Research Article

Makespan Optimisation in Cloudlet Scheduling with Improved DQN Algorithm in Cloud Computing

Amine Chraibi , **Said Ben Alla** , and **Abdellah Ezzati**

Hassan First University of Settat, Faculty of Science and Technology, Mathematics and Computer Science Department, LAVETE Laboratory, 26000, Settat, Morocco

Correspondence should be addressed to Amine Chraibi; a.chraibi@uhp.ac.ma

Received 6 July 2021; Revised 28 September 2021; Accepted 1 October 2021; Published 21 October 2021

Academic Editor: Pengwei Wang

Copyright © 2021 Amine Chraibi et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Despite increased cloud service providers following advanced cloud infrastructure management, substantial execution time is lost due to minimal server usage. Given the importance of reducing total execution time (makespan) for cloud service providers (as a vital metric) during sustaining Quality-of-Service (QoS), this study established an enhanced scheduling algorithm for minimal cloudlet scheduling (CS) makespan with the deep Q-network (DQN) algorithm under MCS-DQN. A novel reward function was recommended to enhance the DQN model convergence. Additionally, an open-source simulator (CloudSim) was employed to assess the suggested work performance. Resultantly, the recommended MCS-DQN scheduler revealed optimal outcomes to minimise the makespan metric and other counterparts (task waiting period, resource usage of virtual machines, and the extent of incongruence against the algorithms).

1. Introduction

Cloud computing denotes an established shared-computing technology that dynamically conveys measurable on-demand services over the global network [1]. Essentially, cloud computing offered users limitless and diverse virtual resources that could be obtained on-demand and with different billing standards (subscription and static-oriented) [2]. The CS (task scheduling or TS) also outlined independent task mapping processes on a set of obtainable resources within a cloud context (for workflow applications) for execution within users' specified QoS restrictions (makespan and cost). Workflows (common applications associated with empirical studies involving astronomy, earthquake, and biology) were migrated or shifted to the cloud for execution. Although optimal resource identification for every workflow task (to fulfil user-defined QoS) was widely studied over the years, substantial intricacies required further research:

- (1) The TS on a cloud computing platform is an acknowledged NP-hard problem
- (2) Multiple TS optimisation objectives are evident: completion time reduction and high resource usage for the entire task queue
- (3) Cloud resource dynamics, measurability, and heterogeneity resulted in high complexity

Recent research has been performed to enhance TS in cloud environment through artificial intelligence algorithms (particularly metaheuristics involving particle swarm optimisation or PSO [3], ant colony, and genetic algorithm or GA) with TS capacities. However, this article does not rely on these algorithms but instead proposes a viable alternative to them and compares it to one of the metaheuristic algorithms, such as PSO, a widely used technique in the task scheduling area. The proposed method primary objective was recommending a novel DQN scheduler for optimal

outcomes by comparing TS measures (waiting time, makespan reduction, and enhanced resource usage).

The remaining sections are arranged as follows: Section 2 outlines pertinent literary works, Section 3 presents the DQN algorithm, Section 4 highlights the recommended work, Section 5 explains the research experiment setup and simulation outcomes, and Section 6 offers the study conclusion.

2. Related Work

In cloud computing, TS, jobs scheduling, or resource selection is one of the most substantial complexities that has garnered cloud service providers' and customers' attention. Additionally, specific study types on TS intricacies reflected positive outcomes. The literature's research accomplishments in cloud resources scheduling can be divided into the following categories based on the used techniques.

2.1. Heuristic-Based Research. The heuristic algorithms, including the metaheuristic ones [4] following intuition or experimental development, offered a potential alternative for the affordable resolution of every optimisation occurrence. Following the unpredictable degree of variance between optimal and viable alternatives, past studies selected metaheuristic algorithms, such as PSO [5], GA [6], and ACO [7] to solve optimal TS policy in cloud computing using metaheuristics algorithms. Based on Huang et al.'s [8] recommendation of a PSO-based scheduler with a logarithm-reducing approach for makespan optimisation, higher performance was achieved against other heuristics algorithms. Meanwhile, Liang et al. [9] suggested a TS approach following PSO in cloud computing by omitting some inferior particles (to accelerate the convergence rate and dynamically adjust the PSO parameters). The experimental findings revealed that the PSO algorithm obtained improved outcomes compared to other counterparts. The proposed shift in genetic algorithm crossovers and mutation operators implied flexible genetic algorithm operators (FGAO) [10]. For example, the FGAO algorithm minimised execution time and iterations compared to GA. Furthermore, Musa et al. [11] recommended an improved GA-PSO hybrid with small position value (SPV) applications (for the initial population) to diverge from arbitrariness and enhance convergence speed. Consequently, the improved GA-PSO hybrid reflected more valuable outcomes than the conventional GA-PSO algorithm in resource usage and makespan. Yi et al. [12] recommended a task scheduler model following an enhanced ant colony algorithm under the cyber-physical system (CPS). The numerical simulation implied that the model resolved local searchability and TS quality concerns. Peng et al. [13] proposed a scheduling algorithm based on cloud computing's two-phase best heuristic scheduling to reduce the makespan and energy storage metrics. The authors of the paper [14] suggested a VM clustering technique for allocating VMs based on the duration of the requested task and the bandwidth level in order to improve efficiency, availability, and other factors such as VM utilisation, bucket size, and task execution time.

Sun and Qi [15] proposed a hybrid tasks scheduler based on Local search and differential evolution (DE) to enhance the makespan and the cost metrics. The authors in the paper [16] presented a parallel optimized relay selection protocol to minimise latency, collision, and energy for wake-up radio-enabled WSNs.

2.2. Reinforcement Learning-Based Research. Reinforcement learning (RL) is a machine-learning category that primarily communicated with the specified context using consecutive trials for an optimal TS method. Recently, RL has garnered much attention in cloud computing. For example, a higher TS success rate and minimal delay and energy consumption were attained in [17] by recommending a Q-learning-oriented and flexible TS from a global viewpoint (QFTS-GV). In [18], Ding et al. recommended a task scheduler using Q-learning for energy-efficient cloud computing (QEEC). Resultantly, QEEC was the most energy-efficient task scheduler compared to other counterparts (primarily catalysed by the M/M/S queueing model and Q-learning method). In [19], a TS algorithm was proposed with Q-learning for wireless sensor network (WSN) to establish Q-learning scheduling on time division multiple access (QS-TDMA). The algorithm implied QS-TDMA to be an approaching optimal TS algorithm that potentially enhanced real-time WSN performance. In [20], Che et al. recommended a novel TS model with the deep RL (DRL) algorithm that incorporated TS into resource-utilisation (RU) optimisation. The recommended scheduling model that was evaluated against conventional TS algorithms (on real datasets) in experiments demonstrated a higher model performance of the defined metrics. Another task scheduler under the DRL architecture (task scheduling algorithm based on a deep reinforcement learning architecture or RLTS) was suggested by Dong et al. [21] for minimal task execution time with a preceding dynamic link to cloud servers. The RLTS algorithm (compared against four heuristic counterparts) reflected that RLTS could efficiently resolve TS in a cloud manufacturing setting. In [22], a cloud-edge collaboration scheduler was constructed following the asynchronous advantage actor-critic (CECS-A3C). The simulation outcomes demonstrated that the CECS-A3C algorithm decreased the task processing period compared to the current DQN and RL-G algorithms. The authors of the article [23] suggest a learning-based approach based on the deep deterministic policy gradient algorithm to improve the performance of mobile devices' fog resource provisioning. Wang et al. [24] introduced an adaptive data placement architecture that can modify the data placement strategy based on LSTM and Q-learning to maximise data availability while minimising overall cost. Authors in [25] presented a hybrid deep neural network scheduler to solve task scheduling issues in order to minimise the makespan metric. Wu et al. [26] utilised DRL to address scheduling in edge computing for enhancing the quality of the services offered in IoT apps to consumers. The authors in the paper [27] applied a DQN model with a multiagent reinforcement learning setting to control the task scheduling over cloud computing.

3. Background

3.1. The RL. The RL theory was inspired by the psychological and neuroscientific viewpoints of human behaviour [28] to contextually select a pertinent action (from a set of actions) for optimal cumulative rewards. Although the trial-and-error approach was initially utilised for goal attainment (RL was not offered a direct path), the experience was eventually employed towards an optimal path. An agent only determined the most appropriate action in the problem following the current condition, such as the Markov decision process [29]. Figure 1 presents a pictorial RL representation where the RL model encompassed the following elements [30]:

- (1) A set of environment and agent states (S)
- (2) A set of actions (A) of the agent
- (3) Policies of transitioning from states to actions
- (4) Rules that identified the immediate reward scalar of a transition
- (5) Rules that outlined agent perception

3.2. The Q-Learning. One of the solutions for the reinforcement problem in polynomial time is Q-learning. As Q-learning could manage problems involving stochastic transitions and rewards without action adaptations or probabilities at a specific point, the technique was also known as the “model-free” approach. Although RL proved successful in different domains (game playing), it was previously restricted to low dimensional state space or domains for manual feature assignment. Equation (1) presents Q-value computations where S denoted an actual and immediate agent situation, α implied learning rate, γ reflected a discount factor, and $Q(S_t, a_t)$ denoted Q value to attain the “S” state by acting (a). Specifically, reinforcement began with trial and error followed by posttraining experience (the decisions corresponded to policy values that resulted in high reward counterparts).

$$Q(S_t, a_t) = Q(S_t, a_t) + \alpha \times \left[r_{t+1} + \gamma \times \max_{a'} (Q(S_{t+1}, a')) - Q(S_t, a_t) \right]. \quad (1)$$

3.3. The DQN Architecture. Training encompassed specific parameters $[s_t, a, r, s_{t+1}, \text{done}]$ that were stored as agent experiences: s_t implied the current state, a implied action, r reflected the reward, s_{t+1} denoted the following state, and done implied a Boolean value to identify goal attainment. The initial idea served to ascertain state and action as the neural network input. Meanwhile, the output should denote the value representing how the aforementioned action would reflect within the given state (see Figure 2).

3.4. Experience Replay. Experience replay [31] highlights the capacity to learn from mistakes and adjust rather than repeating the same errors. Essentially, training encompassed several parameters $[s_t, a, r, s_{t+1}, \text{done}]$ that were stored as

FIGURE 1: Representation of reinforcement learning.

FIGURE 2: Representation of deep Q-network.

agent experiences: s_t implied the current state, a denoted action, r implied reward, s_{t+1} reflected the next state, and done denoted the Boolean value to identify goal attainment. As all experiences were stored in fixed-size memory, none were linked to values (raw data input for neural network). Once the memory reached a saturation point during the training process, arbitrary batches of a specific size were chosen from the fixed memory. Regarding the insertion of novel experiences, old experiences were eliminated once the memory became full. In this vein, experience relay deterred overfitting problems. Notably, the same data could be utilised multiple times for network training to resolve insufficient training data.

4. Proposed DQN Algorithm

4.1. The TS Problem. The TS protocol in cloud computing implies one of the vital problem-solving mechanisms on the significant overlap between cloud provider and user needs, including QoS and high profit [32]. Cloud service providers strived to attain optimal virtual machine (VM) group usage through reduced makespan and waiting time. Following Figure 3, a large set of autonomous work with varying parameters was submitted by multiple users (to be managed by cloud providers in a cloud computing setting). For example, the cloud broker performed task delegations to the current VMs [33]. Different optimisation algorithms were also employed to attain optimal VM utilisation. Notably, equation (2) was incorporated to compute the overall execution time (makespan) as follows:

$$\text{makespan} = \max \left(\text{Ex}_{\text{vm}_1}, \text{Ex}_{\text{vm}_2}, \dots, \text{Ex}_{\text{vm}_j}, \dots, \text{Ex}_{\text{vm}_m} \right). \quad (2)$$

Specifically, $\text{Ex}_{\text{vm}_j} = \sum_{i=0}^n \text{Ex}_j(\text{Cloudlet}_i)$, $\text{Ex}_j(\text{Cloudlet}_i)$ demonstrated the cloudlet i execution time on vm_j [34], n implied the total number of cloudlets, and Ex_{vm_j} reflected the

FIGURE 3: Cloud scheduling architecture.

complete execution time of a set of cloudlets on vm_j execution. Figure 4 presents an example of the first-come first-served (FCFS) scheduling process where the number of virtual machines was 2 and the number of tasks was 7. Every task encompassed varied time unit lengths. Notably, makespan denoted the most considerable execution time between the aforementioned VMs. The makespan (computed in VM2) was 45.

4.2. Environment Definition. This study regarded a system with multiple virtual machines and cloudlets. Every VM encompassed specific attributes (processing power in MIPS, memory in GB, and bandwidth in GB/s). As users submitted distinct cloudlets that arrived in a queue, the broker implemented the defined scheduling algorithm to assign every cloudlet to an adequate VM. As the broker scheduling algorithm needed to make an assignment decision in every cloudlet input from the queue, the system state was changed in line with the decision. Figure 5 presents CS with a length of 3 to VM_2 .

4.2.1. State Space. Only the time taken for each virtual machine during a set of task execution was regarded in this study to support the defined system state identification process. The time counted on every virtual machine implied the total cloudlet time running on VM. The virtual machine running time facilitated makespan computation to enhance each novel cloudlet delegation where the system state changed. In this vein, the t system state with n VMs was provided by $\{\beta_t(VM_1), \dots, \beta_t(VM_i), \dots, \beta_t(VM_n)\}$. Specifically, $\beta_t(VM_i) = \sum_{j=1}^{k_i} Ex_i(Cloudlet_j)$, $Ex_i(Cloudlet_j)$ denoted the cloudlet j run time in VM_i while k_i implied the total cloudlets in VM_i . Figure 5 presents the t state as $\{9, 7, 11\}$ and $t + 1$ state as $\{9, 10, 11\}$.

4.2.2. Action Space. Available agent actions were defined in the action space. The broker scheduling algorithm was required to choose a VM from all current VMs to schedule the existing task from the queue. For example, the agent would make an action in the space with the same dimension as the number of VMs so that the action space denoted all VMs in the system. The action space was outlined with n VMs by $\{1, \dots, i, \dots, n\}$, wherein i denoted the VM index conceded by the scheduler for cloudlet assignment. In Figure 5, action space denotes $\{1, 2, 3\}$, while the chosen action implied is 2.

4.3. Model Training. The MCS-DQN model was retrained for each episode in line with the workflow in Figure 6 as follows:

Step 1: the environment and agent contexts were established, including server, virtual machine, and cloudlet attributes.

Step 2: the environment state and cloudlet queues were reset.

Step 3: the next cloudlet was selected from cloudlet queues.

Step 4: the agent selected the following action in line with the existing environment state under ϵ factor. Essentially, the ϵ factor (exploration rate) influenced the choice between exploration and exploitation in every iteration. The possibility of an agent arbitrarily choosing a VM (exploration) was $1 - \epsilon$ while the possibility of the agent choosing a VM under the model (exploitation) was ϵ . The ϵ factor (initialized by one) would reduce in every iteration following a decay factor.

Step 5: the environment state was updated by adding the cloudlet execution time to the chosen VM.

Step 6: the environment produced a reward under the recommended reward function in the following subsection.

Step 7: the agent saved the played experience into the experience replay queue.

Step 8: upon experience storage, the algorithm identified more cloudlets to schedule (to be repeated from Step 3 if more cloudlets were determined).

Step 9: the model was retrained in every episode (completing all cloudlet queues) with a batch of defined cloudlets from the experience queue. The experience replay queue was applied as a FIFO queue. The oldest experience was omitted when the queue reached a limit.

Step 10: the algorithm was repeated from Step 2 if the number of iterations was yet to reach the predefined episode limits.

Step 11: the trained MCS-DQN model was saved and exited.

4.4. Reward Function. The recommended reward function was utilised with the MCS-DQN model in Algorithm 1. The makespan of every potential scheduling was first computed.

FIGURE 4: Makespan calculation example.

FIGURE 5: Example of system state change.

FIGURE 6: The workflow of MCS-DQN training.

```

Input: vms_state//The current state of VMs
Input: vms_mips//VMs speeds in MIPS
Input: cloudlet//The cloudlet to be scheduled in MIPS
Input: action//The selected virtual machine
Output: reward//The reward
Function rewardCurrentState(vms_state, vms_mips, cloudlet, action):
    vms_length ← length(vms_state)//Get the number of VMs
    makespans ← (0, ..., 0)//List of makespans initialized with zeros with a length of vms_length
    for  $i = 0; i < \text{vms\_length}; i = i + 1$  Iterate over VMs
    do
        vms_state_copy ← copy(vms_state)//Make a copy of the current VMs state
        task_run_time ← runtime(cloudlet, vms_mips[i])//Calculate the run time of the given task in  $VM_i$ 
        vms_state_copy[i] ← vms_state_copy[i] + task_run_time//Add task's execution time to  $VM_i$ 
        makespans[i] ← max(vms_state_copy)//Calculate the makespan of the copied state
    end
    makespans_sorted ← descSort(makespans)//Sort VMs executions times in descending order
    makespans_unique_length ← unique_length(makespans)//Get the unique length of makespans to exclude makespans with the
    same value
    ranks_dict ← emptyDict//Create an empty dictionary to store each VM rank.
    start_rank ← makespans_unique_length/2//The highest rank is given to a VM
    for  $i = 0; i < \text{vms\_length}; i = i + 1$  Iterate over VMs
    do
        makespan ← makespans_sorted[i]//makespan's value from sorted makespans
        makespan not in ranks_dict//Check the makespan's value if it is not in the rank dictionary
        then
            ranks_dict[makespan] ← start_rank//Assign the rank's start value to makespan,
            start_rank ← start_rank - 1//Decrement rank's start value
        end
    end
    vm_scores ← (0, ..., 0)//List of scores initialized with zeros with a length of vms_length
    for index = 0; index < vms_length; index = index + 1//Iterate over makespans
    do
        makespan ← makespans[i]//Get makespan,
        vm_scores[index] ← ranks_dict[makespan]//Score the VM based on makespan's rank
    end
    return vm_scores[action]//Select the score based on the given action
End Function

```

ALGORITHM 1: Pseudocode of the proposed reward function.

Every VM was subsequently ranked following the makespan computation during CS. A simple example was provided to present the recommended MCS-DQN reward function (see Figure 7). The example encompassed the reward computation for a specific VM state (elaborated following the total execution time in every VM_i). Based on five VMs, every VM_i involved a set of cloudlet execution times {9, 7, 11, 8, 8}. Specifically, a newly-arrived cloudlet was scheduled with a length of five to VM2 in the example (see Figure 7(a)) by iterating over VMs, creating a copy of VM state in every iteration, adding the cloudlet to the chosen VM in iteration, and computing the makespan following the added cloudlet. Figure 7(b) presents the first iteration where the arrived cloudlet was added to VM1. Figure 7(c) presents the computed makespans. For example, the makespan would be 14 when the cloudlet was added to VM1 in the first iteration, 13 when added to VM2, and so on. In every VM_i ranking, the computed makespans were ranked following the lowest value by sorting the aforementioned makespans (see Figure 7(d)) and providing the highest score to the lowest

makespan (to decrease the highest score to be delivered to the following makespan) (see Figure 7(e)). Lastly, the corresponding reward was identified following the makespan index and corresponding VM to be scheduled. In the study context, VM2 reflected the reward as 2.

5. Results and Discussion

5.1. Experimental Setup. The recommended trained model under deep Q-learning was assessed against FCFS and PSO algorithms with the CloudSim simulator.

5.1.1. CloudSim Parameters. CloudSim is a modular simulation toolkit for modelling and simulating cloud computing systems and application provisioning environments [33]. It enables the modelling of cloud system components such as data centres, virtual machines (VMs), and resource provisioning rules on both a system and behavioural level [33].

FIGURE 7: Running example of the proposed MCS-DQN reward function. (a) The example initialisation. (b) Adding cloudlet to VM₁. (c) The calculated makespans. (d) The sorted makespans. (e) VM ranks.

The CloudSim simulator configuration in the implementation began with establishing one data center, two hosts, and five VMs with subsequent parameters (see Table 1). This configuration setup is taken from the example 6 of CloudSim code source available on GitHub (CloudSim codebase: <https://github.com/Cloudslab/cloudsim>), which is based on real servers and VMs information. At the VM level, a time-shared policy (one of the two different scheduling algorithms utilised in CloudSim) was selected. The time-shared policy facilitated VMs and cloudlets towards immediate multitasks and progress within the host. Moreover, the tasks data used in the experiments are real-world workloads of real computer systems recorded by the High-Performance Computing

Center North (HPC2N) in Sweden (the HPC2N data: https://www.cse.huji.ac.il/labs/parallel/workload/l_hpc2n/). The data contain information about tasks such as the number of processors, the average CPU time, the used memory, and other task specifications. The utilised tasks from the workload completely differ from the independent counterparts employed in the trained model.

5.1.2. The MCS-DQN Model Parameters. The MCS-DQN model application employs a neural network with five fully connected layers (see Figure 8): an input layer (for state), three hidden layers ($64 \times 128 \times 128$), and an output layer (for actions). The network was taken from an original Keras

TABLE 1: Simulation parameters.

Entity	Parameter	Value
Data center	Size	1
	Size	2
Host	Amount of storage	1 terabyte
	RAM	2 gigabytes
	Scheduling's algorithm	Time-shared policy
	Amount of bandwidth	10 gigabytes per second
	Processing power	27079, 177730 MIPS
VM	Cores	2, 6 cores
	Size	5
	Size of the image	10 gigabytes
	Amount of the bandwidth	1 gigabyte per second
	RAM	0.5 gigabyte
	Processing power	9726 MIPS
Workload	Source	The HPC2N Seth log
	Cloudlets's number	From 20 to 200

RL tutorial [35] and modified to fit our defined environment. The training was executed following the parameters in Table 2. The aforementioned parameters were obtained following specific training process execution (for a high score in queue scheduling).

5.1.3. PSO Parameters. The PSO algorithm was applied following the recommended version in [5] with several iterations (equal to 1000), particles (equal to 500), local weights ($c1$ and $c2$) with the same value of 1.49445, and a fixed inertia weight with a value of 0.9.

5.2. Experimental Results and Analysis. Following Figure 9, the MCS-DQN agent average assessment score reflected over 800 episodes. Perceivably, learning remained steady despite approximately 800 training iterations. The ϵ parameter evolution was also incorporated into the ϵ -greedy exploration method during training. Following increased agent scores when ϵ began decaying, MCS-DQN could already generate sufficiently good Q-value estimates for more thoughtful state and action explorations to accelerate the agent learning process.

After the training process, various cloudlet sets were executed with the MCS-DQN scheduler saved model, FCFS, and PSO algorithms for every metric assessment. As every cloudlet of the same set was simultaneously executed, this study essentially emphasised the makespan metric (the elapsed time when simultaneously executing cloudlet groups on available VMs). Figure 10 presents the reduced research makespan compared to other algorithms.

The makespan metric (employed as the primary model training objective) impacted other performance metrics:

- (1) The degree of imbalance (DI) metric demonstrated load-balancing between VMs. Specifically, DI was utilised to compute the incongruence between VMs when simultaneously executing a set of cloudlets. The DI metric reduction was attempted for a more

FIGURE 8: The MCS-DQN network setup.

TABLE 2: Training parameters.

Parameter	Value
Episodes	1000
Discount rate	0.9
Exploration rate max	1.0
Exploration rate decay	0.996
Exploration rate min	0.05
Experience replay queue size	5000
Input size (state size)	5
Output size (actions size)	5

FIGURE 9: Training score.

congruent system. Equation (3) was employed in this research to calculate the DI metric.

$$DI = \frac{E_{\max} - E_{\min}}{E_{\text{avg}}} \quad (3)$$

Specifically, E_{avg} , $E_{\min}$, and $E_{\max}$ implied the average, minimum, and maximum total execution time of all VMs [34]. Figure 11 presents the recommended MCS-DQN scheduler that minimised the DI metric in every utilised set of cloudlets for an enhanced load-balancing system.

FIGURE 10: Makespan.

FIGURE 11: Average DI.

- (2) In the waiting time (WT) metric, the cloudlets arrived in the queue and executed following the scheduling algorithm. For example, the waiting time algorithm was applied to compute all cloudlet sequence and average waiting time measures (see equation (4)). Specifically, WT_i denoted the cloudlet waiting time while i and n reflected the queue length:

$$WT_{avg} = \frac{\sum_{i=1}^n WT_i}{n}. \quad (4)$$

Following Figure 12, the recommended MCS-DQN scheduler could efficiently provide an optimal alternative to heighten the cloudlet queue management speed and effectiveness by reducing cloudlet waiting time and queue length.

- (3) The RU metric proved vital for elevated RU in the CS process.

Equation (5) is employed to compute the average RU [34].

$$ARU = \frac{\sum_{i=1}^n ET_{VM_i}}{\text{Makespan} \times N}. \quad (5)$$

Specifically, ET_{VM_i} denoted the VM_i duration to complete all cloudlets while N reflected the number of resources. In Figure 13, the recommended MCS-DQN scheduler was more improved than PSO and FCFS regarding RU. Specifically, the MCS-DQN scheduler ensured busy resources while CS (as service providers) intended to earn high profits by renting restricted resources.

Furthermore, to prove the effectiveness of our proposed work, more executions based on the same previous VMs configurations were conducted. Figure 14 illustrates the results of these executions where we increased the number of virtual machines to 10, 15, 20, and 30, respectively. In each

FIGURE 12: Average waiting time.

set of VMs, we scheduled a number of tasks equal to 60, 140, and 200, respectively. These experiments were done to the makespan since it is our main metric and compared with the PSO, the chosen scheduling algorithm in this work. We notice that our proposed MCS-DQN algorithm performance is still better than the PSO scheduler even when adding more experiments.

However, our suggested approach is restricted to a set number of virtual machines, and any change in the number of virtual machines requires a new model training. We intend to concentrate on variable-length output prediction in the future such that the number of VMs does not impact the model and no training is necessary for every change in VMs.

FIGURE 13: Average resource utilisation.

FIGURE 14: Increasing the number of virtual machines.

6. Conclusion

This study encompassed effective CS application using deep Q-learning in cloud computing. Additionally, the MCS-DQN scheduler recommended TS problem enhancement and metric optimisation. The simulation outcomes revealed that the presented work attained optimal performance for minimal waiting time and makespan and maximum resource employment. Additionally, the recommended algorithm regarded load-balancing during cloudlet distribution to current resources beyond PSO and FCFS algorithms. This proposed model can be applied to solve task scheduling problems in cloud computing, specifically in cloud broker. To solve the limitation of fixed VMs, we plan in the future to enhance our work by relying on variable-length output prediction using dynamic neural networks to include various VM sizes, as well as adding other optimisation

approaches, taking into account more efficiency metrics such as task priority, VM migration, and energy consumption. Furthermore, assuming that (n) tasks are scheduled to (m) fog computing resource, we can apply adjustments into the proposed algorithm to work on the edge computing; this may also be an idea for future work.

Data Availability

The data for this research are available in the “Parallel Workloads Archive: HPC2N Seth”: https://www.cse.huji.ac.il/labs/parallel/workload/l_hpc2n/.

Conflicts of Interest

The authors declare no conflicts of interest.

Authors’ Contributions

All of the authors participated to the article’s development, including information gathering, editing, modelling, and reviewing. The final manuscript was reviewed and approved by all of the authors.

Acknowledgments

Laboratory of Emerging Technologies Monitoring in Settat, Morocco, provided assistance for this study.

References

- [1] I. Foster, Y. Zhao, I. Raicu, and S. Lu, “Cloud computing and grid computing 360-degree compared,” in *Proceedings of the 2008 grid computing environments workshop*, pp. 1–10, IEEE, Austin, Texas, November 2008.
- [2] M. A. Rodriguez and R. Buyya, “A taxonomy and survey on scheduling algorithms for scientific workflows in iaas cloud computing environments,” *Concurrency and Computation: Practice and Experience*, vol. 29, no. 8, Article ID e4041, 2017.
- [3] J. Kennedy and R. Eberhart, “Particle swarm optimization,” in *Proceedings of the ICNN’95-International Conference on Neural Networks*, IEEE, Perth, WA, Australia, December 1995.
- [4] W. Wong and C. I. Ming, “A review on metaheuristic algorithms: recent trends, benchmarking and applications,” in *Proceedings of the 2019 7th International Conference on Smart Computing & Communications (ICSCC)*, pp. 1–5, IEEE, Miri, Malaysia, June 2019.
- [5] H. S. Al-Olimat, M. Alam, R. Green, and J. K. Lee, “Cloudlet scheduling with particle swarm optimization,” in *Proceedings of the 2015 Fifth International Conference on Communication Systems and Network Technologies*, pp. 991–995, IEEE, Gwalior, India, April 2015.
- [6] N. Zhang, X. Yang, M. Zhang, Y. Sun, and K. Long, “A genetic algorithm-based task scheduling for cloud resource crowd-funding model,” *International Journal of Communication Systems*, vol. 31, no. 1, Article ID e3394, 2018.
- [7] Q. Guo, “Task scheduling based on ant colony optimization in cloud environment,” *AIP conference proceedings*, vol. 1834, p. 040039, 2017.
- [8] X. Huang, C. Li, H. Chen, and D. An, “Task scheduling in cloud computing using particle swarm optimization with time

- varying inertia weight strategies,” *Cluster Computing*, vol. 23, no. 13, pp. 1–11, 2019.
- [9] Y. Liang, Q. Cui, L. Gan, Z. Xie, and S. Zhai, “A cloud computing task scheduling strategy based on improved particle swarm optimization,” in *Proceedings of the 2020 2nd International Conference on Big Data and Artificial Intelligence*, pp. 543–549, New York, NY, USA, April 2020.
 - [10] M. S. Ajmal, Z. Iqbal, M. B. Umair, and M. S. Arif, “Flexible genetic algorithm operators for task scheduling in cloud datacenters,” in *Proceedings of the 2020 14th International Conference on Open Source Systems and Technologies (ICOSST)*, pp. 1–6, IEEE, Lahore, Pakistan, December 2020.
 - [11] N. Musa, A. Y. Gital, F. U. Zambuk, A. M. Usman, M. Almutairi, and H. Chiroma, “An enhanced hybrid genetic algorithm and particle swarm optimization based on small position values for tasks scheduling in cloud,” in *Proceedings of the 2020 2nd International Conference on Computer and Information Sciences (ICCIS)*, pp. 1–5, IEEE, Sakaka, Saudi Arabia, October 2020.
 - [12] N. Yi, J. Xu, L. Yan, and L. Huang, “Task optimization and scheduling of distributed cyber-physical system based on improved ant colony algorithm,” *Future Generation Computer Systems*, vol. 109, pp. 134–148, 2020.
 - [13] Z. Peng, B. Barzegar, M. Yarahmadi, H. Motameni, and P. Pirouzmmand, “Energy-aware scheduling of workflow using a heuristic method on green cloud,” *Scientific Programming*, vol. 2020, Article ID 8898059, 2020.
 - [14] C. Saravanakumar, M. Geetha, S. Manoj Kumar, S. Manikandan, C. Arun, and K. Srivatsan, “An efficient technique for virtual machine clustering and communications using task-based scheduling in cloud computing,” *Scientific Programming*, vol. 2021, Article ID 5586521, 2021.
 - [15] Y. Sun and X. Qi, “A de-ls metaheuristic algorithm for hybrid flow-shop scheduling problem considering multiple requirements of customers,” *Scientific Programming*, vol. 2020, Article ID 8811391, 2020.
 - [16] C. Huang, G. Huang, W. Liu, R. Wang, and M. Xie, “A parallel joint optimized relay selection protocol for wake-up radio enabled wsns,” *Physical Communication*, vol. 47, Article ID 101320, 2021.
 - [17] J. Ge, B. Liu, T. Wang, Q. Yang, A. Liu, and A. Li, “Q-learning based flexible task scheduling in a global view for the internet of things,” *Transactions on Emerging Telecommunications Technologies*, vol. 32, Article ID e4111, 2020.
 - [18] D. Ding, X. Fan, Y. Zhao, K. Kang, Q. Yin, and J. Zeng, “Q-learning based dynamic task scheduling for energy-efficient cloud computing,” *Future Generation Computer Systems*, vol. 108, pp. 361–371, 2020.
 - [19] B. Zhang, W. Wu, X. Bi, and Y. Wang, “A task scheduling algorithm based on q-learning for wsns,” in *Proceedings of the International Conference on Communications and Networking in China*, pp. 521–530, Springer, Shanghai, China, December 2018.
 - [20] H. Che, Z. Bai, R. Zuo, and H. Li, “A deep reinforcement learning approach to the optimization of data center task scheduling,” *Complexity*, vol. 2020, Article ID 3046769, 2020.
 - [21] T. Dong, F. Xue, C. Xiao, and J. Li, “Task scheduling based on deep reinforcement learning in a cloud manufacturing environment,” *Concurrency and Computation: Practice and Experience*, vol. 32, no. 11, Article ID e5654, 2020.
 - [22] F. Qi, L. Zhuo, and C. Xin, “Deep reinforcement learning based task scheduling in edge computing networks,” in *Proceedings of the 2020 IEEE/CIC International Conference on Communications in China (ICCC)*, pp. 835–840, IEEE, Chongqing, China, August 2020.
 - [23] M. Chen, T. Wang, S. Zhang, and A. Liu, “Deep reinforcement learning for computation offloading in mobile edge computing environment,” *Computer Communications*, vol. 175, pp. 1–12, 2021.
 - [24] P. Wang, C. Zhao, Y. Wei, D. Wang, and Z. Zhang, “An adaptive data placement architecture in multicloud environments,” *Scientific Programming*, vol. 2020, Article ID 1704258, 2020.
 - [25] Z. Zang, W. Wang, Y. Song et al., “Hybrid deep neural network scheduler for job-shop problem based on convolution two-dimensional transformation,” *Computational Intelligence and Neuroscience*, vol. 2019, Article ID 7172842, 2019.
 - [26] J. Wu, G. Zhang, J. Nie, Y. Peng, and Y. Zhang, “Deep reinforcement learning for scheduling in an edge computing-based industrial internet of things,” *Wireless Communications and Mobile Computing*, vol. 2021, Article ID 8017334, 2021.
 - [27] Y. Wang, H. Liu, W. Zheng et al., “Multi-objective workflow scheduling with deep-q-network-based multi-agent reinforcement learning,” *IEEE access*, vol. 7, pp. 39974–39982, 2019.
 - [28] V. Mnih, K. Kavukcuoglu, D. Silver et al., “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.
 - [29] S. Jain, P. Sharma, J. Bhoiwala et al., “Deep q-learning for navigation of robotic arm for tokamak inspection,” in *Proceedings of the International Conference on Algorithms and Architectures for Parallel Processing*, pp. 62–71, Springer, Xiamen, China, December 2018.
 - [30] S. He and W. Wang, “Pricing qoe with reinforcement learning for intelligent wireless multimedia communications,” in *Proceedings of the ICC 2020-2020 IEEE International Conference on Communications (ICC)*, pp. 1–6, IEEE, Dublin, Ireland, June 2020.
 - [31] L. J. Lin, “Reinforcement learning for robots using neural networks,” Tech. rep., Carnegie-Mellon Univ Pittsburgh PA School of Computer Science, Pittsburgh, 1993.
 - [32] A. Karthick, E. Ramaraj, and R. G. Subramanian, “An efficient multi queue job scheduling for cloud computing,” in *Proceedings of the 2014 World Congress on Computing and Communication Technologies*, pp. 164–166, IEEE, Trichirappalli, India, March 2014.
 - [33] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, “Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms,” *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2011.
 - [34] M. Kalra and S. Singh, “A review of metaheuristic scheduling techniques in cloud computing,” *Egyptian informatics journal*, vol. 16, no. 3, pp. 275–295, 2015.
 - [35] M. Plappert: keras-rl, 2016, <https://github.com/keras-rl/keras-rl>.