Evaluation of Multi-Armed Bandit algorithms for efficient resource allocation in Edge platforms

Jiangbo Wang, Stéphane Zuckerman, Juan-Angel Lorenzo-Del-Castillo

Evaluation of Multi-Armed Bandit algorithms for efficient resource allocation in Edge platforms

Jiangbo Wang[1 [0009−0005−8451−9655]], Stéphane Zuckerman[1 [0000−0002−8586−0477]], and Juan Angel Lorenzo del Castillo[1 [0000−0002−8354−1288]]

Laboratoire ETIS, UMR 8051, CY Cergy Paris Université, ENSEA, CNRS, F-95000, Cergy, France
{jiangbo.wang,stephane.zuckerman}@ensea.fr
juan-angel.lorenzo-del-castillo@cyu.fr

Abstract. As computational systems become more heterogeneous and the number of computing nodes increases, designing high-performance and energy-efficient scheduling policies for Edge/IoT platforms has become increasingly important. This work evaluates multi-armed bandit (MAB) strategies for efficient resource allocation in Edge platforms, like smart cities or smart buildings. Factors like parallel performance and energy usage optimization were taken into account. The resource allocation methods proposed in this paper extend beyond simulations, involving real tasks run on actual IoT devices. We find that MAB scheduling, adapted to the specifics of applications running on heterogeneous IoT devices, can effectively balance execution time and power consumption, thus achieving optimal task allocation and resource utilization.

Keywords: Multi-Armed Bandit Algorithms · Resource Allocation · Scheduling · Edge Computing · IoT.

1 Introduction

Power-efficient scheduling policies have become a crucial issue in parallel and distributed computational systems. The complexity of the scheduling problem increases with the number of heterogeneous compute nodes displaying various resource properties (*e.g.*, memory, computational power, *etc.*). In addition, time-varying workload characteristics require schedulers to adapt dynamically. Moreover, a beneficial scheduling decision in the short term may become detrimental in the mid or longer term for the whole system. Emerging smart cities or buildings, leveraging IoT systems, directly benefit from solving this issue. In a typical Edge Computing architecture, multiple heterogeneous IoT nodes interact with sensors or actuators to communicate data and/or commands. Data are collected and aggregated in the Edge Layer, to later be pre-processed and further uploaded to an upper layer, *e.g.*, a cloud. The outcome of these activities may turn into knowledge and/or produce responses that will trigger commands in the actuators. The collected and pre-processed data sent to the cloud outside the building for further processing and storage adds to the overall power consumption.

Further, individual objects are becoming smarter: they embed higher-end micro-processors with very low power consumption. Some IoT devices are mostly idle, performing a single action with a high periodicity. Such highly heterogeneous IoT systems could be used to leverage additional computations and be leveraged to preprocess data during idle times, before sending it to a fog or a cloud system. Hence, a smart scheduler is required, capable of coping with the particularities of IoT devices: reduced computing power, limited battery time, *etc.*. Classic schedulers often perform a static workload distribution across heterogeneous but otherwise identical compute nodes. More advanced scheduling techniques categorise workloads according to their characteristics (priority, resources utilisation, *etc.*) to take workload and the architecture into account.

Our previous work [6] explored the resolution of energy-efficient online resource allocation problems in cloud-edge platforms, proposing an energy-efficient online resource allocation method based on MAB algorithms. We theoretically proved that certain MAB algorithms keep converging when we explicitly account for unavailable computing IoT devices, resulting in a time-varying available set of arms at each stage. In this follow-up work, we start from the premise that different devices are suitable for different types of tasks. Therefore, different types of tasks require different devices for task allocation. This work involves implementing these MAB algorithms in an actual IoT testbed, proving that we can reach optimal provisioning policies, with real workloads, in time-varying environments.

We propose an **online adaptive scheduling system** capable of performing resource allocation and load-balance work across the various heterogeneous nodes, while learning the most suitable device allocation policy according to criteria such as performance, energy consumption, *etc.*, in the process.

The rest of this article is organized as follows: Section 2 covers the State of the Art in MAB-based scheduling policies in IoT. Section 3 presents the architecture design of our testbed. Section 4 discusses our results. We conclude our article and explore future work in Section 5.

2 Related work

Task and job scheduling in an IoT context is a very well-explored topic in general. Several surveys tackle the subject, for instance Laroui et al. [3].

Multi-armed bandit (MAB) scheduling has naturally been applied in various IoT-related fields, including wireless communication networks [4]; Fog-based computing [5,7]; application-specific task allocation, *e.g.* federated learning [8,2], space-air-ground-edge maritime networks [9], *etc.* Such works often decide to favour one criterion over another depending of the chosen application, *e.g.*, power consumption over computational capability. All these works tend to target a heterogeneous network of objects. However, our work focuses on using idle times of such devices, *i.e.*, exploit edge devices which are often over-provisioned in an IoT context, with a main task to perform, and idle cores. As battery life is another concern for (self-monitored) devices, we must take into account device

availability as a dynamic factor too, even when the devices are marked compute nodes as "available."

3 Testbed Design

3.1 IoT cluster structure

We need an infrastructure capable of allocating workloads to different IoT devices according to the policy of our MAB-based scheduler. The architecture of our IoT cluster is presented in Figure 1. The cluster comprises twelve devices, divided in three categories: 2 NVIDIA Jetson TX2, 4 Raspberry Pi 4, and 5 BeagleBoard Black. Technical specifications are given in Table 1. Benchmark testing indicates that the performance of the NVIDIA Jetson TX2 surpasses that of the Raspberry Pi 4, which in turn exceeds that of the BeagleBoard. Conversely, the power consumption of the BeagleBoard is less than that of the Raspberry Pi 4, which is less than that of the NVIDIA Jetson TX2.

Table 1: Specifications of the three types of devices used in the experiments.

Device	Specifications
Nvidia Jetson TX2 (IDs: 0-1)	Denver 2.0 (2 cores); ARM Cortex-A57 (4 cores), clocked at 2GHz. 64-bit ISA. 256-core GPU (unused here).
Raspberry Pi 4 (IDs: 2-5)	Broadcom BCM2711 SoC with ARM Cortex-A72, clocked at 1.5GHz. 64-bit ISA.
BeagleBoard Black (IDs: 6-10)	ARM Cortez-A8 (1 core). 32-bit ISA.

The device running the MAB scheduling system is referred to as the *master* device, while the other IoT devices executing the test task programs are referred to as *worker* devices. In our tests, 1 master device was used, along with 11 worker devices. All devices are connected *via* an Ethernet switch.

3.2 Workloads

To evaluate the distribution of a workload among the available devices in the IoT cluster, we used the MPI version of the NAS Parallel Benchmarks suite (NPB) [1]. Due to their properties, we focused on two particular benchmarks: EP (Embarrassingly Parallel), which processes independent data sets with minimal communication overhead, and LU, which requires frequent data exchange and synchronisation during the parallel execution[1]. Our testing task currently involves assigning variable-sized tasks to different devices for execution and obtaining the time and total power consumption upon completion. The current NAS benchmarks do not consider the possibility of allocating uneven workloads to different devices, so we modified the EP and LU benchmarks to accept parameters such as the device type and allocation percentage, and then determine the amount of data to be allocated to each device.

[1] A link to our modified source code will be added upon acceptance of this work.

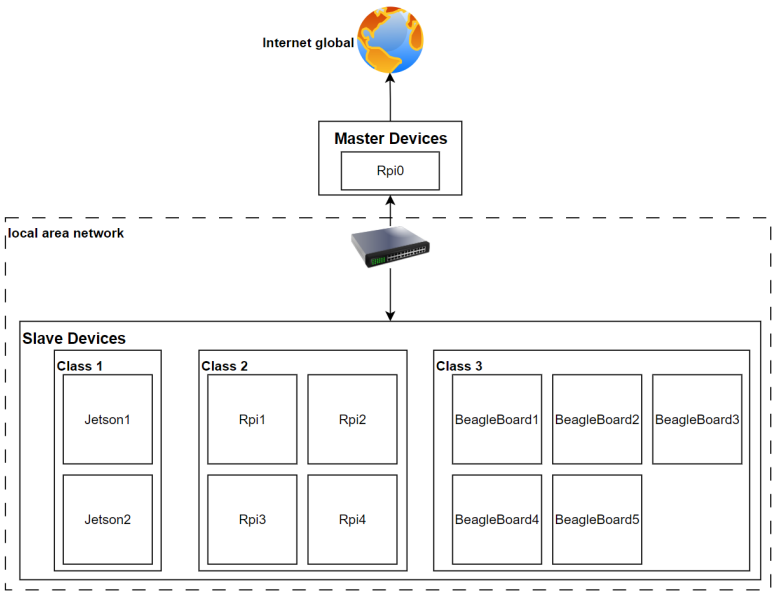


Fig. 1: IoT cluster structure: 2 Jetson TX2; 4 Raspberry Pi 4; 5 Beagleboards.

3.3 Model learning process

Figure 2 depicts the learning model process within our designed MAB scheduling system, which operates on the Master device. It comprises two parts: a MAB allocation model and a scheduling subsystem. The scheduling system features two components: the Device Manager and the Task Manager. The Device Manager is responsible for managing the status of devices. "Device Status" records the state of the devices. If a device is already running a task, and a second task still requires the same device, "Task Blocking" prevents the second task from running until the first task is completed and the device status updates to available, allowing the second task to proceed. The Task Manager is responsible for managing the execution of tasks, including: Generating commands to execute tasks based on the devices and allocation ratios selected by the MAB allocation model; and, after the completion of a task, returning the task execution time and power consumption back to the MAB allocation model.

The Task Manager comprises two threads: Thread 1 is used for executing the task assigned by the MAB Scheduler (Workload Task), while Thread 2 is used for executing the task that the device was originally performing (Main Task). When the Main Tasks are running, the MAB allocation model cannot select the devices for running Workload Tasks; that is, the Main Task is used to recreate a multi-tasking situation where certain devices are unavailable, reflecting the choices made by the MAB allocation model.

After the Workload Task is completed, the MAB allocation model receives the task execution time and power consumption from the task scheduling system, calculates the Reward based on the Reward calculation formula, and updates the

model parameters according to the Reward. Then, based on the current state of devices provided by the task scheduling system, it selects new devices and allocation percentages. This process continues in a loop until the model learning process concludes. Ultimately, we obtain a MAB allocation model tailored for this specific task. Authors will be providing a link to the source code upon acceptance of this paper.

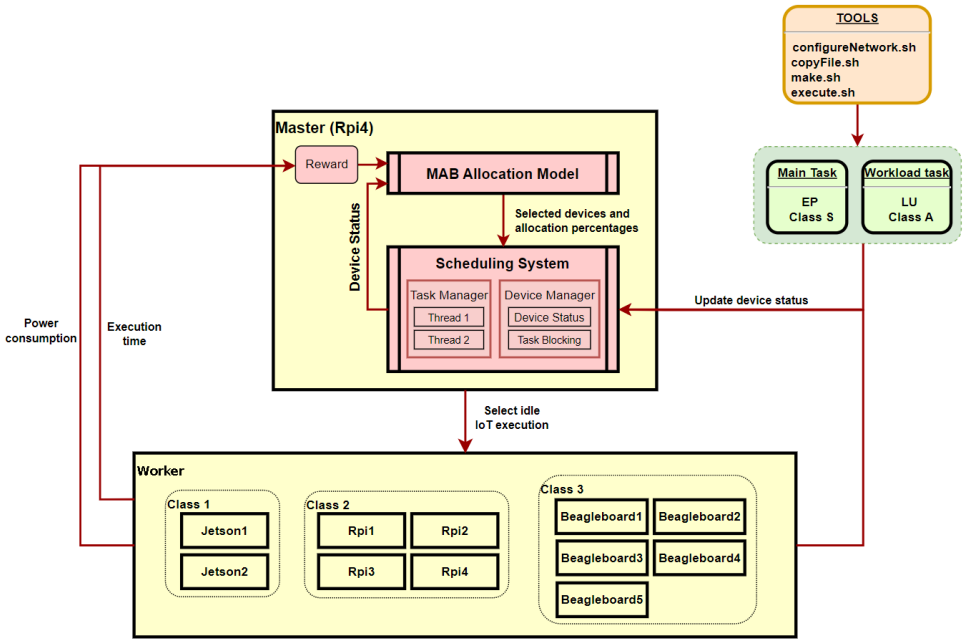


Fig. 2: Model Learning System Structure

3.4 MAB Action Space

By setting the allocation ratio precision to 0.1, a total of 11 allocation ratios can be produced. We refer to the combination of the device and workload percentage chosen by the MAB allocation system at one time as an *action*, for example, "Jetson1: 0.3, Rpi4: 0.7". When devices of the same category are considered different (*i.e.*, "Jetson1: 0.3, Rpi4: 0.7" and "Jetson2: 0.3, Rpi4: 0.7" are considered different actions), the resulting action space explodes exponentially the total number of devices and the number of allocations increase, raising up to 34,166 possible combinations for 11 available devices, with 4 of them allocated. In our previous simulations [6], where the complexity of tasks was represented numerically and tasks could be completed instantly, the size of the action space was not a concern. However, in real-world testing, if the action space is very large, it will take a very long time to get the results. Therefore, in our project, we treat devices of the same category as the same device, *i.e.*, only one device of the same type is selected at a time, which can significantly reduce the action space, as shown in Table 2.

Table 2: Action Spaces Composed of the Same Devices within the Same Category

ID	Total Devices	Allocated Devices	Precision	Action Space
1	6[2,2,2]	2	0.1	60
2	6[2,2,2]	3	0.1	366
3	6[2,2,2]	4	0.1	1221
4	11[2,4,5]	2	0.1	60
5	11[2,4,5]	3	0.1	476
6	11[2,4,5]	4	0.1	2431

In our experiments, the following format was used: #devices[#Jetsons,#Raspberry Pi 4s, #BeagleBoards]. For instance, 11[2,4,5] means all our available devices were usable. The device IDs are provided in Table 1.

3.5 Reward

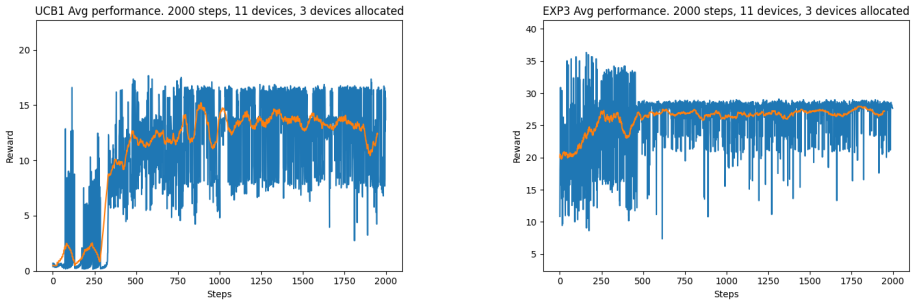
MPI programs distribute the required data across the cores of different devices for computation, and end with a global barrier, meaning the slowest process sets the overall latency to produce results. Hence, in our experiments, the system must wait until all cores have completed their computations before it can collect their results. On the other hand, the power consumption of the device can also be determined using measurements from the device. All things considered, we propose the reward formula shown in Equation 1, which is the reward for action a_t executed at step t, where β is the execution time reward weight factor, and γ is the power consumption reward weight factor.

$$u_t(a_t) = \beta * \frac{1}{execution\_time} + \gamma * \sum_i^n \frac{15}{P\_consumption} \tag{1}$$

4 Results and discussion

As shown in Figure 3, the rewards obtained with actual workloads are similar to those predicted by the simulations, and both converge. This section analyzes the actions selected by the MAB allocation model when different action spaces are chosen, and explores the reasons behind these selections.

Impact of the Number of Devices We chose the action space with ID 5 from Table 2 to run a workload, meaning a maximum of three devices are selected for task allocation, running for 2000 steps and setting both alpha and beta to 1. The results are shown in Table 3. Compared with Rank1, Rank2, and Rank4, it is observed that the longest execution time and the smallest time reward occur when only one device is chosen. However, this setting results in the largest power consumption reward. When the number of devices performing tasks increases, the execution time can be reduced. For example, the time taken for three devices to complete tasks in Rank4 is less than the time for two devices



(a) UCB1 (b) EXP3

Fig. 3: Reward convergence graphs for the EP benchmark using the UCB1 and EXP3 algorithm models.

in Rank1, resulting in a higher time reward. However, selecting more devices leads to increased power consumption, thereby reducing the power consumption reward. Since α and β are set to 1, indicating a need to balance the impact of time and power consumption, the optimal action for the trained MAB allocation model is to distribute the tasks between two Raspberry Pis, with the allocation ratio of 0.5 for each.

Table 3: Action Ranking and Associated Metrics. Action ID=5 for EP and UCB1

Rank	**Action**	**Freq***	**E_time***	**T_R***	**P_R***
1	3: 0.5, 4: 0.5	886	0.189	5.736	8.194
2	4: 1.0	236	0.314	3.300	9.341
3	2: 0.5, 4: 0.5	69	0.213	5.203	7.505
4	2: 0.3, 3: 0.4, 4: 0.3	47	0.165	6.794	6.532
5	3: 1.0	44	0.300	3.363	9.703
6	0: 0.3, 2: 0.4, 3: 0.3	20	0.152	7.184	5.890
24	0: 0.4, 1: 0.4, 2: 0.2	5	0.161	4.446	6.250
25	0: 0.5, 1: 0.5	5	0.204	4.606	4.912

Freq: Frequency, E_time:Average Execution Time
T_R: Execution Time Reward, P_R: Average Power Consumption Reward

Impact of Action Space Size When we run the model with the action space with ID 6, the set of devices available for task allocation is increased from three to four and the action space increases from 476 to 2431. The results for 5000 steps and $\alpha = \beta = 1$ indicate that when the number of devices increases further, the execution time decreases compared to when three devices are used. However, the power consumption also increases. Thus, the best action remains to distribute

tasks between two Raspberry Pis, with an allocation ratio of 0.5 each. Since $\alpha = \beta = 1$ to balance the impact of execution time and power consumption, we observe that the frequency of choosing two, three, or four Raspberry Pis for execution is similar. When increasing from three to four devices, we observe that the learning time increases from 1.21 hours to 4.65 hours. Therefore, deciding how many devices to allocate for tasks must consider the number of available devices and the available running time, depending on whether the goal is to reduce execution time or power consumption.

Impact of the Type of Device As observed from Rank 1 and 25 in Table 3, when Jetson devices are chosen (*i.e.*, IDs 0-1) to execute work, the task execution time is longer than when using Raspberry Pis. Consequently, the time reward is lower. Additionally, since the operating power of Jetson exceeds that of Raspberry Pis, the power consumption associated with tasks executed on Jetson is significantly higher than that on Raspberry Pis, resulting in a substantially lower average power consumption reward. Therefore, the MAB allocation model is less likely to choose Jetson for task execution. There are several possibilities why task execution takes longer on Jetsons compared to Raspberry Pis:

Impact of Inter-Core Data Exchange: The NVIDIA Jetson is a heterogeneous multi-core CPU, containing a dual-core NVIDIA Denver 64-bit CPU and a quad-core Arm-Cortex-A57 processor. These different processor architectures can work together to efficiently complete a variety of tasks. However, when utilizing both to perform tasks, data exchange between different cores may lead to certain time wastage. This is because different types of cores have distinct data processing and memory access patterns. When data is transferred from one core to another, additional time may be required to process and adapt to the different architectural characteristics. To test if the impact of inter-core data exchange is to blame, we separately utilized the Denver and Cortex-A57 CPUs, as well as both processors simultaneously to run the same task (EP, LU, S classes represent lightweight tasks, A class represents complex tasks), analyzing whether the execution time changes and comparing it with the Raspberry Pi. We verified that using both Denver and ARM cores to execute the LU.S task takes 0.012s longer than using the ARM core alone. For the EP.S, EP.A, LU.A tasks, using both Denver and ARM cores to execute tasks is significantly faster than using the ARM core alone. Therefore, data transmission between different cores indeed wastes time, which is in the millisecond range, impacting ultra-lightweight tasks. For complex tasks, using all six cores is a better choice.

Impact of Task Type: As mentioned before, EP is a kernel that essentially does not require inter-processor communication. LU, on the other hand, performs a comprehensive computational fluid dynamics (CFD) calculation, which requires frequent data exchange and synchronization, significantly increasing the communication demands during parallel execution. We empirically verified that the Raspberry Pi completes the EP task in less time than the Jetson. However, for the LU task, the Raspberry Pi's execution time is significantly longer than that of the Jetson. When tested with the LU benchmark, the data obtained by the UCB1 and EXP3 algorithms is shown in Table 4. It can be seen that

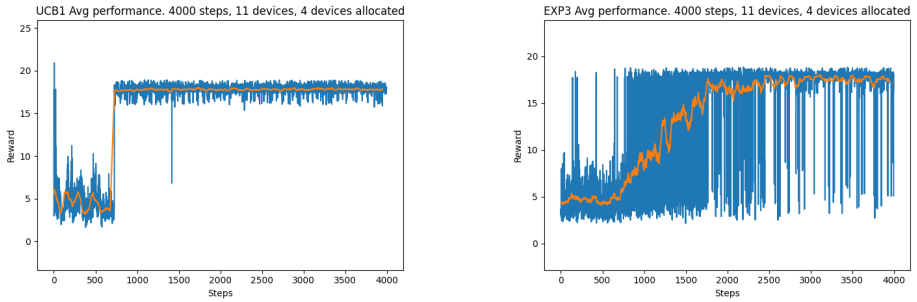
for the LU benchmark, both models eventually converge to selecting the second Jetson(1:1.0) for execution.

Table 4: Best Action and Associated Metrics. Action ID=6 for LU

Rank	**Action**	**Freq***	**E_time***	**T_R***	**P_R***
UCB1	1: 1.0	2962	3.240	6.191	11.609
EXP3	1: 1.0	2553	3.222	6.211	11.646

Freq: Frequency, E_time:Average Execution Time
T_R: Execution Time Reward, P_R: Average Power Consumption Reward

Figure 4 confirms the convergence for the LU benchmark.



(a) UCB1 (b) EXP3

Fig. 4: Reward convergence. LU benchmark using the UCB1 and EXP3 algorithm models.

Also, analyzing the CPU's instruction execution efficiency with the `perf` command in Linux, it is observed that when executing the EP.A task, the instruction per cycle rate of Raspberry Pi is higher than that of Jetson (0.84 VS 0.65). However, when executing the LU.A task, the Jetson's instruction rate surpasses that of Raspberry Pi (0.81 VS 0.61). This suggests that different tasks require different devices to run.

5 Conclusions and Future Work

This paper presented an Edge testbed that allows evaluating different scheduling strategies to run workloads in the free CPU cycles of the heterogeneous IoT devices. We tested two MABs scheduling techniques with two different parallel workloads, proposing a reward function that takes into account both performance and power consumption. In our tests, we confirmed that both algorithms

converge, even in incomplete information scenarios, and that they balance the impact of execution time and power consumption for different tasks, selecting the optimal combination of devices and allocation percentages.

Future work includes testing new workloads with different degrees of parallelism, evaluating the allocation of tasks to devices such as the BeagleBoard, which takes a long learning time, and fine tuning the values of α and β in the reward, according to different performance requirements on the testbed. In addition, we will run the same benchmarks with various types of connectivity (Ethernet, Wi-Fi, *etc.*), to get closer to how edge devices are being used.

References

1. Bailey, D., Barszcz, E., Barton, J., Browning, D., Carter, R., Dagum, L., Fatoohi, R., Frederickson, P., Lasinski, T., Schreiber, R., Simon, H., Venkatakrishnan, V., Weeratunga, S.: The nas parallel benchmarks. Int. J. High Perform. Comput. Appl. **5**(3), 63–73 (sep 1991). https://doi.org/10.1177/109434209100500306
2. Chen, S., Wang, X., Zhou, P., Wu, W., Lin, W., Wang, Z.: Heterogeneous semi-asynchronous federated learning in internet of things: A multi-armed bandit approach. IEEE Transactions on Emerging Topics in Computational Intelligence **6**(5), 1113–1124 (2022). https://doi.org/10.1109/TETCI.2022.3146871
3. Laroui, M., Nour, B., Moungla, H., Cherif, M.A., Afifi, H., Guizani, M.: Edge and fog computing for iot: A survey on current research activities future directions. Computer Communications **180**, 210–231 (2021). https://doi.org/https://doi.org/10.1016/j.comcom.2021.09.003
4. Li, F., Yu, D., Yang, H., Yu, J., Karl, H., Cheng, X.: Multi-armed-bandit-based spectrum scheduling algorithms in wireless networks: A survey. IEEE Wireless Communications **27**(1), 24–30 (2020). https://doi.org/10.1109/MWC.001.1900280
5. Misra, S., Rachuri, S.P., Deb, P.K., Mukherjee, A.: Multiarmed-bandit-based decentralized computation offloading in fog-enabled iot. IEEE Internet of Things Journal **8**(12), 10010–10017 (2021). https://doi.org/10.1109/JIOT.2020.3048365
6. Rey-Jouanchicot, J., Lorenzo Del Castillo, J., Zuckerman, S., Belmega, E.V.: Energy-efficient online resource provisioning for cloud-edge platforms via multi-armed bandits. In: 2022 International Symposium on Computer Architecture and High Performance Computing Workshops (SBAC-PADW). pp. 45–50. https://doi.org/10.1109/SBAC-PADW56527.2022.00017
7. Tran-Dang, H., Kwon, K.H., Kim, D.S.: Bandit learning-based distributed computation in fog computing networks: A survey. IEEE Access **11**, 104763–104774 (2023). https://doi.org/10.1109/ACCESS.2023.3314889
8. Xia, W., Quek, T.Q.S., Guo, K., Wen, W., Yang, H.H., Zhu, H.: Multi-armed bandit-based client scheduling for federated learning. IEEE Transactions on Wireless Communications **19**(11), 7108–7123 (2020). https://doi.org/10.1109/TWC.2020.3008091
9. Yang, T., Gao, S., Li, J., Qin, M., Sun, X., Zhang, R., Wang, M., Li, X.: Multi-armed bandits learning for task offloading in maritime edge intelligence networks. IEEE Transactions on Vehicular Technology **71**(4), 4212–4224 (2022). https://doi.org/10.1109/TVT.2022.3141740