2022 International Symposium on Computer Architecture and High Performance Computing Workshops (SBAC-PADW)

Energy-efficient online resource provisioning for cloud-edge platforms via multi-armed bandits

Jordan Rey-Jouanchicot*, Juan Ángel Lorenzo del Castillo†, Stéphane Zuckerman†, and E. Veronica Belmega†‡
* CY Tech, CY Cergy Paris Université, Cergy, France
† ETIS UMR 8051, CY Cergy Paris Université, ENSEA, CNRS, F-95000, Cergy, France
‡ Univ. Gustave Eiffel, CNRS, LIGM, F-77454, Marne-la-Vallée, France
Email: jordan.rey.jouanchicot@cy-tech.fr, juan-angel.lorenzo-del-castillo@cyu.fr,
stephane.zuckerman@ensea.fr, veronica.belmega@esiee.fr

***Abstract*—Edge computing is a new paradigm in which data are locally collected, aggregated and preprocessed before being sent to a Cloud platform. Edge devices, typically IoT objects, are characterized by limited computational capabilities, alongside high energy efficiency operations. In smart building or smart city applications, the overall amount of available IoT devices may account for an important computational capacity. Aggregating the idle CPU cycles of several devices would allow performing on-site parallel and distributed computing, while reusing the available resources in the network. Timely and important computational tasks such as video surveillance processing, HPC jobs, or deep neural network training, can be performed with local available resources. This includes local servers or a private, on-site cloud, as opposed to the use of public clouds, thus avoiding security issues and fostering energy efficiency. An ongoing research topic in such a distributed and volatile context is to design novel algorithms for resource provisioning that allow tasks to be distributed over a set of available IoT devices in an efficient manner. This work presents two online, adaptive and robust scheduling techniques, based on the UCB and EXP3 Multi-Armed Bandit algorithms (MABs), for resource management in Cloud-Edge Computing based environments while improving performance and reducing energy consumption. The novelty of our adapted algorithms lies in the fact that we explicitly account for unavailable computing IoT devices resulting in a time-varying available set of arms at each stage. Our numerical results show the high relevance of our approach in reaching optimal provisioning policies in time-varying environments.**

***Index Terms*—Cloud Edge computing, resource provisioning, online learning, multi-armed bandits**

I. Introduction

Designing efficient job scheduling and resource management policies with power and energy, as well as execution time in mind, requires new approaches as the cloud moves to the network Edge and to Internet of Things (IoT) systems. Indeed, the complexity of the scheduling problem increases with the degree of heterogeneity and the high diversity of properties within computational nodes in terms of available resources (*e.g.*, memory, computational power, *etc.*). In addition, the scheduler has to adapt on-the-fly to the incoming workloads with characteristics and requirements which may vary over time. Moreover, a given scheduling decision may sometimes be beneficial in the short term, but at the expense of worsening the global system performance in the mid or long term.

This issue becomes particularly important in emerging areas such as the design of *edge architectures*, in which plenty of edge servers are established to be close to multiple heterogeneous IoT devices at the edge of the network [1]. Data from sensors are collected and aggregated in the edge layer, which can be used as a support for delay-critical IoT applications and data pre-processing before they are typically routed to an upper layer, like a cloud.

Data processing at the edge may have multiple objectives, such as obtaining usage statistics of a smart building, improve energy consumption policies, perform predictive maintenance using Artificial Intelligence, *etc.* However, these actions may incur sustainability and security issues. For example, in the case of a smart building, the collected and pre-processed data may be sent to a public cloud (*i.e.*, off-site) for further processing and storage, which adds additional power consumption. In addition, letting sensitive information leave the building may raise security concerns, given that many public cloud providers store their clients' data in countries outside the EU, and are therefore subject to different privacy laws. Finally, the addition of an Edge layer allows preserving system functionalities in the case of communication interruptions at the cloud level.

In this Cloud-Edge Computing context, IoT devices perform a single periodic task with a relatively low frequency, and thus are idle most of the time. A fully distributed and parallel system could use a smart IoT network to pre-process data during the devices' idle computing cycles, before sending it to a cloud for more advanced processing. Thus, what is required, which becomes the main motivation of this work, is an *online adaptive scheduling system* that can perform resource allocation and load-balance work across the various nodes and heterogeneous parts.

Reinforcement learning (RL) and, more specifically, multi-armed bandits (MAB), allow designing online and adaptive scheduling policies which make a trade-off on-the-fly between data exploration and exploitation [2]. MAB algorithms rely on a payoff-based feedback (or reward mechanism) with little or no assumptions regarding the underlying dynamics of the environment (*e.g.*, they do not need any prior workload profiling). Such adaptive algorithms relying on strictly causal information also come with performance guarantees in terms of regret minimization.

978-1-6654-5157-4/22/$31.00 ©2022 IEEE
DOI 10.1109/SBAC-PADW56527.2022.00017

Two well-known MAB algorithms that have shown their relevance in online orchestration problems for IoT are UCB (*Upper Confidence Bound*) [3] and EXP3 (*Exponential-weight algorithm for Exploration and Exploitation*) [4]. Our main contribution in this paper is to adapt these algorithms to a more general setting in which not all the arms can be chosen at every step. In other words, not only the rewards but also the set of available arms is varying with time. This enables us to account for unavailable computing IoT devices, but also imposes the use of the stronger *dynamic regret* performance metric as opposed to the classical regret notion. In this setting, we propose two online adaptive scheduling policies, based on UCB and EXP3, to perform efficient resource allocation in idle IoT devices from an edge network. We propose a software framework, based on realistic premises, that learns the optimal provisioning policies for a job on a given Edge environment.

The rest of the article is organized as follows: Section II highlights the related work on resource allocation for Cloud-Edge computing using RL policies. Section III explains our system model. Section IV provides a gentle introduction to the Multi-Armed Bandit problem. In Section V we present the algorithms implementation. Section VI shows and discusses the obtained results. Finally, Section VII concludes this article and outlines the future work.

II. Related work

In recent years there has been an important rise in the number of articles that propose resource allocation techniques in the Edge-Cloud context using RL. Authors in [5] investigate computation offloading mechanisms of multiple selfish users with resource allocation in IoT edge computing networks by using a multi-agent RL framework. However, they refer to factors such as the transmitted power level or the radio access technology, whereas our work focuses on hardware factors such as CPU and/or memory availability. Ali S. et al. [6] present a fast uplink grant scheduling method based on a probabilistic *sleeping MABs* (i.e. the set of available arms varies over time) for machine type devices (MTDs) in wireless systems. They know that the availability of the MTDs follows the distribution of their traffic, which is not applicable to our problem, much more general. Mahmood et al. [7] have an objective closer to ours, which aims at minimizing the task duration in mobile edge cloud contexts by studying the optimal task segmentation. However, their algorithm performs optimal resource allocation by solving a convex optimization problem instead of using RL techniques. By and large, the differences of our work with respect to the existing literature are twofold: on the one hand, our objective is to efficiently approach task dispatching and its parallelization among a set of available IoT resources in a network-agnostic manner; that is, focusing only on the available devices and not on how they communicate. On the other hand, whereas in the literature each prospective resource to be chosen is seen as an arm in the MAB settings, we approach the problem in a novel way, by considering each arm as a workload distribution.

III. System model

As outlined in Section I, we propose a scenario in which an Edge infrastructure will comprise a set of heterogeneous IoT devices with different capabilities. For example, we can cite input devices such as sensors or cameras; devices with limited storage and processing capacity (*i.e.*, a Raspberry Pi); low-power microcontrollers, or devices with a high processing capacity, such as GPUs (Nvidia Jetson, for example); or network devices, such as switches and routers.

In this context, one or multiple tasks will be parallelized and executed in the edge infrastructure. Nodes (any device with network capabilities and enough computational power to receive and process data) will announce their presence in the network as well as their specifications (CPU, available disk and memory, battery level, etc.). Nodes available to perform a given task will place themselves in a hierarchical structure, with a node (either in a local Cloud or an Edge device) taking the role of a master in charge of scheduling and dispatching chunks of the parallel task among the available worker nodes, according to MAB-based policies.

Based on the premises of the proposed scenario, we provide a simulated environment, using the OpenAI GYM environment, for a set of IoT devices with different and non-stationary features. The environment generates scenarios comprising a random combination of three different types of IoT devices with varying computational power and power consumption rates denoted A, B, and C. The metrics used for each device are shown as (*maximal number of attainable GFLOPS*, *mean consumption in Watts*). We have assigned the values $A(1000, 80)$, $B(500, 30)$ and $C(100, 5)$. A is the most powerful but also the most power hungry, C is the least powerful computationally speaking but also the most economical power-wise, while B fits somewhere in the middle. For realistic purposes, every time a simulated device is initialized, a variation from a Gaussian distribution is applied to their nominal performance and consumption with a factor of 5% of the mean value, in order to reflect the little differences in performance that we find on similar manufactured devices. The environment will return, respectively, the following simulated performance values (*s_perf*) and simulated consumption values (*s_watts*) from the nominal values *n_perf* and *n_watts*:

$$s\_perf = n\_perf * \frac{5x}{100} + n\_perf \tag{1}$$

$$s\_watts = n\_watts * \frac{5x}{100} + n\_watts \tag{2}$$

With $x \in X(\Omega), X \sim N(0,1)$ in both cases.

The proposed scheduler allocates resources from a subset of the generated devices in order to distribute and run a task among the given resources. During the execution of the task, the environment behaves accordingly to reflect the effect of running a program on the IoT infrastructure and returning the program's achieved performance (Gflops, MIPS, consumed Watts, execution time) according to the chosen devices.

Algorithm 1 outlines the steps followed by our program to simulate the execution of a workload in our environment. At

Algorithm 1 Resource allocation algorithm

```
Input: Number of steps and runs
Input: Task workload
Output: Workload distribution with best estimated value
for $each\ run$ do
    $Generate\ new\ random\ IoT\ environment$
    for $each\ step$ do
        $action = action\_selection\_rule(reward)$
        $Select\ devices\ to\ allocate\ workload$
        for $d\ on\ selected\ devices$ do
            $assign[d] \leftarrow workload \times action[d]$
            $observation[d] \leftarrow perf\_feedback[d]$
        end for
        $Update\ IoT\ environment$
        $reward = update\_reward(observation)$
    end for
end for
```

each step, a workload is parallelized and distributed among the chosen devices according to the weights in the *action* array. Its execution will return performance parameters such as GFlops or execution time. Our environment simulates the execution of a fictitious workload by returning the following performance values where s_perf, n_perf, s_watts and n_watts are the values from equations (1) and (2):

$$wl\_perf = s\_perf \times \frac{y}{100} + s\_perf \tag{3}$$

$$wl\_watts = s\_watts \times \frac{y}{100} + s\_watts \tag{4}$$

With $y \in Y(\Omega), Y \sim N(0,1)$ in both cases. Again, a variation from a Gaussian distribution is applied to their performance and consumption with a factor of 1% of the mean value, which we deem to be representative enough of a typical workload variation which would run on this system.

The environment provides a framework as well, to test different MAB agents, where each agent implements a workload scheduler, as shown in the next section.

IV. Multi-armed bandits: basics

In the prototypical multi-armed bandit (MAB) problem[1], at each stage, the agent chooses one action, or arm, and receives a reward as a result. The agent's objective is to choose the actions that maximize its long-term cumulative reward. Since the feedback received is limited, the agent does not know the random process generating the rewards and only observes the reward of the chosen arm at each stage (and not the rewards of the unchosen ones).

Stochastic MABs: Formally, in a stochastic k-armed bandit problem we define the set of possible actions or arms as $A = \{a_1, a_2, \dots a_k\}$, $|A| = k$. The action $a_t \in A$ selected at time $t = \{1, 2, \dots, T\}$ receives a reward $u_t(a_t)$, where $u_t(a_t)$ is a random variable drawn from the statistical distribution P_a of arm a_t. The agent's objective is to play the arm with the highest mean reward as many times as possible over the horizon of play T, *i.e.*, maximize the cumulative reward $\sum_{t=1}^{T} u_t(a_t)$. In this setup, we can quantify the agent's performance at time T via the seminal notion of *regret*. Let μ_a be the mean value of the reward distribution P_a. Following Belmega *et al.*'s notation [8], we define

$$\mu^* = \max_{a \in A} \mu_a \quad \text{and} \quad a^* = \arg\max_{a \in A} \mu_a$$

respectively as the bandit's maximal mean reward and the arm that achieves it. We define the agent's ***mean regret***, $\bar{R}_T$, by aggregating the mean difference between the best arm and the online policy a_t chosen by the agent at each step $t = 1, 2, \dots, T$.

$$\bar{R}_T = \sum_{t=1}^{T} \mathbb{E}[u_t(a^*) - u_t(a_t)] = T\mu^* - \sum_{t=1}^{T} \mathbb{E}[u_t(a_t)]. \tag{5}$$

More precisely, the regret tells us whether or not the agent fails to identify the best arm, *i.e.*, the mean number of sub-optimal choices made, up to the instant T (weighted by the suboptimality gap of each arm). Maximizing the aggregated reward will therefore amount to looking for a regret that is sublinear in T, which is a property known as ***no regret*** and defined mathematically as $\bar{R}_T = o(T)$. This property guarantees that the relative number of the agent's sub-optimal choices will decay to zero with time.

In stochastic MABs, the best online policy in terms of regret minimization rate is the *upper-confidence bound* (UCB) algorithm. In contrast to a pure *exploitation* policy, UCB includes an additional *exploration* term, that leads to a logarithmic regret. If we define the empirical mean payoff of arm a as

$$\begin{aligned} \hat{\mu}_{a,t} &\doteq \frac{\text{sum of rewards when } a \text{ is chosen prior to } t}{\text{number of times } a \text{ is chosen prior to } t} \\ &= \frac{\sum_{i=1}^{t-1} u_i(a) \cdot \mathbf{1}_{a_i = a}}{\sum_{i=1}^{t-1} \mathbf{1}_{a_i = a}} = \frac{\sum_{i=1}^{t-1} u_i(a) \cdot \mathbf{1}_{a_i = a}}{N_t(a)} \end{aligned} \tag{6}$$

then UCB will select among the non-greedy actions according to their potential for actually being optimal, according to its equation:

$$a_{t+1} \doteq \arg\max_{a \in \mathcal{A}} \left[\hat{\mu}_{a,t} + c \sqrt{\frac{\ln t}{N_t(a)}} \right] \tag{7}$$

where $c > 0$ is the parameter that controls the degree of exploration vs. exploitation trade-off. As already mentioned, UCB has a logarithmic mean regret: $\bar{R}_T \leq \mathcal{O}(\log\ T)$, which is attained with $c > 2$ and is optimal in stochastic problems.

Adversarial MABs: Another important MAB algorithm is the *exponential weights for exploration and exploitation* (EXP3) algorithm [9], which is tailored to a more general environment, in which the problem is not purely stochastic in nature and, therefore, it is not possible to identify an optimal arm. This context is known as *adversarial MABs*, since the agent may face any possible sequence of rewards, including those imposed by an adversary that actively tries to minimize

[1] The multi-armed bandit name comes from the slot machines in casinos.

the agent's reward. In this non-stationary or adversarial MAB setting, the notion of regret becomes:

$$R_T = \max_{a\in\mathcal{A}} \left(\sum_{t=1}^{T} u_t(a) \right) - \sum_{t=1}^{T} u_t(a_t). \tag{8}$$

In order to reach a sublinear regret, the choice of arms has to be randomized. More precisely, EXP3 keeps a cumulative score of the performance of each arm and then employs a random arm drawn with probability x_t that is exponentially proportional to this score

$$\begin{aligned} \hat{u}_t(a) &= u_t(a) \cdot \mathbf{1}_{a=a_t}/x_{a,t}, \ \forall a \in \mathcal{A} \\ y_{t+1} &= y_t + \gamma \hat{u}_t, \\ x_{t+1} &= \Lambda(y_{t+1}), \end{aligned} \tag{9}$$

where the *logit choice map* $\Lambda : \mathbb{R}^k \to \Delta(\mathcal{A})$ is given by

$$\Lambda(y) = \frac{(\exp(y_a))_{a\in\mathcal{A}}}{\sum_{b\in\mathcal{A}} \exp(y_b)}, \tag{10}$$

where $\Delta(\mathcal{A}) = \{x \in \mathbb{R}^k | x_a \geq 0, \ \forall a \in \mathcal{A}, \ \sum_{b\in\mathcal{A}} x_b = 1\}$ is the probability simplex. The first step, called importance sampling, is required to build $\hat{u}_t$, an unbiased estimate of the unobserved reward vector based on the obtained (scalar) reward $u_t(a_t)$ from choosing arm a_t.

EXP3 attains a sublinear regret $\bar{R}_T = \mathcal{O}(\sqrt{kT\log k})$ with a trade-off parameter $\gamma = \sqrt{\log k/(kT)}$. This regret minimization rate is indeed slower than that of UCB, but it is nevertheless optimal in adversarial problems, in which UCB has no theoretical performance guarantee. Worse, it can also be brought to a halt entirely by an adversary.

V. Our proposed MAB-based algorithms

In our case, the set of available arms is not fixed and may change at each stage: we call this a time-varying set of arms, $\mathcal{A}_t$. Our online policy chooses an action $a_t \in \mathcal{A}_t \subseteq \mathcal{A}$ such that $|\mathcal{A}_t| = n \leq k$ where n is arbitrarily fixed from the beginning[2]. The motivation behind this stems from the fact that some of the IoT devices $(k - n)$ might not be available at each stage t. This hence leads to a non stochastic setting, since the statistics of the available arms vary with time.

A. Dynamic regret

In this non stationary setting, we consider the dynamic regret, which is defined as follows:

$$R_T^\star = \sum_{t=1}^{T} (u_t^\star - u_t(a_t)), \tag{11}$$

where $u_t^\star = \max_{a\in\mathcal{A}_t} u_t(a)$. One can easily see that the above target is much more ambitious than the classic regret in (8). Indeed in (8), the online policies are compared with the *best fixed policy* on average over the time horizon, whereas in (11), the online policies are compared with the *instantaneous dynamic policy*. Notice that online MAB-policies are only exploiting past observed rewards or *strictly causal information* in their updates, which hints that achieving *no dynamic regret* is only possible when the environment is not varying too drastically from one stage to the next [10].

Nevertheless, using the classic (as opposed to dynamic) regret is not possible in this setting because we do not allow our adapted algorithms to choose an arm that uses one or more unavailable devices. When using a dynamic regret, by having $R_T^\star/T$ go to zero (which is equivalent to no dynamic regret, *i.e.*, $R_T^\star = o(T)$), this implies that our algorithms are capable of tracking the instantaneous optimal solution as it changes in time. Otherwise, this would imply that the dynamic optimal solution has too strong temporal variations to be tracked by our adaptive algorithms relying on past information solely. In this case, a less ambitious target has to be identified by properly adjusting the classic regret in (8) to account for the time-varying available subset of arms, $\mathcal{A}_t$, at each stage.

B. Our MAB-based task assignment algorithms for dynamic available arms

We have modified, for both the UCB and EXP3 algorithms, the subroutine devoted to selecting actions, in order to adapt it to our problem where only a (known) subset of all actions is available. Algorithms 2 and 3 show, respectively, our UCB and EXP3 implementations.

Regarding our adapted UCB algorithm, we update a w vector containing as inputs the scores: w_a values for each arm a, corresponding to the objective in (7) based on the exploitation and exploration terms. Then, the action with the maximum associated w_a is chosen and we verify whether this action is valid, *i.e.*, whether it can be mapped only to available devices. If it is valid, then this action is selected. Otherwise, this action is discarded and we draw a new action with the maximum associated score.

Regarding our adapted EXP3 algorithm, at each step, we first choose a random action a_{t+1} from the discrete probability vector x_{t+1} as with classic EXP3. If the action is valid, we choose it and the algorithm continues. Otherwise, we re-draw (with replacement) a new random action from the same distribution until a valid action is finally identified.

VI. Numerical results

A. Simulation environment and parameters

In the existing literature on resource allocation problems via MABs, each prospective resource to be chosen is typically seen as an arm, or *action*. Here, we propose each action to be a *workload distribution*, *i.e.*, a set of devices chosen from the available IoT devices, each receiving a fraction of the workload. For example, for an infrastructure composed of six devices where only three are chosen by the resource allocator, possible valid actions, or arms, may be:

$$\begin{aligned} a_1 &= [\mathbf{0.3}, 0.0, \mathbf{0.1}, 0.0, 0.0, \mathbf{0.6}] \\ a_2 &= [0.0, 0.0, \mathbf{0.4}, \mathbf{0.4}, 0.0, \mathbf{0.2}] \\ a_3 &= [\mathbf{0.0}, 0.0, \mathbf{1.0}, 0.0, \mathbf{0.0}, 0.0] \end{aligned}$$

[2] When $n \equiv$ k, the classical setup stands where all arms are always available and can be exploited by the agent.

Algorithm 2 Adapted UCB algorithm

```
Require: tuning parameter $c \geq 2$, initial reward sample $\hat{\mu}_{a,1}$
    from each arm $a \in A$
set $n_a \leftarrow 1$ for all $a \in \mathcal{A}$
for $t = 1$ to $T$ do
    set $w_a \leftarrow \left[\hat{\mu}_{a,t} + c\sqrt{\frac{\ln t}{N_t(a)}}\right]$ for all $a \in A$
    repeat
        $a_{t+1} \leftarrow \arg\max_a(w_a)$
        if at least one device of $a_{t+1}$ is not available then
            discard arm $a_{t+1}$
        end if
    until all devices selected by $a_{t+1}$ are available
    receive utility $u_t(a_t)$
    $N_{t+1}(a) = N_t(a), \forall a \neq a_t,\ N_{t+1}(a_t) \leftarrow N_t(a_t) + 1$
    $\hat{\mu}_{a_t,t+1} \leftarrow (1 - \frac{1}{N_{t+1}(a_t)})\hat{\mu}_{a_t,t} + \frac{1}{N_{t+1}(a_t)}u_t(a_t)$
end for
```

Algorithm 3 Adapted EXP3 algorithm

```
Require: Parameter $\gamma > 0$
set $y_1 \leftarrow 0$
for $t = 1$ to $T$ do
    $x_{t+1} \leftarrow \Lambda(y_t)$
    repeat
        $a_{t+1} \leftarrow draw(x_{t+1})$
    until $a_{t+1}$ use only devices available at this step
    receive utility $u_t(a_t)$
    $\hat{u}_t(a) \leftarrow u_t(a) \cdot \mathbf{1}_{a=a_t}/x_{a,t}, \forall a \in \mathcal{A}$
    $y_{t+1} \leftarrow y_t + \gamma\hat{u}_t$
end for
```

Each element in an action array represents the percentage of work delivered to each device according to the array index. For example, for action a_1, device 0 will receive 30% of the workload, device 2 will receive 10%, and device 5 the remaining 60%. After running the workload, the agent will receive an aggregated reward from the selected devices. We propose the following reward $u_t(a_t)$ for an action a_t executed at *step* t:

$$u_t(a_t) = \frac{Gflop}{\alpha * consumption + \beta * execution\_time^2} \tag{12}$$

Hence, to compute the reward, we take into account the number of floating-point operations $Gflop$ to be executed in a task, and we compute its ratio with the sum of two criteria: $consumption$ and $execution\_time$. The term $consumption$ represents the workload's aggregated *consumed power in Watts* multiplied by the execution time: $execution\_time$, which represents the workload task's completion time, measured in seconds. Each criterion is weighted by α for $consumption$ and (respectively) β for $execution\_time^2$. We squared the value of the execution time in the second term, after experimenting with various workload simulations, to provide a balance between energy consumption and execution time. From these experiments, we decided to set $\alpha = 1$ and $\beta = 1$.

TABLE I: Test battery executed in our framework

Generated devices	*Chosen devices*	*Agent(s)*	*Workload*
4	2	UCB and EXP3	Invariable
6	2	UCB and EXP3	Invariable
8	2	UCB and EXP3	Invariable
4	4	UCB and EXP3	Invariable
6	2	UCB and EXP3	Variable
4	4	UCB and EXP3	Variable

The simulator has been evaluated by setting up a battery of tests as presented in Table I. *Generated devices* represents the number of devices available in our IoT infrastructure. *Chosen devices* is the number of IoT devices – chosen among the generated devices – that the scheduler will use to distribute and run the workload. We choose these ratios of generated/chosen devices by considering that, in a real setup, an agent will not exceed an occupation of 75% of the available devices, although we have also tested a 4 over 4 devices (or 100%) case for performance testing purposes. In all cases, we have evaluated the behaviour of both our adapted UCB and EXP3 algorithms. To better understand whether the learning curve of our algorithms depends on the type of workload, we have tested them either by generating a workload and using it through the whole experiment (*Invariable Workload*, by default) or by generating a new workload at each step of the algorithm (*Variable Workload*, indicated as *VW* in the results). In the latter case, the workloads vary between 500 and 5000 $GFlops$. Each test comprises $T_{\max} = 10000$ algorithm *steps*, which makes up for *a run*, and we execute 20 such runs

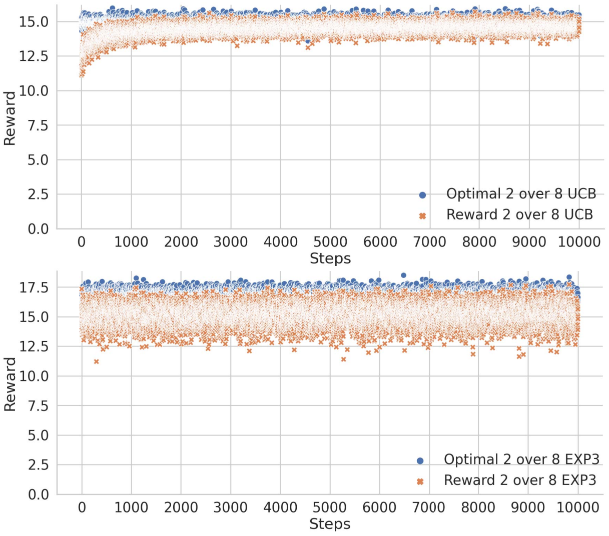

Fig. 1: Average performance for 2 devices chosen out of 8. Each figure shows the dynamic optimal rewards (blue dots) as well as the adaptive MABs obtained reward (orange dots), at each step. Data are averaged over 20 runs with different scenarios. Top: our adapted UCB. Bottom: our adapted EXP3.

to obtain the average behaviour of each algorithm. We have measured the convergence speed of the reward and the regret for each test.

B. Simulation results

Figure 1 shows our adapted UCB and EXP3 performance for the 2 *over* 8 *devices* test proposed in Table I. Dots in blue show the dynamic optimal reward at every step, whereas the dots in orange show the obtained reward for the chosen actions. We appreciate that, for this case in which the workload is invariable, UCB quickly tracks the optimal rewards. EXP3, however, improves much more slowly although it seems to converge with a higher number of steps. The dynamic regret for both UCB and EXP3 are displayed in figures 2 and 3, respectively. They confirm the same behaviour for other settings: 2 devices over 4, 6 or 8 devices and invariable workloads.

When using variable workloads we have observed a gap between the obtained rewards by our adapted MAB algorithms and the dynamic optimal rewards. The reason for this is that, if the workloads strongly vary in time, then the optimal work distribution among the available devices can also change considerably. Figures 2 and 3 show that there is indeed an offset in the average dynamic regret for variable workloads and, hence, it does not tend to zero. The reason lies in the fact that the dynamic regret is a much more ambitious target than the classic regret. A zero dynamic regret is very hard, if not impossible, to achieve with strictly causal information (only past rewards are observed) and in highly dynamic environments. In such environments, a less ambitious target has to be defined inspired from the classical regret.

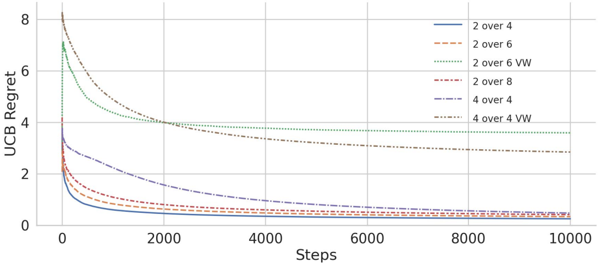

Fig. 2: Dynamic regret obtained by our adapted UCB algorithm for our test battery. We notice the convergence towards zero in all cases with invariable workloads.

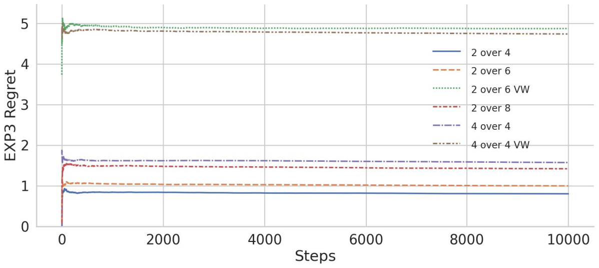

Fig. 3: Dynamic regret obtained by our adapted EXP3 algorithm for our test battery. The regret's convergence towards zero is not obvious, although it does decrease with time.

VII. Conclusions and Future Work

In this paper, we exploit a modified version of the well-known UCB and EXP3 multi-armed bandit algorithms to propose a resource allocation scheduler for Cloud-Edge Computing environments. The scheduler selects a subset of IoT devices among the available ones to dispatch a computational workload and hence, takes advantage of the available CPU cycles of such devices. Our simulation results show that, for a fixed workload and a subset of the available devices, both algorithms can track the dynamic optimal rewards, with UCB having a much faster dynamic regret decay rate. For variable workloads, though, none of the algorithms are able to track the optimal rewards based only on strictly causal information, in which a less ambitious target needs to be introduced. These results are very promising in realistic environments, given that the scheduler will always select only a subset of available IoT devices and it will run identical or very similar computationally-intensive workloads. Future work may include a more exhaustive testbed where a larger number of arms has to be accounted for and in which our MAB algorithms may have to be adjusted further. This work is a first step in considering a larger setup where the scheduler will be adapted to a real IoT testbed.

References

[1] S. Deng, Z. Xiang, P. Zhao, J. Taheri, H. Gao, J. Yin, and A. Y. Zomaya, "Dynamical resource allocation in edge for trustable internet-of-things systems: A reinforcement learning method," vol. 16, no. 9, pp. 6103–6113.

[2] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. The MIT Press, 2018. [Online]. Available: http://incompleteideas.net/book/the-book-2nd.html

[3] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," vol. 47, no. 2, pp. 235–256. [Online]. Available: https://doi.org/10.1023/A:1013689704352

[4] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire, "The nonstochastic multiarmed bandit problem," vol. 32, no. 1, pp. 48–77. [Online]. Available: https://epubs.siam.org/doi/abs/10.1137/S0097539701398375

[5] X. Liu, J. Yu, Z. Feng, and Y. Gao, "Multi-agent reinforcement learning for resource allocation in IoT networks with edge computing," vol. 17, no. 9, pp. 220–236.

[6] S. Ali, A. Ferdowsi, W. Saad, and N. Rajatheva, "Sleeping multi-armed bandits for fast uplink grant allocation in machine type communications," in *2018 IEEE Globecom Workshops (GC Wkshps)*, pp. 1–6.

[7] A. Mahmood, Y. Hong, M. K. Ehsan, and S. Mumtaz, "Optimal Resource Allocation and Task Segmentation in IoT Enabled Mobile Edge Cloud," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 12, pp. 13 294–13 303, Dec. 2021.

[8] E. V. Belmega, P. Mertikopoulos, R. Negrel, and L. Sanguinetti, "Online convex optimization and no-regret learning: Algorithms, guarantees and applications." [Online]. Available: http://arxiv.org/abs/1804.04529

[9] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire, "Gambling in a rigged casino: the adversarial multi-armed bandit problem," in *Annual Symposium on Foundations of Computer Science - Proceedings*, pp. 322–331. [Online]. Available: https://princeton-staging.pure.elsevier.com/en/publications/gambling-in-a-rigged-casino-the-adversarial-multi-armed-bandit-pr

[10] N. Cesa-Bianchi and G. Lugosi, *Prediction, learning, and games*. Cambridge university press, 2006.