The 20th IEEE International Symposium on Network Computing and Applications (NCA 2021)
23 - 26 November, 2021
Virtual

Leveraging Reinforcement Learning for online scheduling of real-time tasks in the Edge/Fog-to-Cloud computing continuum

Gabriele Proietti Mattia, Roberto Beraldi

Department of Computer, Control and Management Engineering “Antonio Ruberti”, Sapienza University of Rome, Italy

proiettimattia@diag.uniroma1.it • gpm.name

SAPIENZA
UNIVERSITÀ DI ROMA

DIAG
Dipartimento di Ingegneria
informatica, automatica e gestionale
Antonio Ruberti

Outline

1. **Context and Challenge**
2. **System Model**
3. **Reinforcement Learning for online scheduling**
4. **Results**
5. **Conclusions**

1

Introduction

Context

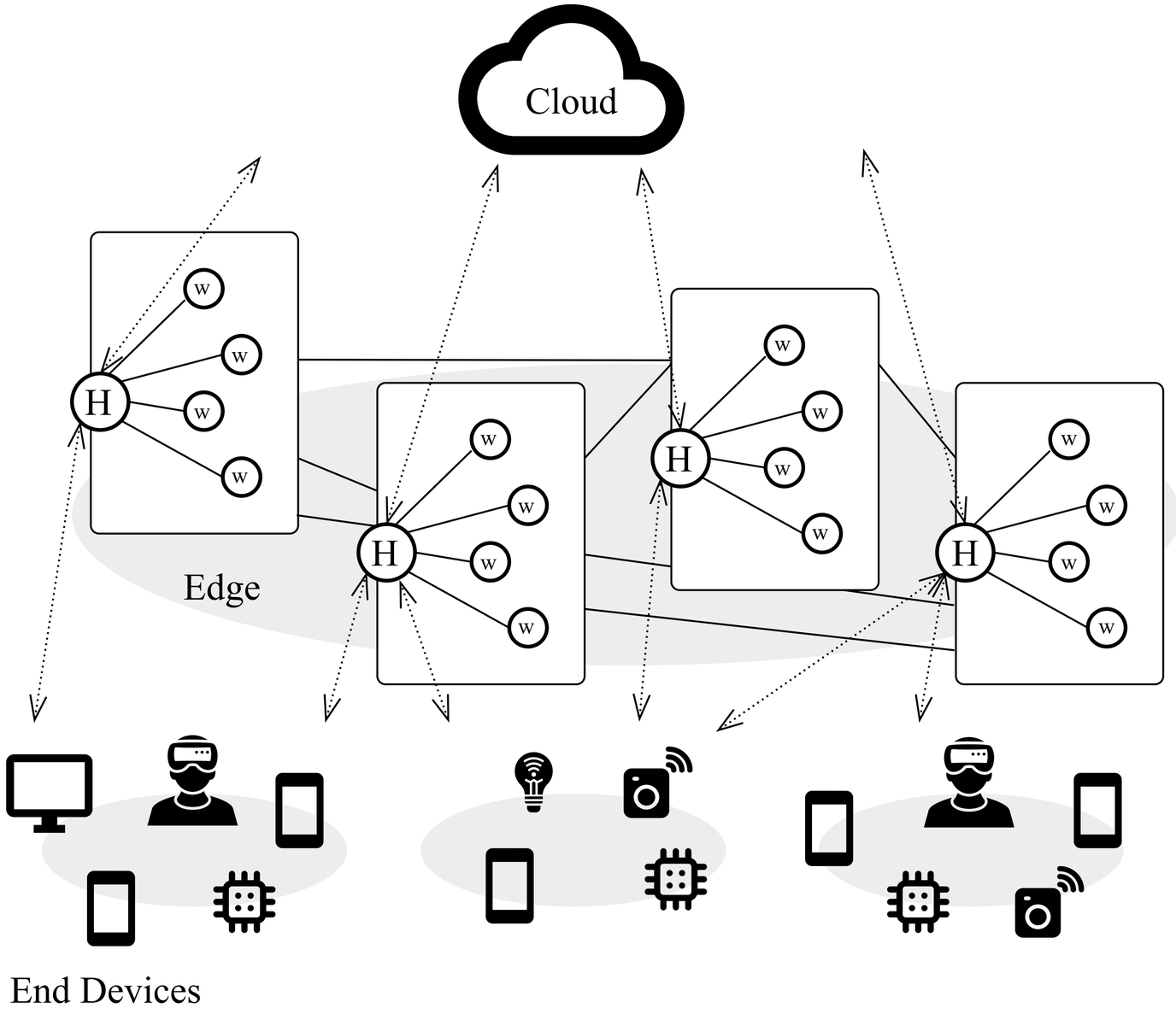


Figure 1.1 The Edge-to-Cloud continuum environment considered as computing scenario

The challenge

We now focus on the design a single cluster, we envisioned:

- a **scheduler** (H) node which receives the tasks requests and chooses which worker should execute them;
- a set of heterogeneous **workers** (w_i) nodes which execute tasks requests.

In the paradigm of Edge/Fog Computing devices are heterogeneous and it is not easy to **estimate** the exact computational power of a worker node, because it can vary over time, moreover our scheduler must **react** if a fail suddenly happen.

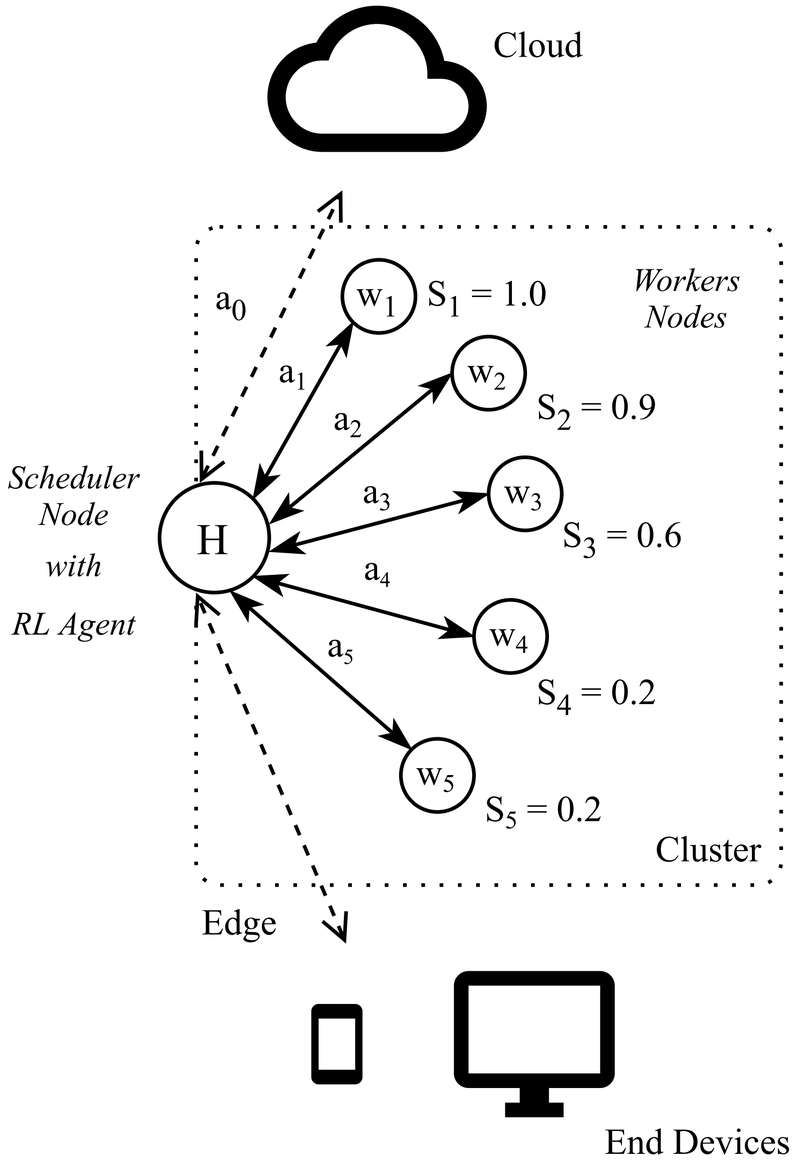


Figure 1.2 An hypothetical single cluster configuration

Objectives

The purpose of our work is to design a scheduling algorithm based on Reinforcement Learning:

- which is able to cope with **node heterogeneity**
- which does not know in advance the **computational power** of the worker nodes
- which is able to satisfy **precise application requirements** in terms of service requests rate (e.g. fps)
- which is able to perform an **online decision making** (per-task request)
- which can decide to forward a task to a **local worker** in the cluster, to the **cloud** or to **another cluster** (*computing continuum*)

The designed scheduling algorithm is then run on simulation.

State-of-the-art

- X. Xiong et al. in "*Resource allocation based on deep reinforcement learning in IoT edge computing*" present a deep reinforcement approach for resource allocation in a MEC system **but the allocation scheme is based on time slices**
- X. Chen et al. in "*Performance optimisation in mobile-edge computing via deep reinforcement learning*" focus on base stations that must be selected by the client in a ultra high-dense network
- S. Sheng et al. in "*Deep reinforcement learning-based task scheduling in iot edge computing*" specifically studies the task scheduling in the edge computing by using the approach uses the deep reinforcement learning **but the scheduling is not done online**
- S. Huang et al. "Scheduling for mobile edge computing with random user arrivals — an approximate mdp and reinforcement learning approach" studies the scenario in which mobiles can appear randomly in a cell
- Y. Zhang et al. "Online scheduling optimization for dag-based requests through reinforcement learning in collaboration edge networks" focuses on online scheduling and using time differential learning, **but the tasks do not have to meet a deadline**
- A. Luckow et al. "Exploring task placement for edge-to-cloud applications using emulation" introduces a specific study on the task placement in the edge-to-cloud computing continuum

2

System Model

RL Rationale

The idea is to make each scheduler of a cluster a learner agent, model the problem as a Markov Decision Process and solve the learning task with a RL framework. All of the following entities must be defined.

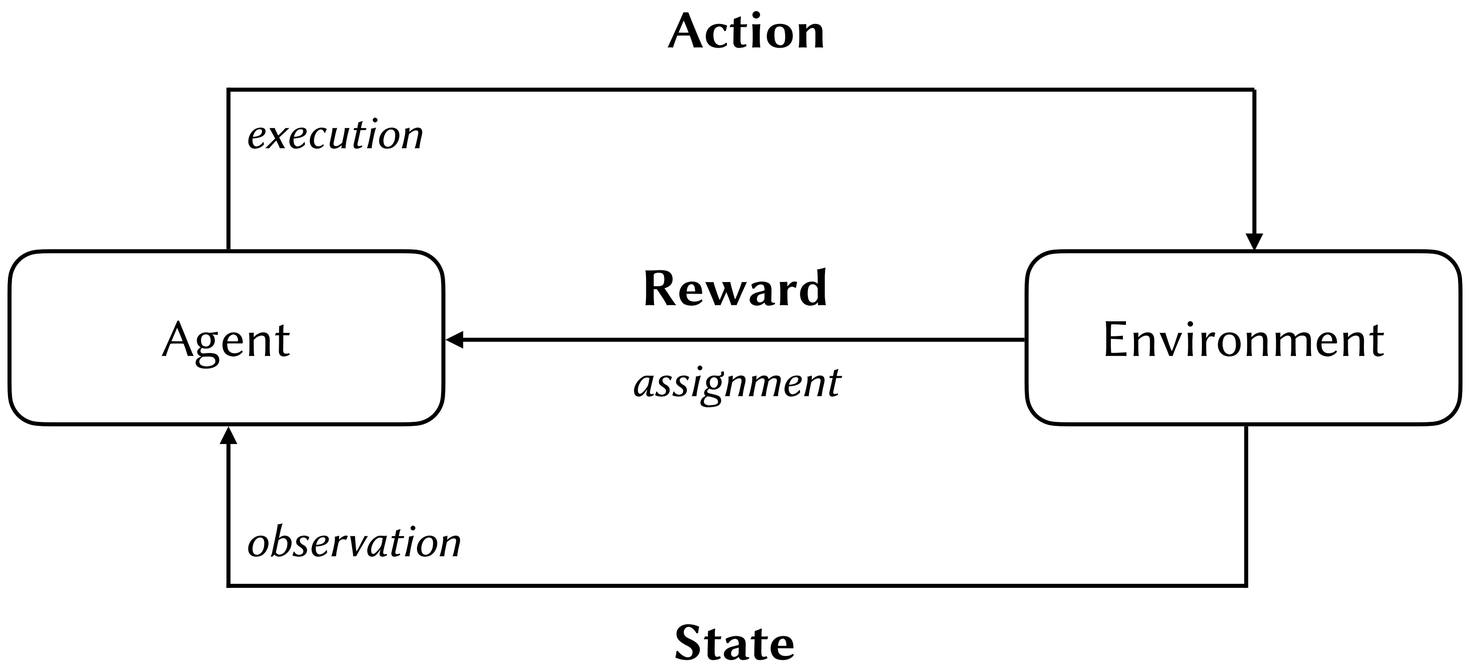


Figure 2.1 The classic Markov Decision Process representation

Tasks and delay model

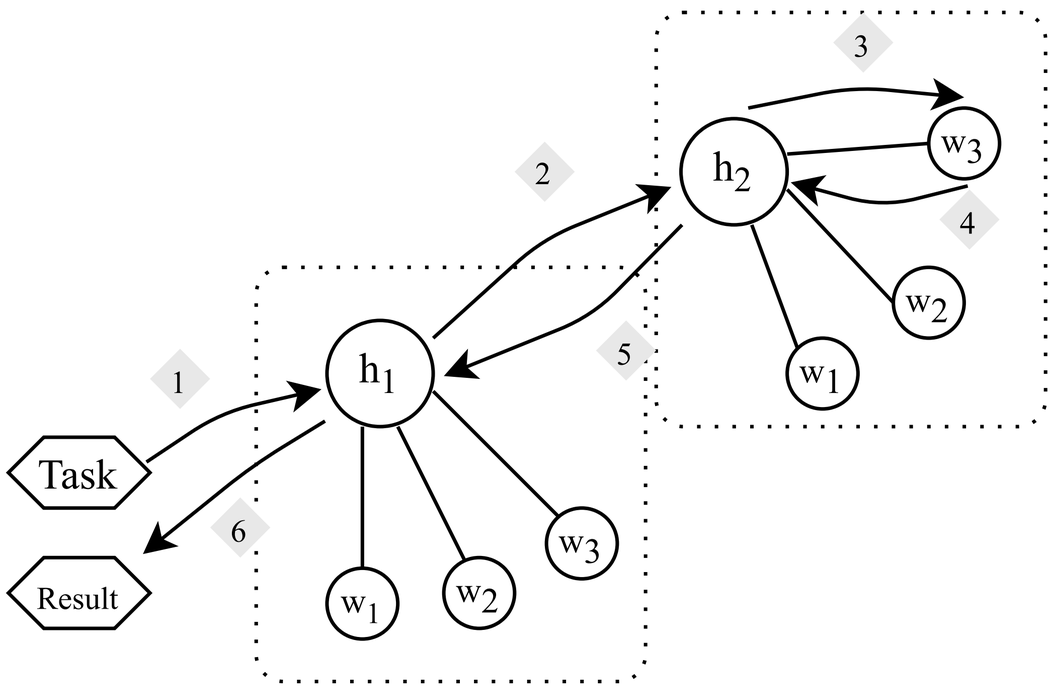


Figure 2.2 An hypothetical execution path of a frame task

In our model, each task is the **processing a frame**. We have different users to which corresponds a traffic flow tf_i that is periodic because it follows the frame generation.

Simplistically to every user i corresponds:

- ω_n the rate of frame generation
- ω_m the minimum response rate that the cloud continuum must satisfy
- ω_e the effective processing rate of the cloud continuum

During the entire execution path we simulate the **delay of every hop** by using different values of the bandwidth

Actions

Upon the arrival of a task execution request to the cluster *i* the action that can be performed by the scheduler is a scheduling action, one of the following:

- reject
- forward to cloud
- forward to worker 1
- ...
- forward to worker n
- forward to neighbour cluster 1
- ...
- forward to neighbour cluster m

$\mathcal{A}_i$

$\mathcal{A}'_i$

The actions are grouped in **two sets** because we first test only one cluster and then more clusters

When a task is forwarded we wait for its completion in order to derive the **reward**.

Reward

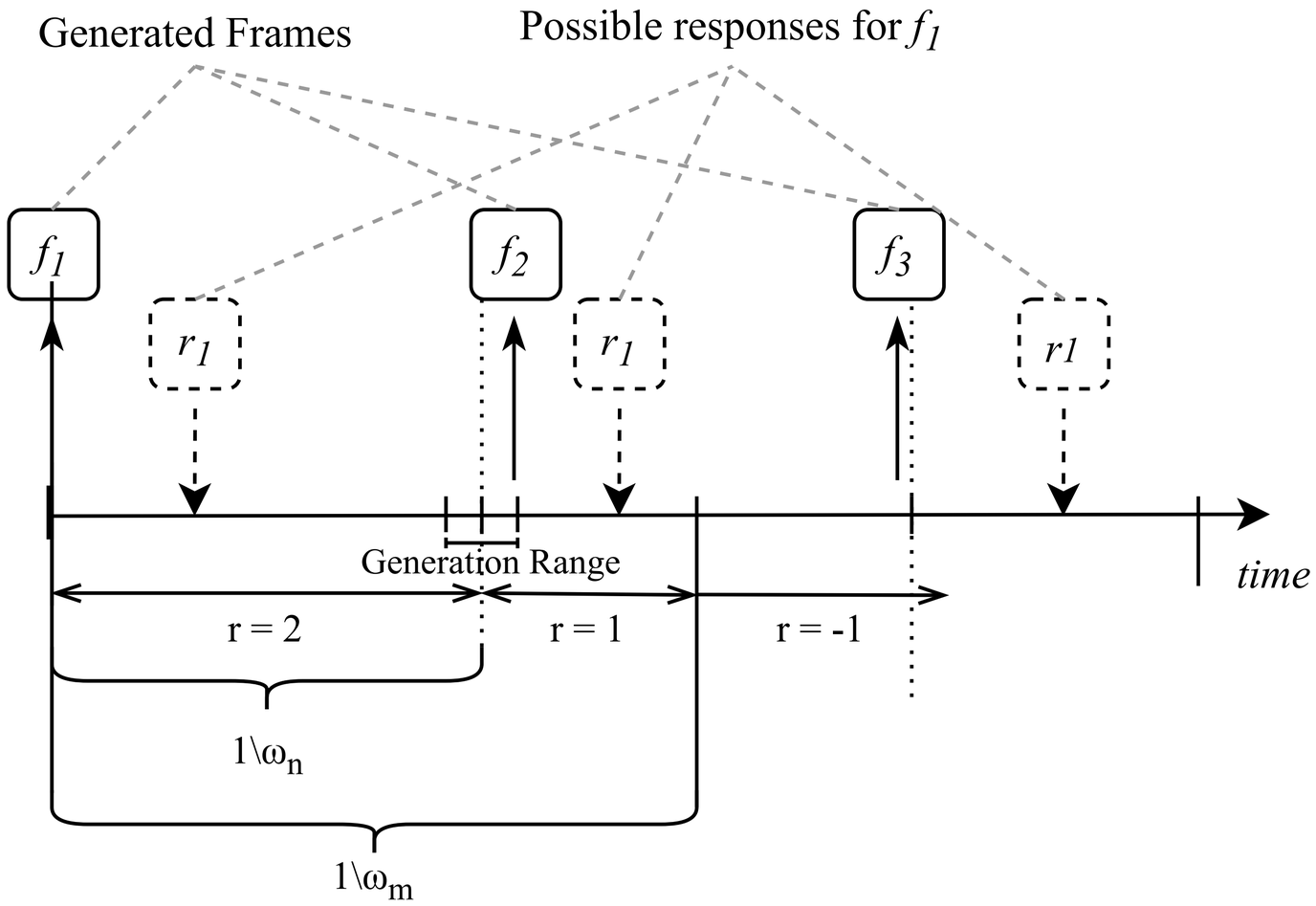


Figure 2.4 Diagram that illustrates how reward is assigned when the task is executed and the frame f_1 returns to client after being processed (r_1).

Performance Parameters

The performance parameters that we used are the following:

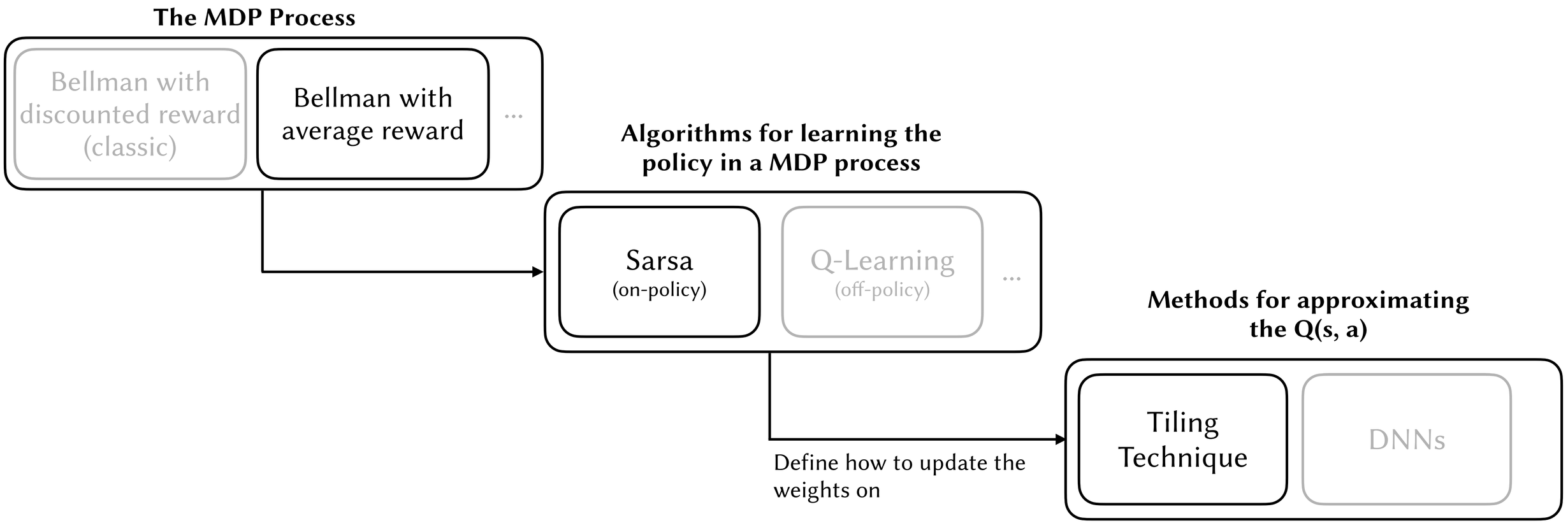
- **total reward** (R), as defined earlier
- the **effective frame rate** (ω_e) measured in frames-per-second and computed as the sum of the total number of frames successfully processed every second (not rejected)
- the **total response time** (d_t) measured in milliseconds and computed as the average response time of all the tasks finished every second
- the **lag time** (τ) measured in milliseconds and computed as:

$$\tau = d_t - 1/\omega_n$$

3

Reinforcement Learning for online scheduling

RL Theoretical Stack



Sutton, Richard S., and Andrew G. Barto. *Reinforcement learning: An introduction.* MIT press, 2018.

Time Differential Sarsa w/ avg. Reward

Decisions are taken by approximating the *q(s,a)* action-value function, that returns the value of an action *a* given the state *s*. For approximating the q(s,a) function we can take into consideration the difference between $q(s,a)_t$ and $q(s,a)_{t+1}$ that is defined as (supposed we are at t+1):

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

The **immediate** reward, right after executing the task

The current **average** reward

The value of the action chosen in the current state by using the same policy

The value of the previous action (that lead to S_{t+1}) in the previous state S_t

4 Results

Single Cluster

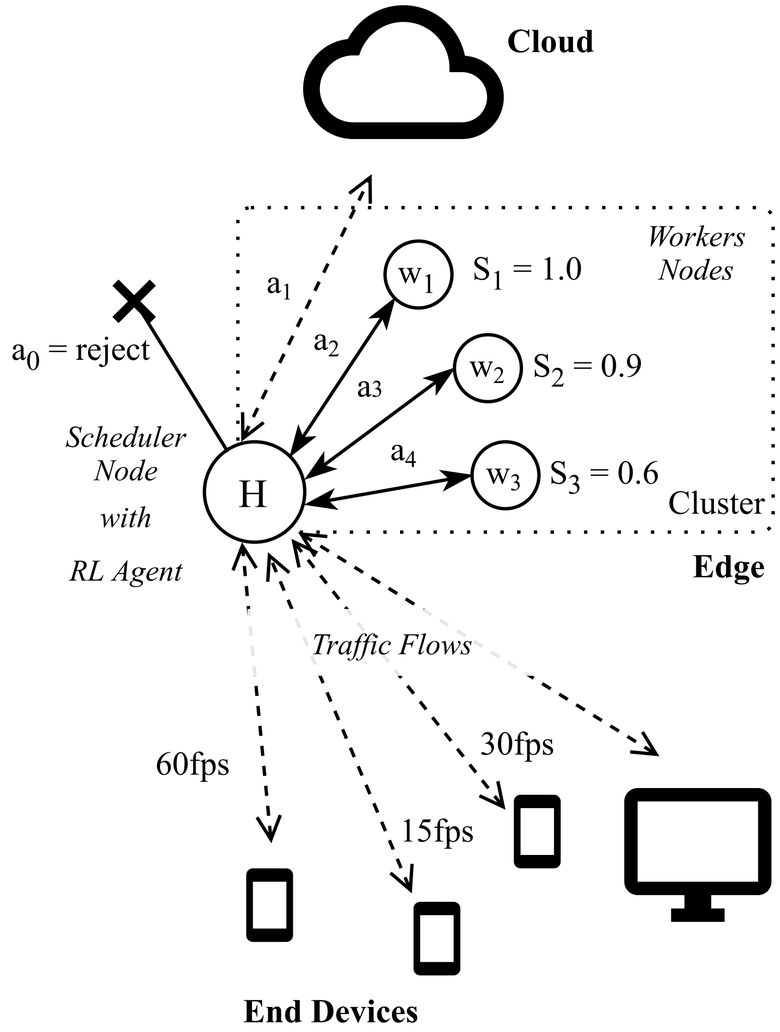


Figure 4.1 Single cluster experiment configuration

Brand name	Frequency	Parallelism	S	K
Odroid-C4	2.0 GHz	4 cores	1.0	4
Asus Tinker	1.8 GHz	4 cores	0.9	4
Rock Pi N10	1.4 GHz	4 cores	0.7	4
Raspberry Pi 3	1.2 GHz	4 cores	0.6	4

Table 4.1 The used devices speeds

	ω_n	ω_m	Distr.	σ	d_e	Distr.	σ	Payload
tf_1	60 fps	50 fps	G. Periodic	0.001	10 ms	Gauss.	0.0003	50 kb
tf_2	30 fps	20 fps	G. Periodic	0.002	20 ms	Gauss.	0.0003	50 kb
tf_3	15 fps	10 fps	G. Periodic	0.01	55 ms	Gauss.	0.0003	50 kb
tf_4	10 fps	-	Exp.	-	100 ms	Gauss.	0.0003	50 kb

Table 4.2 The list of the traffic flows for the experiment with one cluster

Single Cluster

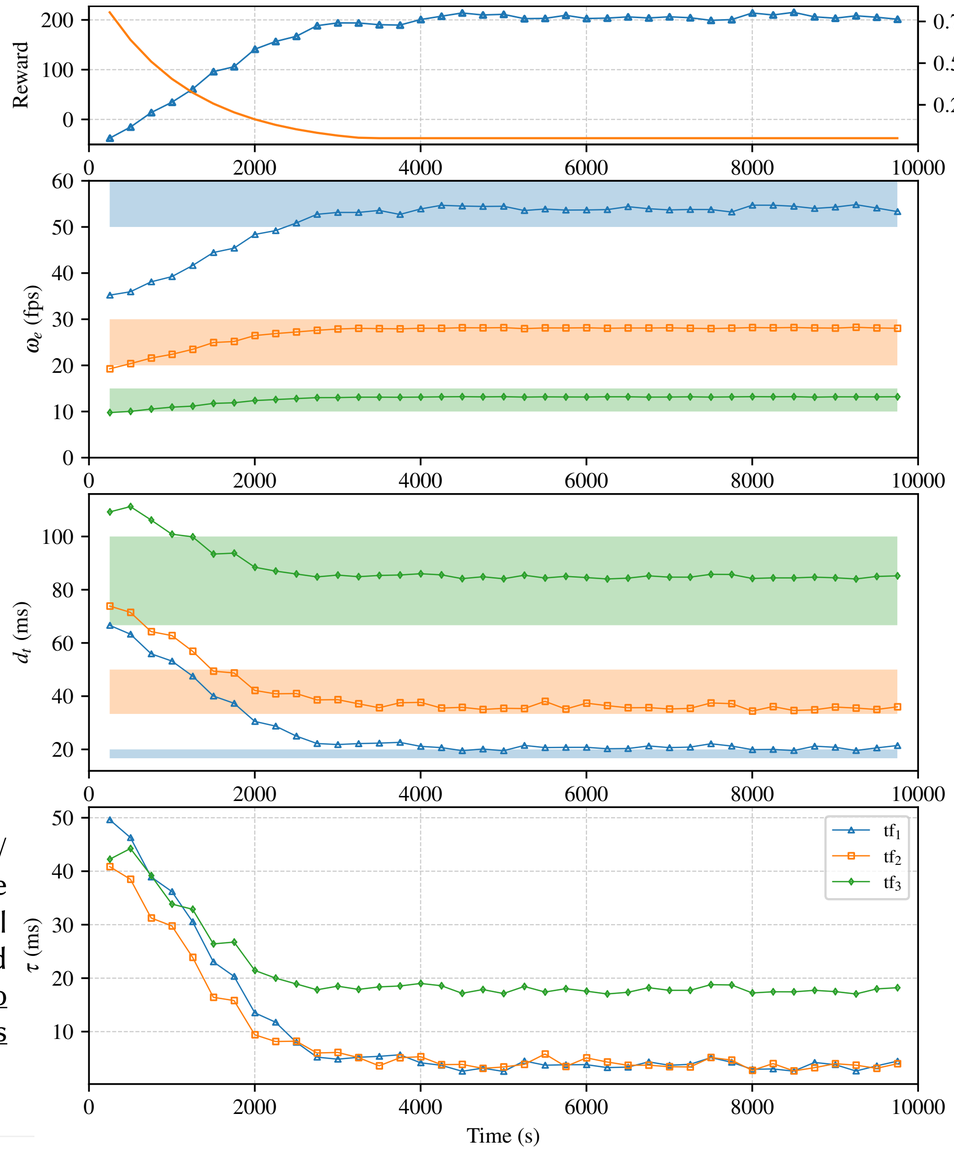
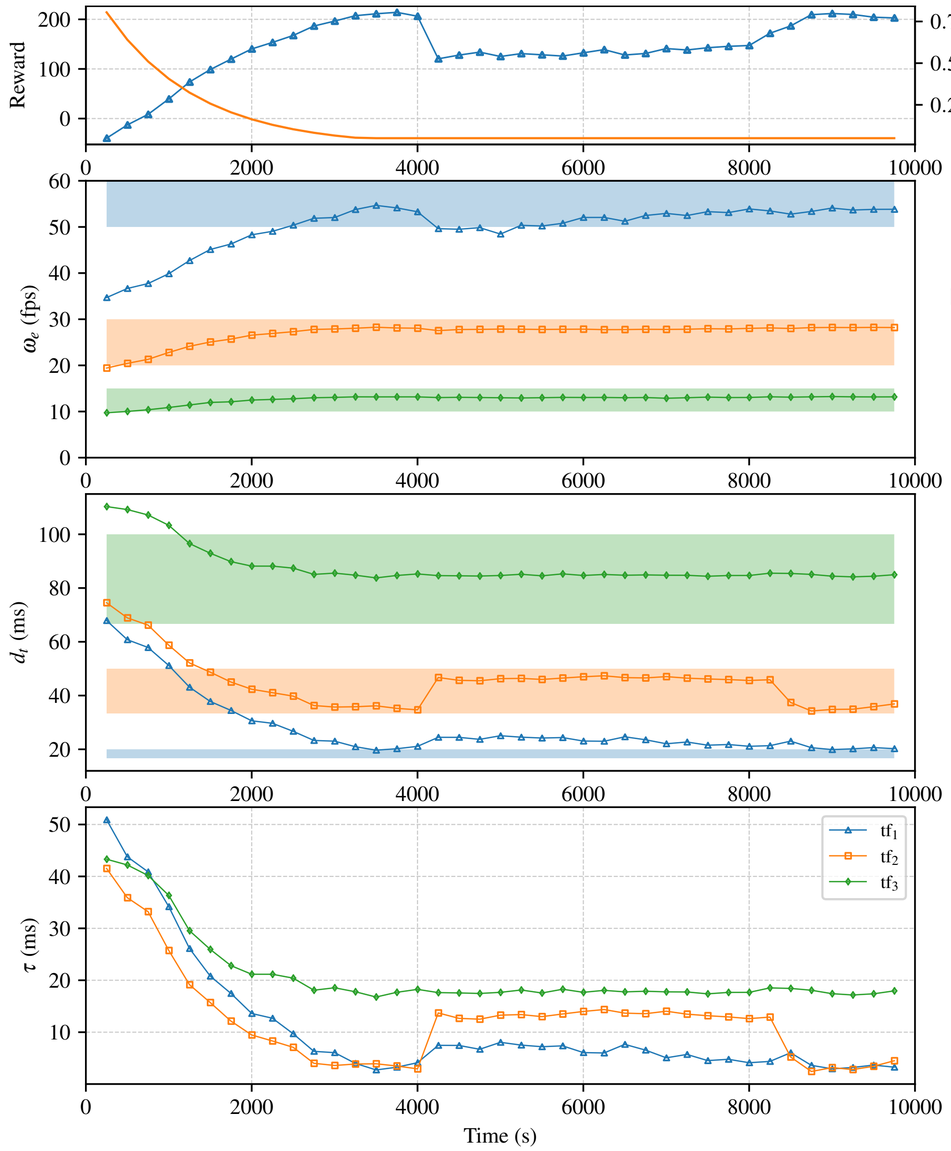


Figure 4.1 Reward/s, effective frame rate ω_t, total response time d_t and lag time τ with no failures



Coloured ranges represent the user/ application requirements

Figure 4.1 Reward/s, effective frame rate ω_t, total response time d_t and lag time τ with failures: we assume that node #1 fails at time 4000 for 4000 seconds

Single Cluster

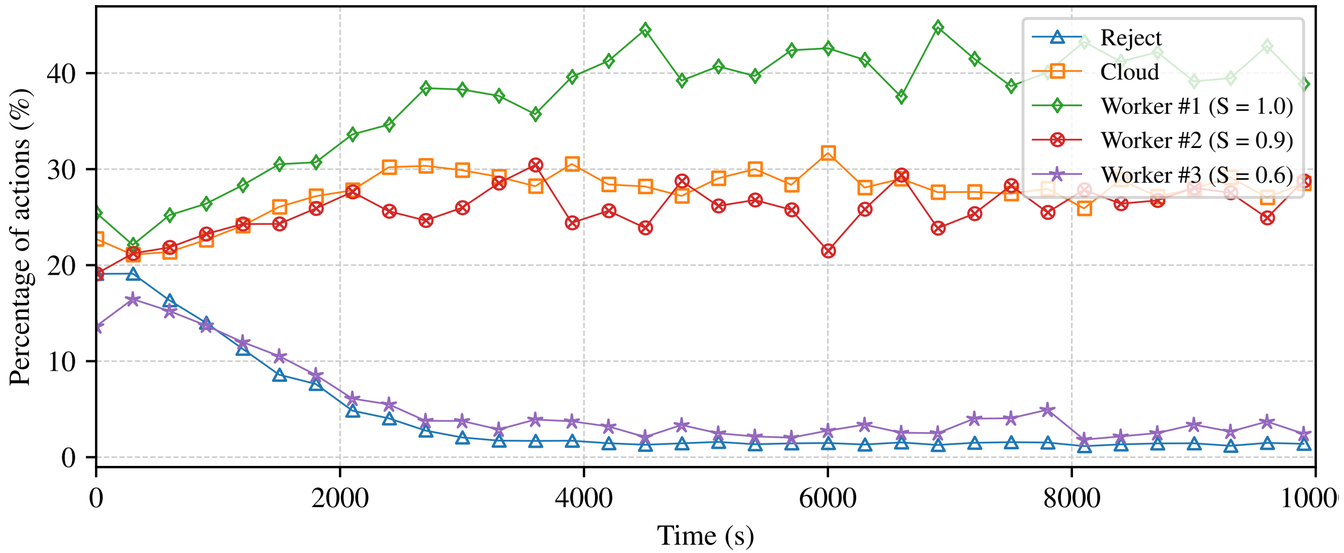


Figure 4.3 Actions distribution over time in case of no failures

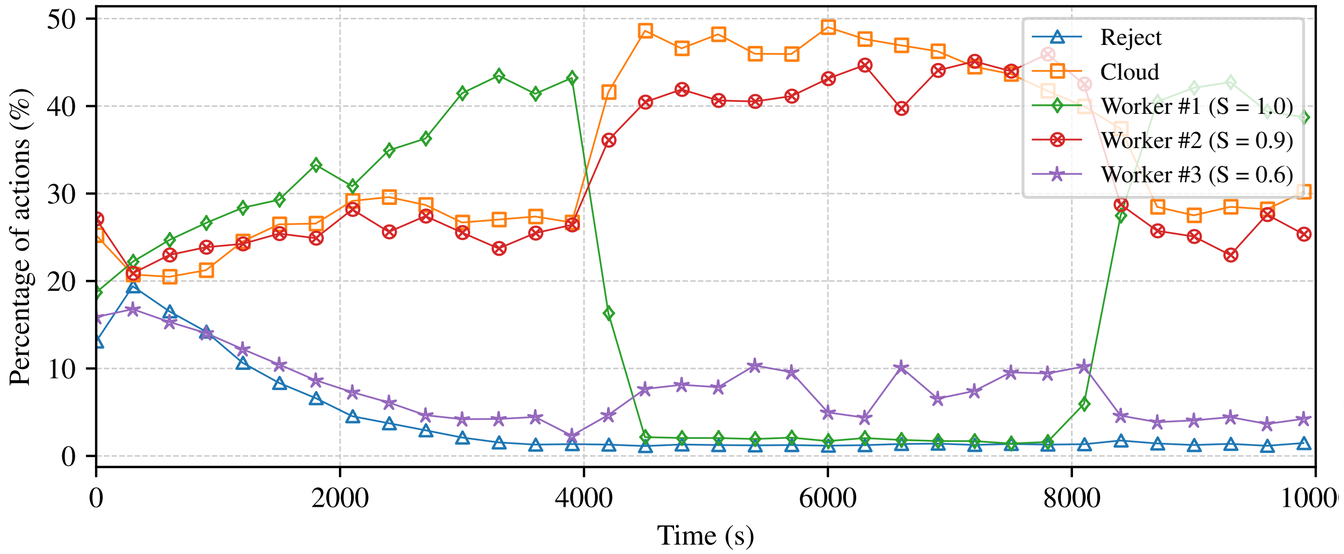


Figure 4.4 Actions distribution over time in case of failures: we assume node #1 to fail at time 4000 for 4000s

Single Cluster

	Sarsa Trained			*Least Loaded*			*Random*		
	ω_e	τ	d_t	ω_e	τ	d_t	ω_e	τ	d_t
tf_1	54.08	19.63	20.57	37.85	48.75	96.05	29.61	65.23	100.71
tf_2	28.15	36.00	35.69	19.04	72.22	110.50	15.72	88.35	109.64
tf_3	13.17	76.34	84.71	9.52	120.35	148.80	8.11	142.38	144.25

Table 4.3 Comparison regarding the effective frame rate (ω_e), the lag time (τ) and the total response time (d_t) between our algorithm "Sarsa Trained" and other two approaches: scheduling to the least loaded node and random scheduling.

Multiple clusters

In the multiple clusters setting we suppose that schedulers of clusters can decide to forward other clusters without knowing nothing about them.

We suppose to have **three** clusters:

- cluster #1 has three nodes with speeds 1.0, 0.9 and 0.6;
- cluster #2 has two nodes with speeds 0.9 and 0.6;
- cluster #3 has three nodes with speeds 1.0, 0.7 and 0.6.

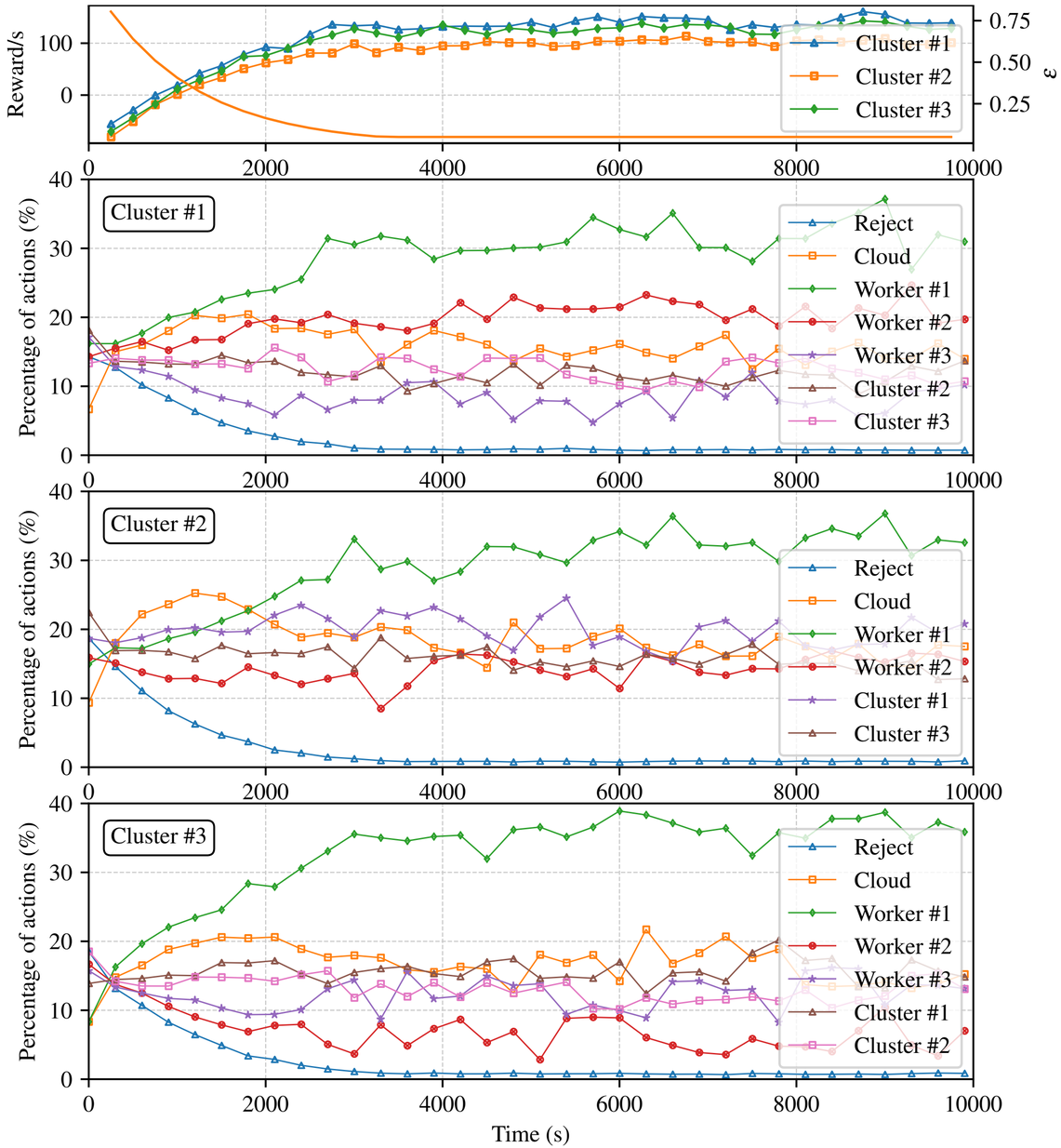


Figure 4.5 Reward/s and actions distribution over time in the multi-clusters setting

5

Conclusions

Conclusions & Future Work

- in the presented work we **designed** and r**un in simulation** a reinforcement learning based algorithm for dealing with online scheduling in the edge/fog-to-cloud computing continuum, addressing the problems of heterogeneity, failures, online decision making and **application requirements** (QoS)
- we firstly tested a single cluster environment and then a multi-cluster one even comparing the solution with other scheduling strategies

Future work

- consider nodes **speed** not fixed over time
- investigate the impact of **frame skipping** when frames returns in different order with respect the generation one - case Figure 2.3 (d)
- consequences in scalability of the increasing in the **number of task types**

Leveraging Reinforcement Learning for online scheduling of real-time tasks in the Edge/Fog-to-Cloud computing continuum

Gabriele Proietti Mattia*, Roberto Beraldi*

TALK & PRESENTATION

Gabriele **Proietti Mattia**

*Department of Computer, Control and Management Engineering “Antonio Ruberti”, Sapienza University of Rome, Italy