Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 184 (2021) 42–51

Procedia Computer Science

www.elsevier.com/locate/procedia

The 12th International Conference on Ambient Systems, Networks and Technologies (ANT)
March 23-26, 2021, Warsaw, Poland

Task Scheduling in Cloud Using Deep Reinforcement Learning

Shashank Swarup[a,*], Elhadi M. Shakshuki[a], Ansar Yasar[b]

[a]*Jodrey School of Computer Science, Acadia Unversity, Wolfville, Nova Scotia, B4P2R6, Canada*
[b]*Transportation Research Institute, B-3500 Hasselt, Hasselt University, Belgium*

Abstract

Cloud computing is an emerging technology used in many applications such as data analysis, storage, and Internet of Things (IoT). Due to the increasing number of users in the cloud and the IoT devices that are being integrated with the cloud, the amount of data generated by these users and these devices is increasing ceaselessly. Managing this data over the cloud is no longer an easy task. It is almost impossible to move all data to the cloud datacenters, and this will lead to excessive bandwidth usage, latency, cost, and energy consumption. This makes it evident that allocating resources to users' tasks is an essential quality feature in cloud computing. This is because it provides the customers or the users with high Quality of Service (QoS) with the best response time, and it also respects the established Service Level Agreement. Therefore, there is a great importance of efficient utilization of computing resources for which an optimal strategy for task scheduling is required. This paper focuses on the problem of task scheduling of cloud-based applications and aims to minimize the computational cost under resource and deadline constraints. Towards this end, we propose a clipped double deep Q-learning algorithm utilizing the target network and experience relay techniques, as we as using the reinforcement learning approach.

Keywords: task scheduling; computational cost; energy consumption; deep reinforcement learning; Clipped Double Deep Q-learning (CDDQL).

1. Introduction

Due to the rapid increase in the deployment of applications over the cloud, there is a large amount of raw data that requires real-time processing and analysis. According to Statista [14], computing instances are expected to have the

* Corresponding author. Tel.: 1(902)585-1524; fax: 1(902)585-1067.
E-mail address: shashankswarup@acadiau.ca

1877-0509
10.1016/j.procs.2021.03.016

highest share of global data center storage capacity by 2021, with capacity forecast to reach around 470 exabytes by then. It is stated that 60% of the workloads are running on hosted cloud service as of 2019 and will increase to 90% by 2021. To process such huge data by utilizing the cloud resources efficiently, scheduling the tasks which are generated by those applications, is extremely important. One of the most important research directions in computer paradigms of resource management is through task scheduling. Due to the uncertainty and complexity of the networks, it is not easy to develop efficient algorithms to tackle complex problems in different network scenarios. Machine learning (ML) approaches are known for dealing with complex problems and help in decision making, which consequently facilitates resource management and task scheduling. Machine learning helps in constructing models that learn and make decisions directly from data without following predefined rules. Most of the ML-based resource management schemes used supervised learning and reinforcement learning. Recently, the combination of deep learning (DL) and reinforcement learning have shown acceptable results [1]. The characteristics of deep learning hold the capability to learn continuously and produce good learning models. Hence, deep reinforcement learning has a strong capability to deal with decision-making problems, which in turn deal with resource management problems. In this paper, deep reinforcement learning (DRL) approach called clipped double deep Q-learning (CDDQL) is utilized.

In this paper, we aim to combine Q-learning and clipped double deep Q-learning algorithms where the Q-learning sets the initial values using random actions for the clipped double deep Q-learning through the q-table or matrix, and the updated q-table for each episode. We summarize our contributions as follows:

- The task scheduling problem is modeled as Markov Decision Process (MDP) in actor critic method to define interactions between task scheduler and cloud data center. By using this approach in a cloud resource allocation scenario, we aim to reduce computational costs under deadline and resource constraints.
- Task scheduling algorithm is developed by combining Deep Q-learning, Clipped Double Q-learning (a modified version of double q-learning), target networks, and experience replay techniques.
- The efficiency of our proposed algorithm is evaluated and validated. Both cost and time metrics are used. The performance of our proposed approach is compared with baseline algorithms such as time-shared scheduling, space-shared scheduling, random scheduling, and Q-learning Scheduling (QLS).

The rest of the paper is organized as follows. Section 2 provides related works. Section 3 describes the system model and formulation. Section 4 discusses the proposed algorithm, followed by the experimental results in section 5.

2. Related Work

Resource allocation through task scheduling is an important field of research in cloud computing. Researchers have proposed many attempts to address this issue. Using algorithms for task scheduling is the most attractive approach for researchers to deal with resource allocation in the cloud [2, 4, 5, 7, 9, 10, 12, 18, 19]. Machine learning (ML) approaches have proven to perform well in dealing with complex issues. A surveyed study in [18] identified seven approaches with respect to performance-driven self-adaptaion supported by queuing networks. Th study [18] concluded that optimization techniques such as Machine learning, Search-Space Exploration and Mixed Integer Programming, can be exploited to optimize a fitness function involving performance indices. Cloud is a complex environment, where allocation of tasks to resources is a complex job. Hence, ML can be considered in dealing with resource allocation issues in the cloud. Among various machine learning approaches, reinforcement learning with deep learning has shown convincing results. Zhiping Peng et al in [4] used Q-Learning based task scheduling with immediate reward function and state aggregation to coordinate optimization problem in multiple VMs. The authors in [4] focus was on reducing the response time in the cloud computing environment. However, using a discrete reward function cannot show the exact effect of each action. A QL-based task scheduling scheme in a grid or IaaS cloud is used in [5]. Additionally, during the evaluation metrics, the authors focused on minimizing average waiting time and optimize resource utilization. A unique study of assessing a fog model's performance by using simulation with three scheduling with combinations of three load balancing strategies has been proposed in [19]. The researchers in [19] have also concluded that AI and deep learning models can assist in task placement decision.

Deep reinforcement learning approaches have a great effect in several applications such as video games [6], cooling datacenters [7], etc. According to [8], the DeepRM method which is described in [9] is the first work that has applied

deep reinforcement learning (DRL) algorithm for cluster job scheduling. They have taken a policy network that takes a collection of distinct images as inputs and then outputs a probability distribution over all possible actions. Then, they selected the average job slowdown as an objective metric. In [10], the authors have proposed a DQN- based task scheduling algorithm which is a task-specific. The proposed algorithm deals with the characteristics of the server, tasks being regarded as state inputs, and numbers being regarded as actions. The authors in [10] proposed a DQN-algorithm that sorts the tasks in upward ranking. Then, the algorithm decides the scheduling order of the tasks and progress by using two neural networks: target network and evaluation network. The target network has the target Q-value which is the maximum Q-value of the next predicted action in the upcoming state and the evaluation network evaluates the action taken in the state by computing the q-value with the reward assigned to the action. The output of each network is action distribution and the probability of choosing an action. The authors in [10] focused on minimizing the response time. Since the algorithm involves complex tasks like sorting and ranking for each iteration of task scheduling. By looking at the performed evaluation in [10], it is found that the algorithm can take more time and cost to execute. Nevertheless, other parameters such as the deadline of tasks and cost constraints that are important in a real cloud environment are overlooked.

Xianfu Chen et al. in [11] have focused on parallel offloading in radio access networks by formulating it as Markov Decision Process and derived a double deep Q-network based reinforcement learning algorithm. They have resort to a deep neural network-based function approximator to deal with state-space explosion and learn from the optimal computational offloading policy without any prior knowledge. The authors in [11] have proposed a State-Action-Reward-State-Action (SARSA) based deep learning algorithm to achieve the best computation offloading performance. SARSA has a slight variation of the Q-learning algorithm, as represented in Equation (1). While Q-learning is an off-policy approach that uses the greedy approach to learn Q-value, SARSA is an on-policy approach that uses action performed by current policy to learn Q-value. They also claimed that their contribution is the first work to combine the Q-function decomposition technique with the double DQN. However, they have eliminated the transmission and the propagation delay.

$$Q(s_t,a_t) = Q(s_t,a_t)+\alpha(r_{t+1}+\gamma Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)) \quad (1)$$

In [12], the researchers have modified the Q-learning algorithm with user-defined parameters such as throughput, the number of VMs, and latency to ensure fairness in resource allocation. Resources are divided based on properties like Pareto efficiency, incentive sharing, envy-freeness, and strategy-proofness. The distribution was performed in fairness scheme so that all the users requested for resources will get a guaranteed resource. In [13], the authors used the Double Deep Q-learning algorithm for scheduling by allocating fog nodes with VMs. The idea in [13] of using fog nodes and assigning multiple schedulers to run in parallel can decrease the execution time effectively. In [17], the authors have introduced double Q-learning by using two Q-networks where each one updates another. The research in [17] proved to be effective in reducing overestimation and maximization bias. The authors in [13] proposed a double deep Q-learning scheduling algorithm to schedule the incoming user tasks. They have also used several hyperparameters' analysis that can achieve the least average service delay. However, they didn't mention how the VMs in the datacenter are allocated to fog nodes. Moreover, no discussion as to how or on what basis the fog nodes are created. In their approach, giving a proper analysis of the algorithm's performance is restricted although a standard environment with predefined fog nodes, number of user tasks, and number of virtual machines are provided. In [20], clipped double Q-learning has been introduced. The researcher in [20], has used two Q networks for computing the update targets and taking minimum of next two next-state action values produced by the two Q-networks. In the clipped double Q-learning algorithm [20], when the Q- estimate of one is greater than the other, we reduce it to minimum to avoid overfitting. Our work modifies the clipped double Q-learning algorithm with deep learning techniques to schedule tasks to resources in cloud.

3. Proposed Model

This section provides a detailed description of our proposed Algorithms. The scheduling scheme is divided into two parts. The first part is related to the configuration of the cloud's environment and the agents, and this is the subject of Algorithm-1. Algorithm-2 explains the proposed clipped double deep Q-learning used in this research. This is the

main algorithm that schedules the tasks to their respective virtual machines based upon action selection done in Algorithm-3. Algorithm-3 deals with selecting action based on the ε value to achieve a trade-off between exploration and exploitation.

3.1. Configuration of Environment and Agents:

All our proposed algorithms for the scheduling scheme are first described as a container in Algorithm-1, which contains all the necessary pre-processes for the other proposed algorithms. Firstly, we create the network topology that defines the count of datacenters, hosts, and VMs. Secondly, we configure all of the VMs and determine the value of each VM according to its features. Thirdly, we determine the paths in the network topology. Finally, we calculate the total number of resources (CPU, Storage, and Memory) in each path of the network by getting the sum of each resource separately. In our proposed work, we describe the scheduler as an agent.

In our approach, two separate Q-models are created for an agent, namely: Q1 and Q2. Where Q1 is used for action selection according to Algorithm 3, and Q2 is used for action evaluation by calculating Q-values (using the Bellman equation). After that, we create the agents and assign them their models and parameters as per Epsilon Decay Rate (EDR), which is explained in Algorithm-3. These parameters include state space, action space, and step value. Consequently, we will be able to determine the size of the replay memory. Replay memory is the storage space that stores historical Q-value data that is necessary to update the Q-network Q1 from the target Q network Q2. At the start of the algorithm, Q1 and Q2 are initialized with random weights. Since the weights are learnable parameters in neural networks, they use error information to update connection weights. At the beginning of each episode, each agent starts its activities from the initial state. Therefore, we reset the environment to its initial state and reset the timer to 0 msec. We also reset each agent's cumulative reward to 0 at the beginning of each episode. When the initialization is completed, it is possible to run the agents in this environment simultaneously, using Algorithm-2.

Algorithm-1: *Configuration of Environment and Agents*

1. **Input**: *tvm*, *no_of_agents*, *possible_actions*, *rm_capacity*, *no_of_episodes*, broker // **tvm represents total number of virtual machines and rm_capacity represents replay memory capacity**
2. *// Create network topology*
3. **for** *i*=1 to *tvm* **do // Gather all virtual machines necessary information from the datacenter**
   2.1 *Configure* VMs (ID_i, CPU_i, $Memory_i$, $Storage_i$, $PIdle_i$, $PRun_i$, $Cost_i$)
4. end **for**
5. **for** *i*=1 to *tvm* **do**
   4.1. *Submit* VM information to broker
6. end **for**
7. **for** *j*=1 to *no_of_agents* **do // create the Q-network agents responsible for scheduling**
   7. 1. *Determine* $path_j$ in network topology // **broker path is set to datacenter and incoming user tasks**
   7. 2. *Determine* total resources of the agent in $path_j$ (total_CPU, total_Memory, total_Storage) **// broker obtains necessary information about the VMs**
   7. 3. *Select* scheduler // **broker selects the scheduler for scheduling the tasks based on VM information obtained in step 7.2**
   7. 4. *Create* $agent_j$*'s* Q_1 and Q_2 models //**the term *agent* refers to scheduling or learning algorithm used by the scheduler**
8. end **for**
9. **for** *k*=1 to *no_of_agents* **do // set weights for the Q-networks and Q-table limit as rm_capacity**
   9. 1. *Set* replay memory to rm_capacity
   9. 2. *Initialize* Q_1 with random weight w_1 for $agent_k$ // **weight is assigned for Q_1's learning *agent. agent* refers the learning or the scheduling algorithm.**
   9. 3. *Initialize* Q_2 with w_2 for $agent_k$ // **weight is assigned for Q_2's learning agent. These agent weights (*w1* and *w2*) help in updating the evaluation network with values in target network.**
   9. 4. $step_k = 0$

10. end **for**
11. // *Create and initialize cloud's problem environment*
12. **for** *l*=1 to *no_of_episodes* **do // every episode is considered as a fresh start. Number of episodes help in converging the difference between target and evaluated Q-network**
    12.1. *Reset* the environment to initial state
    12.2. *Run* all the agents according to Algorithm-2
13. end **for**

3.2. Clipped Double Deep Q-Learning (CDDQL) Scheduling:

The CDDQL scheduling algorithm described in Algorithm-2 is the core of the proposed model. It depicts the internal operations of each agent.

Algorithm-2: *Clipped Double Deep Q-Learning (CDDQL) Scheduling*

1. **Input**: *no_of_agents*, *total_tasks*, *possible_actions*, *EDR*, ε, *Mini_Batch_Size*, γ, *Target_Update_Frequency*
   // EDR is epsilon decay rate whose ε value is usually set to 0.5
2. **Output**= *Cumulative_Reward*
3. **for** *i*=1 to *no_of_agents* **do**
   3.1. *Set Cumulative_Reward* to 0 // **initial cumulative reward is zero**
   3.2. *Shuffle* the primitive task set
   3.3. **for** *j*=1 to *total_tasks* **do // for every set of tasks, analyse its state**
      3.3.1. $step_k = step_k + 1$// Increment the step
      3.3.2. *Calculate* s_k^i = (CPU, Memory, Storage)
      3.3.3. *Set activation_flag* to *false*
      3.3.4. **while** activation_flag is false **do //select action based on epsilon value**
         3.3.4.1. *Select* a_j^i // Using Algorithm-4
         3.3.4.2. *Validate* a_j^i and *set activation_validation_flag* accordingly
         3.3.4.3. **if** *activation_validation_flag==true* **then // the validation flag means that the action is being evaluated and no other action can take place until the selected action is executed**
            3.3.4.3.1. Break
         3.3.4.4. end **if**
      3.3.5. end **while**
   3.4. *Allocate* vm_j^a to $task_j$ and *set* status as busy.
   3.5. *Execute* $task_j$ and *calculate* execution time
   3.6. *Calculate* waiting time of $task_j$ in $path_i$
   3.7. *Release* vm_j^a and set status as *idle*
   3.8. *Calculate Immediate_Reward* of $action_a$ at $task_j$ in $path_i$
   3.9. *Set* $Cumulative\_Reward_j$ to $Cumulative\_Reward_j + Immediate\_Reward_j^i$ (*action*)
   3.10. *Add* transition tuple (s_j^i, a_j^i, r_j^i, s'_j^i) into replay memory // **(present state, action, reward, next state)**
   3.11. *Update* ε_j^i // Using ε_j^i =0.5
   3.12. *Get* random *Mini_Batch_Size* from the transition tuples
   3.13. *Calculate* target Q-value of action *a* by $agent_i$ to perform $task_j$ // Using Equation (2)
   3.14. *Train* Q_1^i and *perform* gradient descent step
   3.15. *Select* the minimum Q-value from the two maximum Q-values obtained in Q_1^i and Q_2^i
   3.16. **if** $step_k$ % *Target_Update_Frequency* is 0 **then**
      3.16.1. *w2=w1* //**update the targets so that the Q-table is updated frequently**
   3.17. end **if**
   3.18. end **for**

4. end **for**
5. **Return** *Cumulative_Reward*

For the purpose of generalization, at the beginning of each episode, we reset the cumulative reward of agents to 0 and shuffle the primitive task set. In order to avoid making the wrong decision by the agent before taking an action, it validates the output using ε- greedy policy. The validation process is essentially based on the comparison between the selected resources and the required resources of the current task. If the action is not valid, then the agent runs ε- greedy policy again until the *activation_validation_flag* changes to true.

$$Q^*(s_t,a_t) = \mathbb{E}_{s`}\,[R_{t+1}+\gamma\ \max_{a`}\ Q^*(s`_t,a_t`)|\ s,a] \tag{2}$$

Where $\mathbb{E}_s$ is the sum of the Q-value of the state-action pair and learning rate α. Learning rate specifies how reliable the algorithm should be on the unbiased reward value. γ represents a discount factor that controls the reaction of immediate and future rewards

After the initialization process is completed, the incoming task is sent to the cloud server where the selected VM is located. If the selected VM is idle, then the task is immediately allocated. If the selected VM is busy executing another task, the incoming task is appended to the VM's waiting queue. After allocating VM with a task, the agent calculates the next state of the environment and the execution time. In the next step, it calculates the waiting time of the current task. The proposed algorithm is non-preemptive and can run only one task in each of the VMs.

Upon the completion of the task, the VM is released and output data is returned to its designated destination. To calculate the immediate reward of the action, the agent normalizes the elements of service delay individually and takes the sum of the normalized elements. And finally, it uses *Immediate_Reward* = 1/ (sum of the normalized elements) to calculate the immediate reward. The sum of the normalized elements is the sum of waiting time, execution time, total (upstream and downstream) transmission time, and total (upstream and downstream) propagation time. Then, the agent adds the immediate reward to the cumulative reward of the current episode and adds the transition tuple of the current experience to replay memory.

After each step, the agent updates the value of ε using this formula $\varepsilon_j^i = (\varepsilon_{start} - \varepsilon_{end}) * e^{-EDR*step_k}$. To train the Q_1 model; first, the agent gets the random mini batch size of *Mini_Batch_Size* from experiences and uses the current state (s) values as training samples (X) and the next state (s') values to calculate the target values (Y). Then, the Q_1 model of the agent performs a gradient descent step on loss and updates Q_1 weights as *w1*. At last, it updates all weights of Q_2 model, *w2* by using *w1* at each step of *Target_Update_Frequency*. This process is used for every task that enters the agent. The target Q-value is calculated by using CDDQL Equation (3).

$$Q_1^*(s_t,a_t) = r_t + \gamma\ min_{i=1,2}\ Q_1\ (s_{t+1},\ argmax_a,\ Q_1(s_{t+1},a')) \tag{3}$$

Where, γ is the discount factor. As the discount factor quantifies the importance of the algorithm that gives future rewards, we set the value of γ to 0.99. This helps the algorithm to consider future rewards with high weights. It is important to update network parameters for each iteration. Maintaining the updated target networks can decrease the growth of approximation error. An error function reports back the difference between the estimated reward of any given state or time step and the actual reward. Such an error is known as the temporal difference (TD) error. To reduce the TD error, it is important to update the target network with the primary network regularly. So, we update the target network parameters by using Equation 4 [3].

$$\theta` = \tau*\theta + (1-\tau)*\theta` \tag{4}$$

Where, θ` is the target network parameter and it is updated by the primary network parameter θ. τ is the rate of averaging that is set to 0.01. The target networks use a slow-moving updated rate, parameterized by τ. To update a randomly selected state without a target network τ is set to 1. But in our considered case, we need to update the target networks slowly. This helps us to decrease the TD error and free the algorithm from overestimation bias (if the error is vanished; it means the algorithm has overestimated). Every reinforcement learning algorithm is like chasing a target while the target keeps moving. We need to make sure that the difference between the estimated value and the target

value is not large, but the difference must exist to ensure overestimation is not being done. So, we update the target network parameter at a slow rate of $\tau = 0.01$.

Algorithm-3: *ε- greedy policy*

1. **Input**: present state (s^i_j), ε_j^i , *possible_actions*
2. **Output**: selected action (a_j^i)
3. *Select random_number* in range [0,1]
   3.1. **If** *random_number* < ε_j^i **then** //**exploration, ε value is 0.5**
   3.2. Select a_j^i in range [1, *possible_actions*] randomly
4. **Else**
   4.1. *Calculate* a_j^i // Using Equation (2) // **exploitation**
5. end **if**
6. Return a_j^i

This algorithm is used as an action selection strategy. The ε-greedy policy is used to the trade-off between exploration and exploitation. The only difference between QLS and CDDQLS is that in CDDQLS, the agent performs exploration using Q_1 model to select the action with maximum Q-value and Q_2 for calculating the Q-values. For QLS algorithm, it uses only one Q-model for both action selection and validation by referring to Q-table. The CDDQLS uses Equation (5) for action selection.

$$a_j^i = \text{argmax}_a \, Q_1(s_{t+1}, a')_j^i \tag{5}$$

4. Evaluation

4.1. Simulation Environment

To evaluate the efficiency of the proposed algorithm, Cloudsim [15] with Cloudsim plus [16] java-based cloud simulation framework is utilized to create and configure the cloud environment. Python is used to work with the proposed deep reinforcement learning algorithm in the cloud environment created in Cloudsim [15]. We used the Google task events dataset to generate user tasks. In our simulations, we used four Virtual Machines with 500MB RAM and 1000MB storage. Virtual machine ranges from 213 to 431MB are used for a task to execute. With the Google task events dataset, nine user tasks are generated. As reinforcement learning needs to learn the environment through exploration, random scheduling is performed for the first set of tasks, and the efficiency of each virtual machine is judged. Based upon the simulation results after performing random scheduling, the average computational efficiency (Megabytes/second) of each virtual machine is calculated. With four virtual machines, we created four categories of reward allocation to each virtual machine. If the VM with the best computation capability is considered, the action is rewarded with +2. When the action schedules the task to the second-best VM, it is rewarded with +1. If the user task is allocated to the slowest functioning VM, the action is given a penalty of -2. If the action selects VM with the third fastest computation capability, it is given a penalty of -1.

In the proposed deep reinforcement learning-based scheduling, an action that schedules a task to a VM is rewarded. This means that VMs are assigned with rewards. An action of selecting a VM is assigned with rewards or penalties, not the task as such the scheduling process is non-preemptive.

The deep Q- network in the CDDQL algorithm contains a sequential model with 4 dense layers, an input, and an output layer along with ReLu activation with Adam optimization.

4.2. Simulation Results

Figure 1 shows the time taken to execute each user task for the following scheduling: time shared scheduling (blue), space shared scheduling (orange), random selection scheduling (grey), Q-learning scheduling (yellow), and our proposed CDDQLS scheduling (light blue).

In a cloud environment, the incoming tasks vary in size. Thus, random sizes are taken from the Google Task Events

dataset in Cloudsim plus [16] and randomly distributed in chunks of user tasks. The sizes of tasks ranges from 1 to 4 are equal to 431MB, tasks from 5 to 6 are equal to 213MB, tasks from 7 to 8 are equal to 228MB, and the task size of the 9th task is equal to 248MB. It should be noted that there is a big difference between the execution time of the first four tasks and the other tasks due to the fact that of the size difference between them.

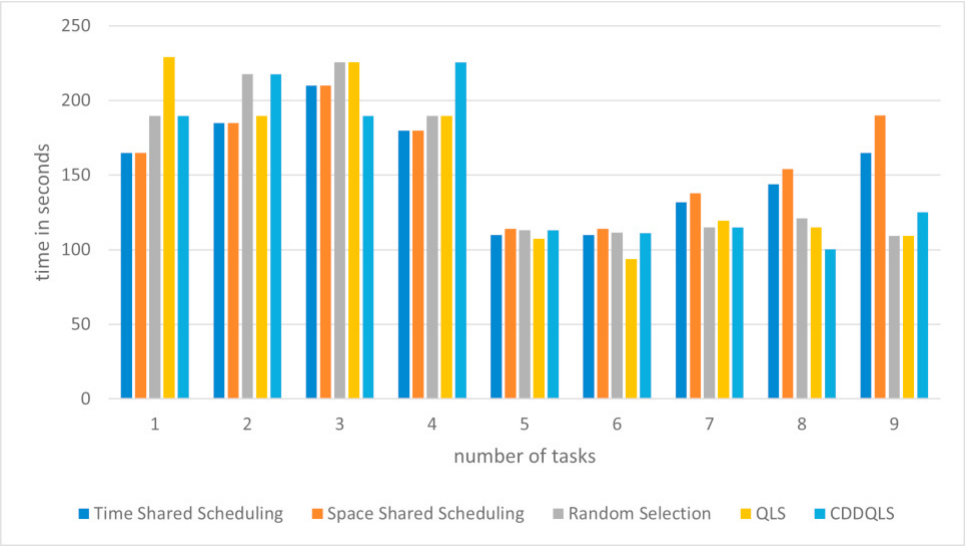

Fig. 1. Execution time of each user task.

In many cases, we can notice that the results of random scheduling show similar efficiency to QLS and CDDQLS scheduling. The QLS and CDDQLS scheduling select the actions based on ε- greedy strategy. Whenever the random value is less than ε value (i.e., 0.5), random action is performed. This means in many cases tasks are scheduled to VMs randomly. This can help the learning agent to explore the environment; however, it leads to a decrease in the efficiency of the scheduling.

To visualize the efficiency of the proposed algorithm clearly, we took a batch of nine tasks from the Google task-events dataset and used them as inputs to various scheduling algorithms five times. The size of each task is the same as the size used previously. The evaluation results that visualize the algorithms' performance when the batches of tasks are fed to input is shown in Figure 2.

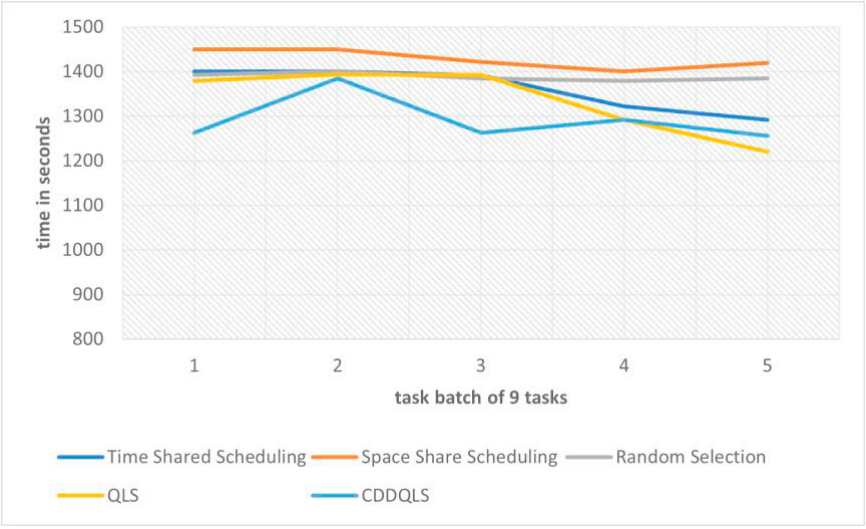

Fig. 2. Execution time of 5 batches with 9 tasks each.

The evaluation results from Figure 2 show that the proposed algorithm performs better than the other algorithms in accordance with the total execution time of a batch of nine tasks. This is because the Q-table continuously updates itself for each episode or epoch to learn the environment.

The average service delay in the reinforcement learning approach is effectively dependent on what extent the learning agent knows about the environment. This evaluation metric is inspired by [2]. Exploration and exploitation need to go together to fetch healthy results leading to optimal resource allocation. The selection between exploration and exploitation is dependent on the ε value. In our approach, we choose the value of ε equal to 0.5. Figure 3 shows how the waiting time of a batch of 9 tasks changes with respect to the ε value that effecting the service delay. The average service delay is calculated by taking the average waiting times for each task.

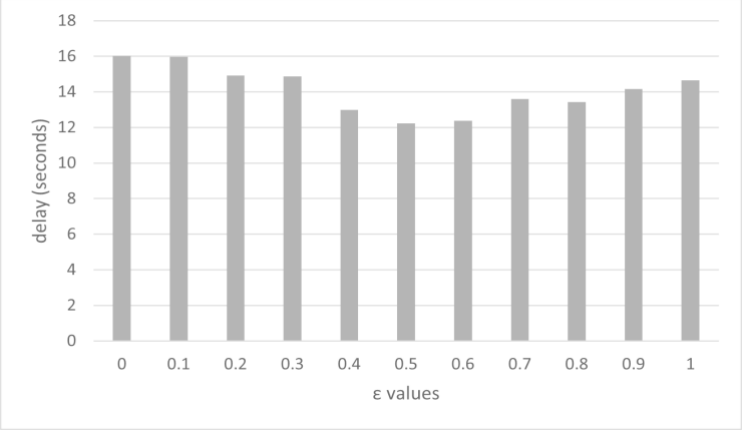

Fig. 3. Waiting time or delay of a batch of 9 tasks with respect to changing ε values.

From Figure 3 we can understand that on both ends the delay is high when ε value is too low or too high. The proposed algorithm is designed in such a way that actions are chosen based on the ε value. Every time a randomly chosen integer is less than the ε value (0.5), the q-value is calculated by choosing a random (next state, action) value from the q-table, as demonstrated in Algorithm 3. This decreases the efficiency of the algorithm by not choosing high rewarding action, i.e. an optimal action. This results in scheduling the task to a VM with low execution speed and accordingly delays other tasks in its queue. This explains the reason of the delay that being high when ε=1. As the ε value decreases, the possibilities of the random number being less than ε also decreases. This helps the learning agent to choose maximum (next state, action) q-value when using Equation 2. This allows optimal usage of VMs and decrease in service delay. Nevertheless, when the ε value is 0, it restricts the learning agent to explore the environment and forces the agent to perform Equation 2 in every iteration. This eliminates exploration, which results in poor selection of virtual machine and causing traffic in the VM queue. Hence, it is important to choose an ε value that can result in a good management between exploration and exploitation. This made it natural for us to choose ε equals to 0.5 for our approach.

5. Conclusion and Future Work

In this paper, we studied the task scheduling and VM allocation problem in a cloud environment with the aim of decreasing execution time and cost. To ensure the optimality in using resources, we proposed a Clipped Double Deep Q-learning (CDDQL) based scheduling algorithm. The algorithm focuses on maximization of its objective function, target network, and experience replay techniques.

According to the experimental results, the proposed algorithm performed better than conventional scheduling algorithms, such as time shared and space shared, and a reinforcement Q-learning algorithm. The hyperparameter experiment using ε showed the importance of the trade-off between exploration and exploitation.

In the future, we intend to use better cloud modeling techniques like fog or edge computing with multiple schedulers to decrease the load on cloud datacenter and ensure faster scheduling. Choosing an ε value that accurately deals with choosing between exploration and exploitation is important.

References

[1] Ravichandran S. (2018) "Hands-on Reinforcement Learning with Python: Master Reinforcement and Deep Reinforcement Learning Using OpenAI Gym and TensorFlow", *Packt Publishing Ltd*.

[2] Pegah Gazori, Dadmehr Rahbari, Mohsen Nickray (2019) "Saving time and cost on the scheduling of fog-based IoT applications using deep reinforcement learning approach", *Future Generation Computer Systems Volume 110, September 2020, Pages 1098-1115.*

[3] Scott Fujimoto, Herke van Hoof, David Meger (2018) "Addressing Function Approximation Error in Actor-Critic Methods" *Proceedings of the 35th International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.*

[4] Zhiping Peng, Delong Cui, Jinglong Zuo, Qirui Li, Bo Xu, Weiwei Lin (2015) "Random task scheduling scheme based on reinforcement learning in cloud computing" *Cluster Comput (2015) 18:1595–1607DOI 10.1007/s10586-015-0484-2.*

[5] Delong Cui, Zhiping Peng, Jianbin Xiong, Bo Xu, Weiwei Lin (2015) "A Reinforcement Learning-based Mixed Job Scheduler Scheme for Grid or IaaS Cloud" *DOI 10.1109/TCC.2017.2773078, IEEE Transactions on Cloud Computing.*

[6] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, D. H. I. Antonoglou, D. Wierstra, and M. A. Riedmiller (2015) "Human-level control through deep reinforcement learning", *Nature, 2015.*

[7] J. Gao and R. Evans "Deepmind ai reduces google data centre cooling bill by 40%." *https://deepmind.com/blog/deepmind-ai-reducesgoogle-data-centre-cooling-bill-40/*

[8] Wang M., Cui Y., Wang X., Xiao S., Jiang J. (2018) "Machine learning for networking: Workflow, advances and opportunities" *IEEE Netw., 32 (2) (2018), pp. 92-99, 10.1109/MNET.2017.1700200*

[9] Mao H., Alizadeh M., Menache I., Kandula S. (2018) "Resource management with deep reinforcement learning" *Proceedings of the 15th ACM Workshop on Hot Topics in Networks, IEEE (2016), pp. 50-56, 10.1145/3005745.3005750*

[10] Tingting Dong, Fei Xue, Chuangbai Xiao, Juntao Li (2019) "Task scheduling based on deep reinforcement learning in a cloud manufacturing environment" *https://doi.org/10.1002/cpe.5654*

[11] Xianfu Chen, Honggang Zhang, Celimuge Wu, Shiwen Mao, Yusheng Ji, and Mehdi Bennis (2019) "Optimized Computation Offloading Performance in Virtual Edge Computing Systems via Deep Reinforcement Learning" *IEEE INTERNET OF THINGS JOURNAL, VOL. 6, NO. 3, JUNE 2019*

[12] K. Karthiban, Jennifer S. Raj (2020) "An efficient green computing fair resource allocation in cloud computing using modified deep reinforcement learning algorithm" *Soft Computing (2020) 24:14933–14942 https://doi.org/10.1007/s00500-020-04846-3*

[13] Pegah Gazori, Dadmehr Rahbari, Mohsen Nickray (2020) "Saving time and cost on the scheduling of fog-based IoT applications using deep reinforcement learning approach" *Future Generation Computer Systems Volume 110, September 2020, Pages 1098-1115*

[14] Arne Holst (2020) "Data center storage capacity worldwide: consumer and business segments 2016-2021" *www.statista.com*

[15] Calheiros, R. N.; Ranjan, R.; Beloglazov, A.; Rose, C. A. F. D.; and Buyya, R. (2011). "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms." *Software - Practice and Experience 41(1):23–50*

[16] Filho, M. C. S., Oliveira, R. L., Monteiro, C. C., Inacio, P. R. M., & Freire, M. M. (2017). "CloudSim Plus: A cloud computing simulation framework pursuing software engineering principles for improved modularity, extensibility and correctness." *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM). doi:10.23919/inm.2017.7987304*

[17] Hado van Hasselt and Arthur Guez and David Silver (2015). "Deep Reinforcement Learning with Double Q-learning" *arXiv:1509.06461v3 [cs.LG] 8 Dec 2015*

[18] Davide Arcelli (2021). "Understanding and Comparing Approaches for Performance Engineering of Self-adaptive Systems Based on Queuing Networks" *Journal of Ubiquitous Systems and Pervasive Networks (JUSPN), Volume 14, No. 2 (2021), pp.27 -35*

[19] Elarbi Badidi (2020). "QoS-Aware Placement of Tasks on a Fog Cluster in an Edge Computing Environment", *Journal of Ubiquitous Systems and Pervasive Networks (JUSPN), Volume 13, No. 1 (2020), pp. 11-19*

[20] Scott Fujimoto, Herke van Hoof, David Meger (2018). "Addressing Function Approximation Error in Actor-Critic Methods" *arXiv:1802.09477v3 [cs.AI] 22 Oct 2018*