1	48.0	27656
RAMD	1.10	3.1	5635	8669	29.9	0.0	15.1	74.6	21672

TABLE III: Results of the experiment in the case of three workers with batteries values B_1 = 9 Wh, B_2 = 8 Wh, and B_3 = 7 Wh, and cloud avaible.

quickest and slowest workers yields significant changes. In the latest experiment (refer to Table IV), we observe results similar to previous ones but significantly improved, thanks once again to the integration of cloud computing. It becomes evident that assigning smaller batteries to the quickest workers yields better results for values greater than 0.5. Conversely, in the switched case, we notice good results even for smaller values, especially when aiming for a tradeoff. The selection of alpha is closely tied to the specific environment under consideration; hence, determining an optimal value beforehand is not feasible. Instead, it must be dynamically adapted in real-time based on the prevailing conditions.

VI. Conclusions

In this paper, we continued a previous work [4] by incorporating energy constraints and aiming to maximize battery management and frames per second (FPS) management in the edge or Fog-to-Cloud Computing Continuum model. Our approach utilizes Reinforcement Learning to solve the online task scheduling problem in this dynamic context, which is well-suited for addressing challenges such as heterogeneity

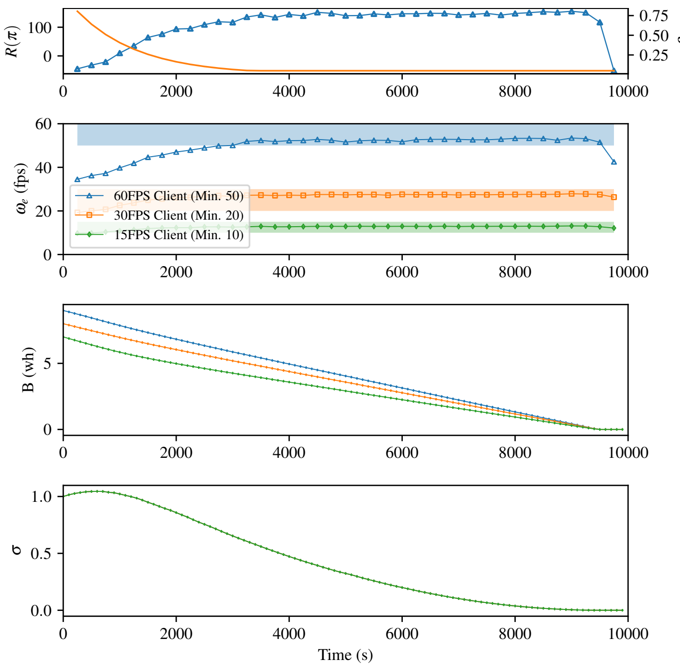

Fig. 6: Results of the simulation of a single cluster, three worker nodes and cloud, regarding, from top to bottom, the reward, the effective frame rate ω_e, the batteries remaining wh and the variance of batteries. We assume that $\alpha = 0.3$

α	$\sigma \downarrow$	$\delta \downarrow$	$m \uparrow$	$M \uparrow$	$\gamma \uparrow$	$\gamma_0 \uparrow$	$\gamma_1 \uparrow$	$\gamma_2 \uparrow$	$ts \uparrow$
0.00	0.07	**0.00**	8009	8009	55.5	7.2	76.5	86.2	24027
0.10	**0.06**	**0.00**	7935	7936	52.9	5.6	71.7	83.6	23807
0.20	0.07	**0.00**	7980	7981	61.4	11.3	79.6	91.2	23941
0.30	**0.06**	**0.00**	7903	7904	73.5	27.3	92.0	99.6	23711
0.40	**0.06**	**0.00**	7713	7714	78.0	42.0	92.0	99.8	23140
0.50	0.15	0.42	8318	8631	88.0	66.5	97.4	**100**	25579
0.60	0.38	1.23	7639	8753	**90.7**	**74.6**	**97.6**	**100**	25109
0.70	0.85	2.49	7017	8955	88.9	71.3	95.4	**100**	23925
0.80	1.28	2.99	6977	9416	89.0	70.9	96.2	99.9	24057
0.90	1.34	3.13	6847	9321	89.1	71.8	95.6	**100**	23740
1.00	1.46	3.19	6824	**9536**	89.4	72.5	95.7	**100**	23916
LLAC	0.77	2.79	5570	7576	66.0	46.5	64.2	87.4	19887
MLIF	0.11	**0.00**	**9183**	9184	11.3	0.0	0.1	33.8	**27551**
RAND	0.14	0.72	6699	7363	27.0	0.0	20.7	60.3	21299

TABLE IV: Results of the experiment in the case of three workers with batteries values $B_1 = 7$ Wh, $B_2 = 8$ Wh, and $B_3 = 9$ Wh, and cloud avaible.

of nodes, difficulties in estimating the real execution speed of nodes, possible node failures, cooperation strategies, and different QoS requirements (e.g. minimum frame rate, service time). We presented the results of our approach in both only workers and cloud environment, demonstrating that in either case, the agent placed in the scheduler can derive the best scheduling policy without any prior knowledge of the characteristics of the worker nodes or neighboring clusters. It needs only to load and battery information. Future work includes implementing and testing our solution on a real system rather than a simulation environment and exploring the use of more advanced reinforcement learning techniques, such as deep neural networks and policy gradient methods. It is also important to consider whether a more complex algorithm would lead to better performance or if the added complexity would not be suitable for a real-time scheduling system.

Acknowledgment

This work was partially supported by project no. 202277 WMAE CUP B53D23012820006, "EdgeVision against Varroa (EV2): Edge computing in defence of bees" funded by the Italian's MUR PRIN 2022 (ERC PE6) research program and by the European Union – Next Generation EU. This work was also partially supported by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU.

References

[1] M. Iorga, L. Feldman, R. Barton, M. J. Martin, N. S. Goren, and C. Mahmoudi, "Fog computing conceptual model," 2018.

[2] D. Kimovski, R. Mathá, J. Hammer, N. Mehran, H. Hellwagner, and R. Prodan, "Cloud, fog, or edge: Where to compute?" *IEEE Internet Computing*, vol. 25, no. 4, pp. 30–36, 2021.

[3] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[4] G. P. Mattia and R. Beraldi, "Leveraging reinforcement learning for online scheduling of real-time tasks in the edge/fog-to-cloud computing continuum," in *2021 IEEE 20th International Symposium on Network Computing and Applications (NCA)*. IEEE, 2021, pp. 1–9.

[5] Y. Sui and S. Song, "A multi-agent reinforcement learning framework for lithium-ion battery scheduling problems," *Energies*, vol. 13, no. 8, p. 1982, 2020.

[6] X. Xiong, K. Zheng, L. Lei, and L. Hou, "Resource allocation based on deep reinforcement learning in iot edge computing," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 6, pp. 1133–1146, 2020.

[7] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji, and M. Bennis, "Performance optimization in mobile-edge computing via deep reinforcement learning," in *2018 IEEE 88th Vehicular Technology Conference (VTC-Fall)*. IEEE, 2018, pp. 1–6.

[8] S. Sheng, P. Chen, Z. Chen, L. Wu, and Y. Yao, "Deep reinforcement learning-based task scheduling in iot edge computing," *Sensors*, vol. 21, no. 5, p. 1666, 2021.

[9] Y. Wei, Z. Zhang, F. R. Yu, and Z. Han, "Joint user scheduling and content caching strategy for mobile edge networks using deep reinforcement learning," in *2018 IEEE International Conference on Communications Workshops (ICC Workshops)*. IEEE, 2018, pp. 1–6.

[10] S. Huang, B. Lv, R. Wang, and K. Huang, "Scheduling for mobile edge computing with random user arrivals—an approximate mdp and reinforcement learning approach," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 7, pp. 7735–7750, 2020.

[11] Y. Zhang, Z. Zhou, Z. Shi, L. Meng, and Z. Zhang, "Online scheduling optimization for dag-based requests through reinforcement learning in collaboration edge networks," *IEEE Access*, vol. 8, pp. 72 985–72 996, 2020.

[12] A. Luckow, K. Rattan, and S. Jha, "Exploring task placement for edge-to-cloud applications using emulation," in *2021 IEEE 5th International Conference on Fog and Edge Computing (ICFEC)*. IEEE, 2021, pp. 79–83.

[13] J. Banks, J. S. Carson, B. L. Nelson, and D. M. Nicol, "Discrete-event system simulation," 1995. [Online]. Available: https://api.semanticscholar.org/CorpusID:122566976

[14] A. A. Sherstov and P. Stone, "Function approximation via tile coding: Automating parameter choice," in *International symposium on abstraction, reformulation, and approximation*. Springer, 2005, pp. 194–205.