Deep Adversarial Imitation Reinforcement Learning for QoS-aware Cloud Job Scheduling

Yifeng Huang, Long Cheng, *Senior Member, IEEE,* Lianting Xue, Cong Liu, *Member, IEEE,* Yuancheng Li, Jianbin Li, Tomas Ward, *Senior Member, IEEE*

***Abstract*—Although cloud computing is one of the promising technologies for online business services, how to schedule real-time cloud jobs with high QoS (quality of service) is still challenging current techniques. With the advancing of machine learning, Deep Reinforcement Learning (DRL) has demonstrated its outstanding capability in dispatching time-sensitive tasks. However, the reinforced rewards in DRL are typically unavailable until the completion of the scheduling for all the jobs. Considering the fact that the trajectory of jobs in cloud is always long, current DRL-based solutions will meet challenges in finding the trajectories with high rewards and thus would have issues such as that the finally trained scheduling policy is suboptimal. To improve the problem, in this work, we propose a more advanced approach called AIRL, a deep adversarial imitation reinforcement learning framework for scheduling time-sensitive cloud jobs. Specifically, we focus on scheduling user requests in a way to maximize job successful rate along with a significant reduction on job response time. We present the detailed design of our method and our experimental results demonstrate that AIRL can generally outperform the existing cloud job scheduling approaches including the DRL-based method in the presence of different real-time workload and computing resource configurations.**

***Index Terms*—QoS, cloud computing, job scheduling, DRL, adversarial imitation learning, real-time jobs.**

I. Introduction

CLOUD computing is a model that enables users to access computing services over the Internet, including both hardware and software resources, from anywhere at any time. Because of some specific characteristics, such as high-performance, low-maintenance, elasticity and scalability, cloud computing has been considered as one of the encouraging technologies for modern society. Specifically, cloud computing has been widely used for providing online business services that more and more companies have been migrating their applications to the cloud.

One of the fundamental mechanisms in cloud computing is job scheduling, which aims to effectively dispatch computing tasks submitted by end-users to a resource pooling constituting of a lot of heterogeneous virtual machines (VMs) [1]. Since job scheduling affects not only system efficiency (e.g., resource utilization) but also user experience (e.g., QoS), it has been seen as of paramount importance to the cloud ecosystem. In fact, the scheduling problems in cloud computing can be generally categorized into two layers: one is to allocate jobs submitted by users to a set of available VM resources, and the other is to make a VM in a suitable host to create or migrate [2]. We focus on optimizing the former problem in this work, because we are aiming to handle transactional workloads, which occur commonly in today's business world (like e-commerce) that a company has rented a set of VMs and would like to use the resources to their most potential to improve QoS for online services.

The complexity of the job scheduling optimization problem in cloud has shown to be NP-hard in many cases [3]. This means that the problem solving time would be in exponential time from a theoretical perspective. To get an approximate optimal solution in an acceptable time, a large number of job scheduling approaches have been proposed in the past decades, such as customized heuristics [4] and metaheuristics as well as their variations [5], [6]. However, due to the computational overhead in optimization, almost all the approaches are designed specified for handling batch workloads, which are typically operated in the background (e.g., process huge volumes of data [7]) and are rarely time sensitive. In comparison, the scheduling of real-time jobs like the order processing in transactional workloads has been seldom studied.

Although we can use the most conventional methods such as Random and Round-Robin to make real-time decisions for incoming jobs in cloud, the approaches have shown to be inefficient in processing complex transactional workloads [8]. With the advancing of machine learning technology, deep learning (DL) starts to represent an effective solution to tackle complex scheduling problems in different domains [9]. Moreover, among all the emerging DL technologies, Deep Reinforcement Learning (DRL), which can perform optimization withtout any prior knowledge of a system, has demonstrated its outstanding capability in handling scheduling problems in cloud computing in recent two years [10]. Specifically, considering the dynamics of incoming jobs and the complex status of a cloud system, it will be hard to use conventional cloud job scheduling approaches to handle real-time incoming jobs in an effective way. In comparison, a DRL-based intelligent agent can learn effective scheduling strategies from training and on that basis to dispatch cloud jobs in real-time based on

Part of this work was supported by the Fundamental Research Funds for the Central Universities (2021MS017), the National Science Foundation of China (61902222), the Taishan Scholar Youth Program of Shandong Province (tsqn201909109), and Science Foundation Ireland under Grant No. SFI/12/RC/2289P2. *(Corresponding Author: Long Cheng)*

Y. Huang, L. Cheng, L. Xue, Y. Li, and J. Li are with the School of Control and Computer Engineering, North China Electric Power University in Beijing, China. E-mail: lcheng@ncepu.edu.cn

C. Liu is with the School of Computer Science and Technology, Shandong University of Technology, China. E-mail: liucongchina@sdust.edu.cn

T. Ward is with the Insight SFI Research Centre for Data Analytics, School of Computing, Dublin City University, Ireland. E-mail: tomas.ward@dcu.ie

the status of used instances.

To date, some works have tried to apply DRL for cloud job scheduling [11]. Netherless, only quite limited works have considered QoS in their scheduling. Generally, QoS measures the collective effort of service performance and determines the degree of satisfaction from users, e.g., users always expect their requests to be answered immediately, which is critical for cloud economics. Specifically, a high-QoS model can reduce the risk of losing end-users in business and maximize the potrntianal profits for a company [12]. Moreover, the scheduling of real-time incoming cloud jobs can be modeled as a typical Markov Decision Process [13], and the existing DRL-based scheduling approaches have ignored the fact that the training in DRL is actually based on maximizing the expected cumulative reward. Namely, the reinforced rewards will be unavailable until the completion of the scheduling for all the jobs. Since the trajectory of jobs in cloud is always long, the solutions will meet challenges in finding the trajectories with high rewards and thus would have issues such as that the finally trained scheduling policy is suboptimal.

To efficiently handle real-time jobs while keeping high QoS for their execution in cloud, in this work, we propose AIRL, a deep adversarial imitation reinforcement learning approach for QoS-aware cloud job scheduling. Specifically, the imitation learning is a widely used learning strategy in reinforcement learning, and it enables an agent to learn through imitating the expert trajectory. Moreover, to tackle delayed reward issues in DRL, we adopt adversarial imitation learning in our training to cache job trajectories with high rewards as experts and give the DRL agent immediate directions to learn better policy.

In general, the main contributions of this paper can be summarized as follows:

- We propose an advanced QoS-aware job scheduling framework AIRL, which aims to maximize the successful rate and minimize the average response time for time-sensitive jobs in cloud.
- We introduce the optimization model of our scheduling problem and present the details on how we can seamlessly combined the adversarial imitation learning and deep reinforcement learning together for cloud job scheduling. Additionally, we also give the model training strategy we have used in our implementations.
- We compare our approach with six real-time job scheduling algorithms including the DRL-based method, and our experimental results demonstrate that AIRL can generally achieve better performance for the success rate and average response time for job execution in the presence of different real-time workload and VM configurations.

The remainder of this paper is organized as follows. In Section II, we discuss in detail about the related works. We introduce the mathematical model for the studied QoS-aware scheduling problem in Section III and present the details of our AIRL approach in Section IV. We carry out extensive experiments of our method in Section V and conclude this paper in Section VI.

II. Related Works

The optimization of resource utilization and job execution in cloud has been widely studied in the past decades [14], [15]. Generally, the relevant scheduling problems can be categorized into two main types: one is a mapping between resources and jobs which aims to allocate reasonable resources to required jobs, and the other is a VM and host mapping such as creating or migrating VMs over a set of hosts [1], [16]. In this work, we focus on the former problem, i.e., job scheduling. Specifically, cloud job scheduling affects not only the QoS for users (e.g., deadline, makespan, and cost [17], [18]) but also the economic benefits of cloud service providers [19]. An improper scheduling strategy can degrade the overall performance of a cloud system and violate the QoS guarantees [20].

With the growing complexity of cloud jobs and service requirements, getting an optimized plan of job scheduling within a reasonable operational time is still quite chanllenging. Up to now, various mathematical and statistical theories, such as queuing theory [21], [22], control theory [23], [24] and game theory [25], [26], have been applied to job scheduling in cloud. Moreover, a trend of using metaheuristic algorithms as well as their variations is also emerging rapidly (e.g., GA [27], ACO [28], PSO [29] and WOA [30]) and used in solving scheduling problems [31]. In the meantime, to cope with some specified workloads such as DL training jobs, many advanced systems and algorithms have been implemented for cloud enviroments [32], [33], [34]. Although all the approaches have demonstrated their strong capability on cloud job scheduling, they are designed for handling batch jobs. Namely, they are unable to process time-sensitive tasks, due to the significant overhead on solving optimization problems. In comparison, in this work, we focus on proposing a DRL-based method for scheduling real-time jobs in cloud.

In dynamic and uncertain cloud environments, assigning continuously jobs to suitable computing resources in real time is actually very challenging. The main reasons is that cloud systems always do not have any prior knowledge about incoming jobs, such as arrival time and required resources. Conventional real time scheduling strategies, such as FIFO, Round-Robin, etc., can be used for online job scheduling. However, they are not QoS guaranteed and could perform poorly in many conditions (details see our evaluation in Section V). In comparison, a DRL-based approach will be able to learn how to make better decisions in future through the interaction with changing environments during the process of model training.

In fact, without requiring any prior knowledge of a system, DRL has demonstrated its strong capabilty on making decisions for complex problems in many domains [35]. The typical examples include but not limited to network control in IoT and resource management in edge computing [36], [37]. Currently, DRL has also been used in cloud computing, and some works have shown that DRL can effectively improve system energy efficiency, job runtime for jobs, QoS and job success rates [10], [11], etc. However, one possible issue for all the approaches is that they just use DRL in a straightforward way and ignore that fact that the training process is actually

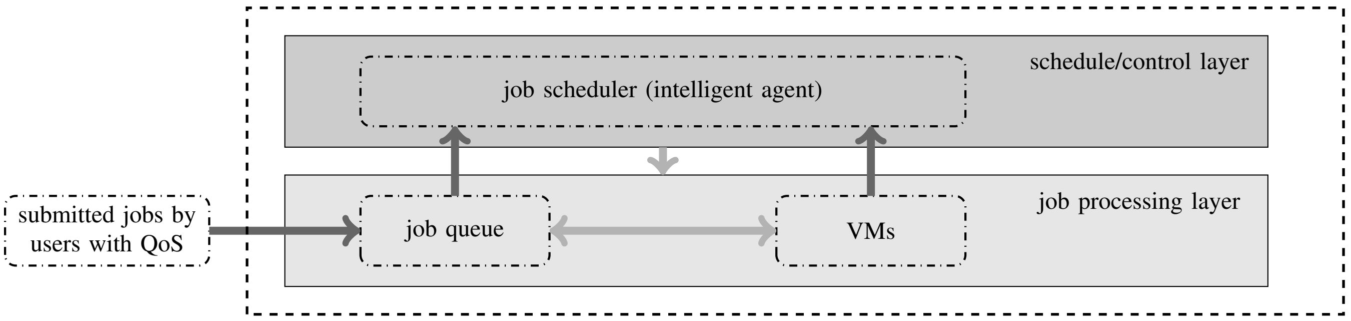

schedule/control layer
job scheduler (intelligent agent)
job processing layer
submitted jobs by users with QoS
job queue
VMs

Fig. 1: The logical view of a cloud job scheduling system [1].

based on maximizing the expected cumulative reward, which are typically unavailable until the completion of the scheduling for all jobs. In contrast to that, we focus on a more advanced design for the reinforced rewards in DRL on the basis of adversarial imitation learning [38], which is an important learning strategy of reinforcement learning. Different from the Q-learning-based approach or policy-gradient-based approach, it enable that an agent to learn from imitating an expert agent or expert trajectory [39]. Moreover, our experimental results demonstrate that our method can indeed further improve the scheduling performance, compared to a DRL-based scheduling approach.

III. QoS-aware Scheduling

In this section, we briefly introduce the general system architecture of job scheduling in cloud as well as the mathematical model for QoS-aware scheduling that we have used in this work.

A. System Architecture

For a cloud system, various components such as job and resource monitors are normally used to support job scheduling. For the purpose of this work, we mainly consider users, job queues, job scheduler and VMs in the system. Specifically, similar to the work [1], we abstract our job scheduling system following a two layer processing architecture as illustrated in Fig. 1. Since we focus on real-time workloads in this paper, without loss of generality, we assume that each job is independent of each other. Moreover, with QoS in mind, the process of the job scheduling in our system can be summarized into the following four steps. (1) For each income job (i.e., user request), based on the detailed information of the job (e.g., job type and QoS requirement) and the status of all VMs, an optimized job-VM mapping will be generated by the intelligent agent at the schedule/control layer. (2) Following the mapping, the scheduler will forward the job to the allocated VM at the job processing layer. (3) The job will be executed if the computing resources of the assigned VM are available, otherwise the job will wait in a job queue on the VM. (4) After the execution, the returned results will be sent back to the responsible user.

TABLE I: Table of notations

Notation	Meaning
J_{ID}	the id of a job
J_{AT}	the arrival time point of a job
J_T	job type (I/O or computing intensive)
J_L	job instruction length (the number of instructions)
J_Q	QoS requirement (expected completion time point)
J_{RT}	response time point of a job
J_{ET}	execution time of a job
J_{WT}	waiting time of a job
V_{ID}	the id of a VM
V_T	VM type (I/O or computing intensive)
V_P	VM processing speed (instructions per time unit)
V_{IT}	the idle time point of a VM (i.e., when the VM is free)

B. Scheduling Model

For cloud jobs, QoS may refer to many factors, such as levels of performance, reliability, availability, etc. Similar to some recent works [8], [11], we mainly focus on the response time in this paper, which is considered as one of the most critical indicators for user requests in real-time scenarios. Specifically, for a submitted job, if the response time of the job is smaller than the user's expected time, we define that the execution of the job is successful. Otherwise, we say that the job fails on execution. On that basis, we give the mathematical definitions of cloud jobs and resources as well as the execution mechanism of jobs in our scheduling as follows. The meanings of the used notations are given in Table I.

1) Workloads and Cloud Resources: We focus on how to use DRL and adversarial imitation learning techniques to optimize job scheduling in cloud. Based on the characteristics of user requests, for the simplicity of our presentation, we categorize the workloads in our study into two main types: I/O intensive jobs and computing intensive jobs. Moreover, we define each job as $\mathbb{J} = \{J_{ID}, J_{AT}, J_T, J_L, J_Q\}$, where J_{ID} is the job id, J_{AT} is the job arrival time point, J_T is the job type, J_L is the job length and J_Q is the QoS requirement of the job. For computing resources, similar as the workloads, we consider two types of virtual machines in this paper, i.e., I/O intensive and computing intensive VMs. For a set of VMs, each VM can be represented as $\mathbb{V} = \{V_{ID}, V_T, V_P\}$, where V_{ID} is the VM id, V_T is the type and V_P is the processing speed of the VM.

2) Execution Mechanism: To capture the core mechanism of job scheduling in cloud, we assume that each VM can only handle one job at any time point. Since a job will have to wait for free resources if the assigned VM is busy, then the response time of the job will be composed of two parts: waiting time and execution time. In this case, for a job $\mathbb{J}_i$ arrives at time J_{ATi} with the QoS requirement J_{Q_i}, if it is scheduled to be executed on the j-th VM $\mathbb{V}_j$, then its response time can be formulated as:

$$J_{RTi} = J_{ETi} + J_{WTi} \tag{1}$$

where J_{RTi} is the response time, J_{ETi} is the execution time, and J_{WTi} is the waiting time for the job $\mathbb{J}_i$. Specifically, the execution time is the runtime of $\mathbb{V}_j$ to process $\mathbb{J}_i$. It is straightforward that the execution time of a job will be large if the type of the job does not match the type of the allocated VM. To model this case, we define J_{ETi} as:

$$J_{ETi} = \frac{(f^i(J_{Ti} == V_{Tj}) + 1)J_{Lj}}{2V_{Pj}} \tag{2}$$

Here, the f^i is a function, the value of which is set to 1 if the types of the job and assigned VM are the same, otherwise the value will be 0. With this design, we can make sure that when the type of a VM and a job does not match, the VM will take more time to execute the job. Namely, we use this as a penalty for assigning a job to an unmatched VM. Moreover, if the idle time point of $\mathbb{V}_j$ is V_{ITj}, then the waiting time J_{WTi} can be formulated as[1]:

$$J_{WTi} = max(0, V_{ITj} - J_{ATi}) \tag{3}$$

For the case that the job $\mathbb{J}_i$ is indeed assigned to $\mathbb{V}_j$ for the processing, the idle time point of $\mathbb{V}_j$ for the newly coming jobs will be updated as:

$$V_{ITj} = max(0, V_{ITj} - J_{ATi}) + J_{ATi} + J_{ETi} \tag{4}$$

With the above four equations (or the status of the whole scheduling system and job properties), we can then calculate the response time for an incoming job at any time point. Moreover, we can further know whether the execution of the job is to be successful or not for a specified scheduling plan, by comparing its response time with its expected time. From this basis, to achieve high QoS in job scheduling, our main target is to maximize the successful rate of execution for all jobs, each of which could come at any time point. To this end, we give the design of our AIRL in the following section.

IV. The Proposed Approach – AIRL

In this section, we first give the preliminaries and the definition of the Markov Decision Process (MDP [13]) environment for cloud job scheduling. Then, we introduce the detailed design of our AIRL framework as well as the training strategy for our model.

[1]Note that the idle time point represents the point when the VM can execute the assigned job. Namely, the VM may be busy at the current time, and the job will have to wait.

A. Preliminaries

MDP for Job Scheduling. The Markov Decision Process for cloud job scheduling in our system can be modeled as $M = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where $\mathcal{S} = \{s_i\}$ is the state space which contains the working state of all virtual machines and the information of the current job, and $\mathcal{A} = \{a_i\}$ denotes the action space, which consists of the action of the intelligent agent at each step, i.e., assigning the current job to which VM. Moreover, $\mathcal{P} = p(s_{t+1}|s_t, a_t)$ is the transition dynamics which determines the transition probability of the next state s_{t+1} at the current state s_t with action a_t, and $\mathcal{R}$ is the reward function that gives a reward when reaching a new state and γ is the discount factor.

Obviously, in our model, the state transition matrix is unknown to the agent. Moreover, for our QoS-aware scheduling problem, QoS is used to represent the reward. Then, we can use the trajectory $(s_0, a_0, r_0; s_1, a_1, r_1; \cdots; s_n, a_n, r_n)$ to denote the whole scheduling procedure. Additionally, the update of each state can be formulated as $p(s_{t+1}) = \sum_{a_t} p(s_{t+1}|s_t, a_t)p(a_t|s_t)$, in which $a_t \in \mathcal{A}$, $p(a_t|s_t) = \pi(s_t, a_t)$, and π is the policy provided by the agent. In the meantime, the final accumulated reward can be written as $\mathscr{R} = \sum_{t=0}^{n} r_t \gamma^t$.

Deep Q-Learning. The implementation of our scheduling is based on Deep Q-Learning, in which the Q-Learning [40] is an off-policy reinforcement learning algorithm aiming to learn a value function $Q : \mathcal{S} \times \mathcal{A} \longrightarrow \mathbb{R}$. The value represents the long-term accumulated reward for a state and its corresponding action. However, there are some limitations for Q-Learning on handling complex jobs with a large state space or action space. With the growing popularity of deep learning techniques, deep Q-learning has been proposed to remedy the problem. Generally, in a Deep Q-learning Network (DQN), a deep neural network (DNN) is used to represent the value function which can be written as $Q_\phi(s_t, a_t)$ and ϕ is the parameters of the DNN.

In DQN, the value function $Q_\phi(s_t, a_t)$ is updated as a regression problem by using the target value $r_t + \gamma max_{\hat{a} \in \mathcal{A}} Q_{\phi^{target}}(s_{t+1}, \hat{a})$, where r_t is the reward at t-th step, γ is the discount factor and $Q_{\phi^{target}}(s_t, a_t)$ is a target value function. These two value functions are trained alternately that $Q_{\phi^{target}}(s_t, a_t)$ is fixed at first and $Q_\phi(s_t, a_t)$ is updated by a regression loss. After a fixed number of training steps, the ϕ^{target} is updated by using ϕ. Typically, the regression loss for $Q_\phi(s_t, a_t)$ is minimized over the mini-batch of the past trajectory (s_t, a_t, r_t, s_{t+1}). The most common loss used to train the DQN is the Mean Squared Error (MSE) loss, which can be formulated as $min_\phi \sum_{i=1}^{|B|}(r_{t_i} + \gamma max_{\hat{a} \in \mathcal{A}} Q_{\phi^{target}}(s_{t_i+1}, \hat{a}) - Q_\phi(s_{t_i}, a_{t_i}))^2$, where B is the training mini-batch and ϕ^{target} is fixed when calculating the MSE loss.

B. Environment Design for DRL

Following the above introduction of MDP and deep Q-learning, we give the environment design for our DRL-based scheduling as follows.

State Space. The state space in our scheduling system is defined as $S = (\mathbb{J}_c, \mathbb{VS})$, where $\mathbb{J}_c$ is the state of the current job to be scheduled. $\mathbb{VS}$ is the state of all VMs $\mathbb{VS} = \{\mathbb{V}_1, \cdots \mathbb{V}_k\}$, where $\mathbb{V}_i$ is the i-th virtual machine, and k is the number of virtual machines, and the MDP will be terminated when all jobs are executed.

Action Space. Our action space only contains one type of action assigning the current job to a virtual machine. In this case, the length of our action space is equal to the number of virtual machines, i.e., $|\mathcal{A}| = k$. Here, we define the action space as $\mathcal{A} = \{a_i, i = 1 \cdots v\}$, where a_i represents that the current job is assigned to the i-th virtual machine.

Transition Dynamics. Suppose that the current state is $s_t = (\mathbb{J}_t, \mathbb{VS})$, then the state of virtual machines will be updated when allocating the t-th job. After that, we can start the scheduling for the $(t+1)$-th job and get the next state of the system. If there are more than one job at the specified time period, our agent will handle them in a FIFO manner.

Reward Function. To achieve high QoS for job scheduling, our main target is to maximize the successful rate of job execution in general. In the meantime, for each job, the smaller the response time is, the better the service will be. It should be noted, in many scenarios, a user will terminate a job if the response time does not meet her/his requirement. Therefore, in our scheduling, we expect that the most possible jobs can be executed in a way to meet deadline requirements from users, and the DRL agent will not receive any reward for a job if the deadline of the job is exceeded. On that basis, we also expect that each job with less response time as possible to further improve the QoS. Therefore, we can make sure that, the less response time for a job execution, the higher reward the DRL agent will receive. From that basis, we define the reward for a job $\mathbb{J}_i$ as:

$$r = \begin{cases} \dfrac{J_{L_i}}{J_{RT\,i} V_{Pj}}, & J_{RT\,i} <= J_Q \\ 0, \quad J_{RT\,i} > J_Q \end{cases} \tag{5}$$

where J_{L_i} is the instruction length, $J_{ET\,i}$ is the execution time, and J_Q is the QoS requirement of the job. With such a reward function design, the DRL agent is encouraged to take actions to improve the QoS (i.e., the success rate and the response time). This is also the reason why we say that our job scheduling is QoS-aware.

C. AIRL for QoS-aware Job Scheduling

Different from existing works on simply using DQN, we have adopted the more advanced Dueling-DQN [41] for job scheduling. Moreover, to learn scheduling policy in a more effective way during the training of our DRL model, we have seamlessly integrated the adversarial imitation learning in our DRL based approach. The general architecture of our AIRL framework is illustrated in Fig. 2 with the details of the used Dueling-DQN being presented in Fig. 3. There, the Dueling-DQN takes the virtual machines and the arriving job as the input state, and the state will be updated when the agent makes a decision on the scheduling. In the meantime, the agent will receive a reward upon the decision. Followed by that, the expert policy will also make a decision for the current state, and the discriminator will calculate an imitation loss to direct the Dueling-DQN. This process will be repeated during the training, and the agent will be able to learn learn how to make better decisions on the scheduling for incoming jobs in future. In the following, we present the detailed design of the Dueling-DQN and adversarial imitation learning for our QoS-aware job scheduling problem.

1) Dueling-DQN for Job Scheduling: Generally, cloud job scheduling is more challenging than many other problems when using reinforcement learning techniques. The main reason is that cloud job are always complex. Since Dueling-DQN has shown to be more powerful on evaluating complex states than DQN [41], we have used the Dueling-DQN as our scheduling agent's model. In a normal DQN, there are only one state-action value function $Q(s_t, a_t)$ used to estimate the return of current state-action pair. In comparison, the state-action value function in Dueling-DQN is divided into two parts, one is the state value function $V(s_t)$, and the other one is the action advantage function $A(s_t, a_t)$. The relationship between these three functions can be formulated as:

$$Q(s_t, a_t) = V(s_t) + A(s_t, a_t) \tag{6}$$

Here, the state value function is used to estimate the value of the current state, and the action advantage function is to estimate the advantages of each action. With these two different functions, the agent can learn which state is valuable and which action is valuable under a state. For our job scheduling problem, when the agent goes into a bad state, the choice of action will have only a little effect on the final return. However, similar to a DRL-based approach, the Dueling-DQN could still meet issues in its learning process, such as that it is unable to alleviate the delayed reward for long trajectory jobs. To improve this problem, we further employ the adversarial imitation learning in our scheduling.

2) Adversarial Imitation Learning for Job Scheduling: Inspired by Generative Adversarial Network (GAN) [42], we introduce the adversarial imitation learning [43] [38] [44] for cloud job scheduling. As illustrated in Fig. 2, the main idea is that we use a discriminating signal (discriminator) to give an immediate direction to the agent. Specifically, we use the adversarial imitation loss as a surrogate reward, which can be densely obtained during behavior distribution fitting. In this way, we can help the agent to make better decisions in the presence of long trajectories. In detail, if there is an expert policy π_E, then the discriminating signal delivered to the agent will teach the agent how its action like the expert policy. In this process, the discriminating signal is formulated as:

$$\underset{\phi}{min}\underset{\theta}{max}(E_{\pi_\phi}(logD_\theta(s_t, a_t)) + E_{\pi_E}log(1 - D_\theta(s_t, a_t))) \tag{7}$$

where D is the discriminating signal, π_ϕ is the policy of our agent, a dueling-DQN, π_E is the expert policy that our agent tend to imitate, ϕ is the parameters of the dueling-DQN and θ is the parameters of the discriminator D.

In our model, we have used the multilayer perceptron (MLP) as the expert policy E_{π_E}, the input of which is state and the output is a probability distribution over all actions. The policy E_{π_E} is pre-trained by a directing policy, and the probability distribution represents the expert policy's decision.

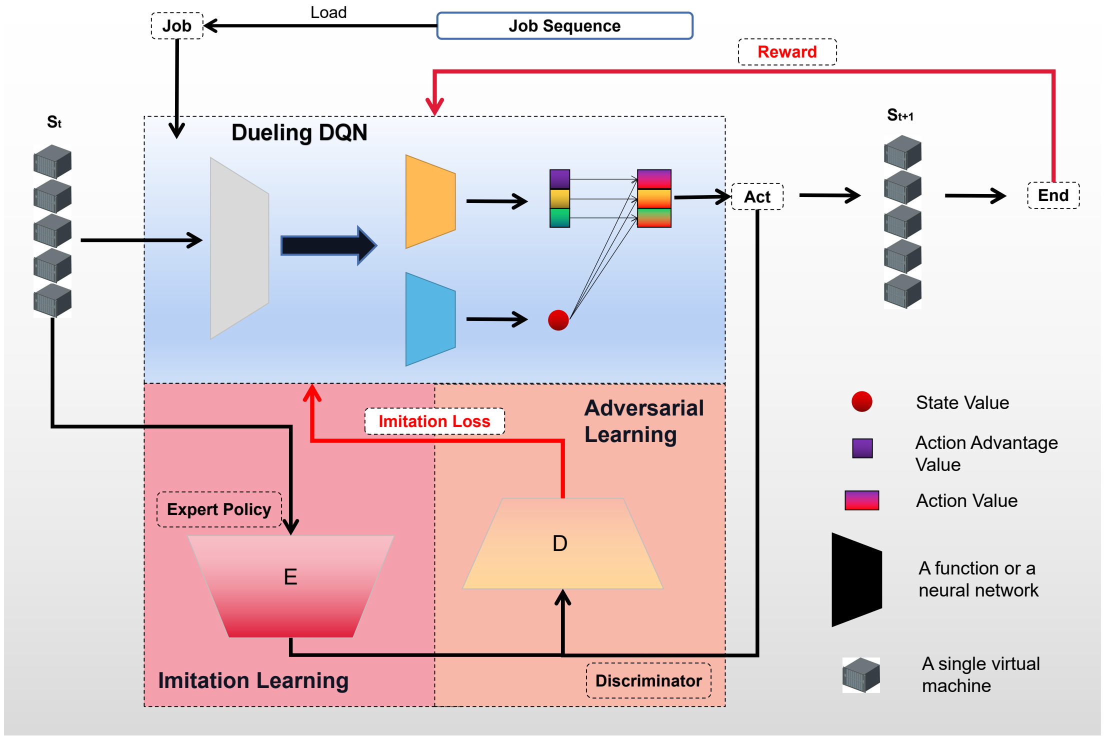


Fig. 2: The general architecture of the proposed AIRL framework for cloud job scheduling.

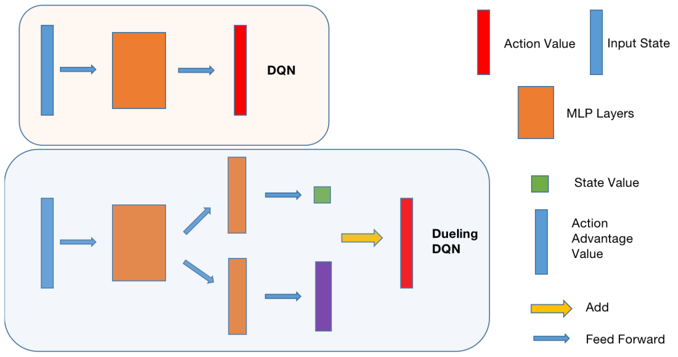


Fig. 3: The details of the Dueling-DQN used in AIRL framework.

Mathematically, the discriminating signal is a function which can provide immediate directions to the agent. In GAN, the discriminating function is a neural network. In our approach, we let the imitation loss follow the Maximum Causal Entropy theory [45] and choose the Jensen-Shannon divergence as the discriminating function, which is defined as:

$$D_{JS} = JS(\mathcal{A}_{\pi_E} || \mathcal{A}_{\pi_\phi}) \tag{8}$$

where $\mathcal{A}_{\pi_E}$ is the action probability distribution of the expert policy and $\mathcal{A}_{\pi_\phi}$ is the Q value for all actions calculated by the agent. D_{JS} is the Jensen-Shannon divergence and it can be calculated as:

$$\begin{aligned} JS(\mathcal{A}_{\pi_E} || \mathcal{A}_{\pi_\phi}) = & \frac{1}{2} KL(\mathcal{A}_{\pi_E} || \frac{\mathcal{A}_{\pi_E} + \mathcal{A}_{\pi_\phi}}{2}) \\ & + \frac{1}{2} KL(\mathcal{A}_{\pi_\phi} || \frac{\mathcal{A}_{\pi_E} + \mathcal{A}_{\pi_\phi}}{2}) \end{aligned} \tag{9}$$

where KL is the Kullback-Leibler Divergence. In this case, the optimization of imitation loss can be written as:

$$\min_{\phi} (E_{\pi_E} E_{\pi_\phi} (D_{JS}(\mathcal{A}_{\pi_E} || \mathcal{A}_{\pi_\phi})) \tag{10}$$

To further improve the performance of the adversarial learning, we let the expert policy and the DQN train jointly in an adversarial way after some iterations. Therefore, the imitation loss can be represented by:

$$\begin{cases} \min\limits_{\phi} (E_{\pi_E} E_{\pi_\phi} (D_{JS}(\mathcal{A}_{\pi_E} || \mathcal{A}_{\pi_\phi})), iter < T_a \\ \min\limits_{\phi} \max\limits_{\beta} (E_{\pi_{E_\beta}} E_{\pi_\phi} (D_{JS}(\mathcal{A}_{\pi_E} || \mathcal{A}_{\pi_\phi})), iter >= T_a \end{cases} \tag{11}$$

where β is the parameters of the expert policy, $iter$ is the current iteration and T_a is the iteration that starts the joint train between E_{π_E} and E_{π_ϕ}.

3) Model Training Strategy: The training procedure of our model can be divided into two parts. The first part is the pre-train of the expert policy π_{E_β}, and the second part is the training of Dueling-DQN π_ϕ as well as the joint adversarial training between π_{E_β} and π_ϕ.

As presented in in Algorithm 1, for the pre-train of π_{E_β}, we first sample a set of job trajectories by using the directing policy. Then, the trajectories are used to train the expert policy π_{E_β}. For the second part, we first define the loss to train our model. Our loss function contains two parts: (1) the first part is the imitation loss, which helps the Dueling-DQN imitate expert policy's action; and (2) the second part is the MSE loss that used to train the Dueling-DQN. In this process, the imitation loss is defined as:

$$\mathcal{L}_{IMI} = D_{JS}(\mathcal{A}_{\pi_\phi} || \mathcal{A}_{\pi_E}) \tag{12}$$

Algorithm 1 Pre-train the Expert Policy

```
Require: Directing Policy: $\mathcal{P}$, Expert Policy: $\pi_{E_\beta}$, Trajectory Sample Function: $\mathcal{F}$, Iterations: $\mathcal{T}$.
1: for $\mathcal{T}$ Iterations do
2:     Sample Training Trajectory: $B_p = \mathcal{F}(\mathcal{P})$
3:     for $(s_t, a_t, rt, s_{t+1})$ in $B_p$ do
4:         $a_E = \pi_{E_\beta}(S_t)$
5:         $\mathcal{L}_p = KL(a_E, a_t)$
6:         Update the parameters $\beta$ with $\mathcal{L}_p$
7:     end for
8: end for
```

Algorithm 2 Training Strategy

```
Require: Expert Policy: $\pi_{E_\beta}$, Dueling-DQN Policy: $\pi_\phi$, Target Dueling-DQN Policy: $\pi_{\phi^{target}}$, Reply Buffer: $\mathcal{R}$, Greedy Parameter: $\eta$, Job Queue: $\mathcal{J}$, Virtual Machine set: $\mathcal{VM}$, Epochs: $E$, Target Update Interval $I_t$, Start Learn Iteration $I_s$, Learning frequency $f$, Adversarial Training Iteration $I_A$, trade off weight $\lambda$.
1: for $E$ Epochs do
2:     for i in length($\mathcal{J}$) do
3:         Load the i-th job $j_i$
4:         With probability $\eta$ randomly select a action
5:         Otherwise $a_i = argmax_a \pi_\phi(concate(j_i, \mathcal{VM}), a)$
6:         Update $\mathcal{VM}$ and calculate the reward $r_i$
7:         Append $(s_i, a_i, ri, s_{i+1})$ to $\mathcal{B}$
8:         if $i >= I_s$ and $i \equiv f \mod 0$ then
9:             Sample a mini-batch $\mathcal{B}$ from $\mathcal{R}$
10:            for $(s_t, a_t, r_t, s_{t+1}) in \mathcal{B}$ do
11:                Calculate the MSE Loss $\mathcal{L}_{MSE}$
12:            end for
13:            Sample a mini-batch $\mathcal{B}$ from $\mathcal{R}$
14:            for $(s_t, a_t, r_t, s_{t+1}) in \mathcal{B}$ do
15:                Calculate the imitation learning loss $\mathcal{L}_{IMI}$
16:            end for
17:            Update $\phi$ with $\mathcal{L}_{MSE} + \lambda\mathcal{L}_{IMI}$
18:            if $i >= I_A$ then
19:                Update $\beta$ with $-\lambda\mathcal{L}_{IMI}$
20:            end if
21:            if $i \equiv I_t \mod 0$ then
22:                $\phi^{target} \leftarrow \phi$
23:            end if
24:        end if
25:    end for
26: end for
```

In the meantime, the MSE loss can be written as:

$$\frac{1}{2}(r_t + \gamma max_{\hat{a} \in \mathcal{A}} Q_{\phi^{target}}(s_{t+1}, \hat{a}) - Q_\phi(s_t, a_t))^2 \qquad (13)$$

where (s_t, a_t, r_t, s_{t+1}) is the trajectory sampled from the reply buffer and γ is the discount factor. With the loss functions, we can then train our model following the steps as described in Algorithm 2. For the whole training process, we first pre-train the expert policy through Algorithm 1. After getting a pretrained expert policy, we then train the expert policy and the dueling-DQN policy in an adversarial manner according to Algorithm 2.

V. Evaluation

In this section, we compare the performance of our approach with existing scheduling methods using different job workload and VM configurations.

A. Experiment Setup

In our experiment, we have set the number of VMs to 10 and the number of jobs to 8000. To simulate the real transactional workloads, similar to many other works [46], the arrival time of jobs is generated by a Poisson Distribution and the mean arrival rate is selected from {10, 15, 20, 25, 30}. Moreover, the length of each job is generated by a Normal Distribution, the mean value and variance of job's length is set to 300 and 20 respectively. As there are two types of jobs in this study, we set the proportion of the I/O intensive jobs to a value selected form {0.1, 0.3, 0.5, 0.7, 0.9}. Similarly, the value of the proportion for the High-I/O VMs is selected from {0.1, 0.3, 0.5, 0.7, 0.9}.

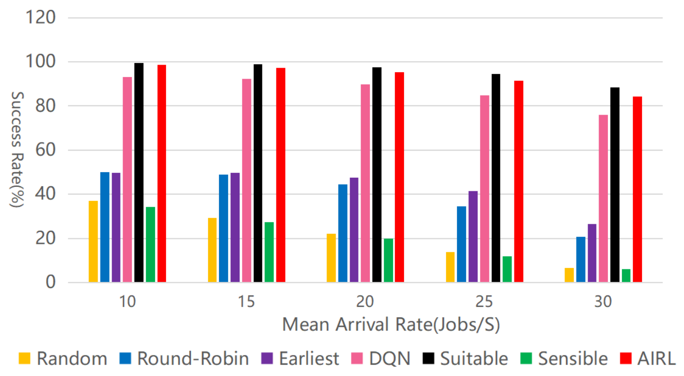
In our evaluation, we have chosen 6 methods as the baselines to our approach, including Random scheduling, Round Robin algorithm, Earliest algorithm, Suitable algorithm, Sensible algorithm [8] and DQN-based approach [11]. Among them, the Suitable algorithm will first try to select a set of VMs, the type of which matches the type of a given job, and then choose the VM with the earliest idle time point to execute the job. Moreover, we have used three metrics to evaluate the performance of our approach. The first metric is the **S**uccess **R**ate (**SR**), which is used to measure how many job are processed successfully, and a job is processed successfully if and only if its response time is lower than the pre-defined QoS requirement. The second metric is the **A**verage **R**esponse **T**ime (**ART**), which is used to measure the average time on processing each job. The third metric is the **U**tilization **R**ate (**UR**), which is used to measure the utilization rate of each VM. It should be noted that the first two metrics are related to our optimization target on QoS. We have used UR because we want to check whether the cost to achieve a high QoS is high or not for our method.

For the detailed parameters used in the evalution, we use two simple MLP layers as the Dueling-DQN and the expert policy in AIRL. The dimension of hidden layer is set to 20 for both DQN and expert policy. The learning rate for Dueling-DQN is set to 0.005. The trade off parameter σ is set to 0.05. The greedy parameter is set to 0.8, the start learn iteration is set to 500, and the target update interval is set to 1200 iterations. The reply buffer size is set to 500 and the mini-batch size is set to 32. The trade off weight parameter λ is set to 0.01. We use ADAM optimizer [47] for training and implement our approach on PyTorch[2]. The source code we have used in this section is available at https://github.com/huang1997214/SAIRL.

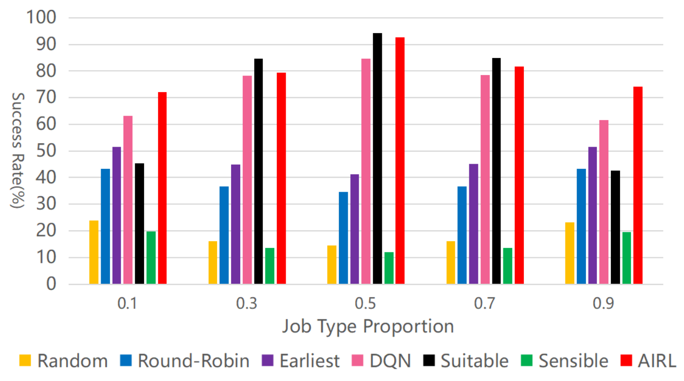
B. Comparison of Scheduling Performance

Varying mean arrival rates of jobs. We first evaluate the performance of our approach under the condition that jobs arrive with different rates. In this test, the proportions of job type and the VM type are both set to 0.5, and the mean arrival rate of jobs is varied from 10 to 30. The experimental results about SR, ART and UR are presented in Fig. 4. From the SR reported in Fig. 4(a), we can see that our AIRL can

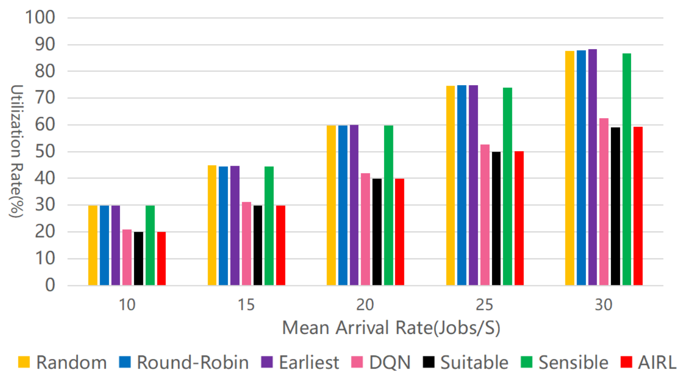
[2]https://pytorch.org/



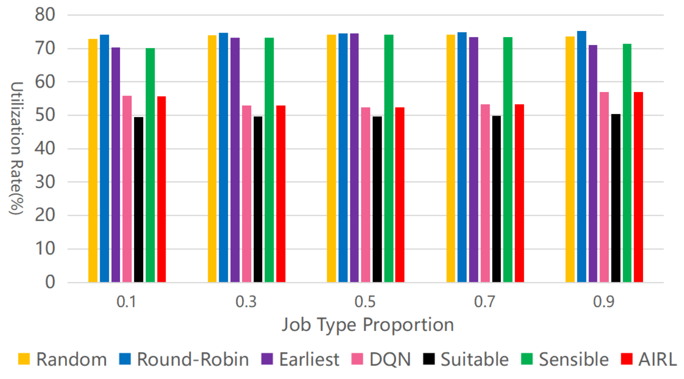
(a) Success Rate.



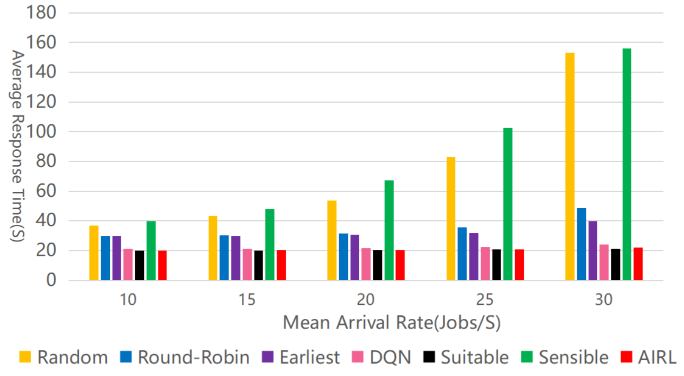
(a) Success Rate.



(b) Utilization Rate.

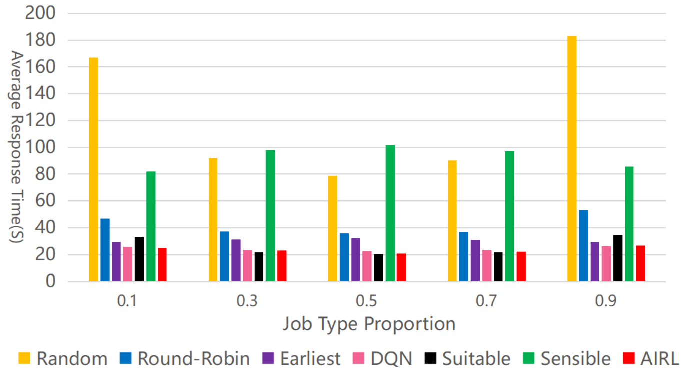


(b) Utilization Rate.



(c) Average Response Time.

Fig. 4: Performance comparison by varying the mean arrival rate of jobs.

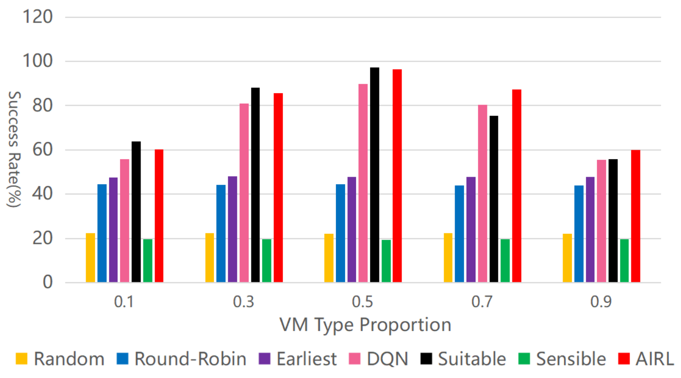


(c) Average Response Time.

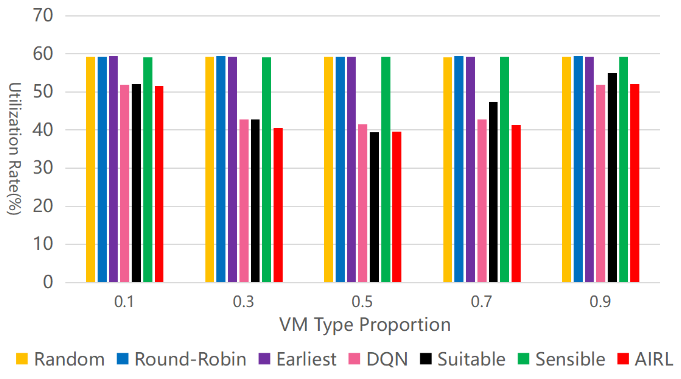
Fig. 5: Performance comparison by varying the proportion of job types.

significantly outperform the conventional approaches such as Round-Robin and Earliest. Moreover, we can achieve around $5\%-10\%$ higher SR, compared to the DQN method. Although the Suitable method performs the best, the SR of AIRL is still very close to the policy. This kind of characteristics is also applied to the results presented in Fig. 4(b) and Fig. 4(c) respectively. Specifically, AIRL and Suitable perform nearly the same on UR and ART, and both are much better than other scheduling methods. From all the results above, we can find that our AIRL can generally outperform the existing approaches on scheduling real-time jobs, in the case that the distribution of jobs and VMs is balanced.

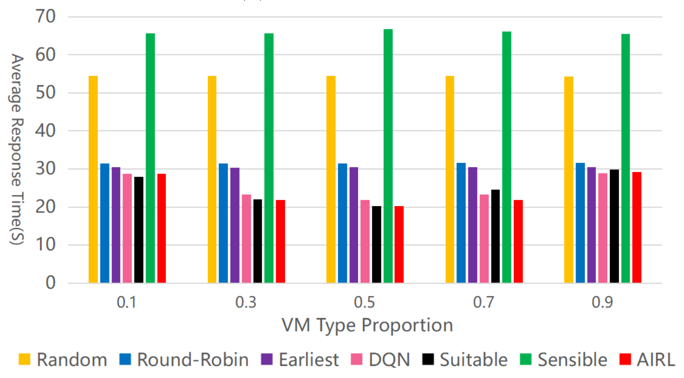
Varying job type proportions. To evaluate the performance of our approach in the presence of different proportions of job types, we vary the job type proportion from 0.1 to 0.9, and the value 0.1 means that there are 10% of all jobs are I/O sensitive while 0.9 represents that 90% of the jobs are I/O sensitive. In this process, we fix the job's mean arrival rate to 25 and the VM type proportion to 0.5. As shown in Fig. 5(a), the SR of our AIRL performs is generally higher than other approaches including DQN. Specifically, it performs much better than all other algorithms when the job type proportion is 0.1 and 0.9. When the proportion is set to 0.3, 0.5 and 0.7, our approach performs slightly worse than the Suitable policy. However, the performance difference is actually not obvious. Moreover, if we consider the results in general, we can see that the performance of our approach is actually much more robust than Suitable, which can be also observed from



(a) Success Rate.



(b) Utilization Rate.



(c) Average Response Time.

Fig. 6: Performance comparison by varying VM type proportions.

the results on ART as reported in Fig. 5(c). From Fig. 5(b), we can observe that the UR of our method is higher than Suitable. Regardless, our main focus is to improve the QoS of job scheduling (i.e., SR and ART) in this work, and our UR is still much lower than the conventional approaches. To conclude, when the job type distribution is unbalanced (e.g., for the cases with 0.1 and 0.9), our AIRL can outperform other baselines significantly. Moreover, when the distributed is balanced, Suitable will be a better solution, compared to AIRL. However, it should be noted that their performance difference is quite small. Moreover, a balanced distribution is actually rare in a highly dynamic cloud environment. In such scenarios, our AIRL will be the best choice for a scheduling system, as it can always achieve a high QoS for job scheduling.

Varying VM type proportions. Similar to the above experiment, we also evaluate our method by varying the VM type proportions from 0.1 to 0.9. Here, the value 0.1 means that only 10% VMs are High-I/O VMs while others are High-Computing VMs. As shown in Fig. 6, our AIRL can still generally outperform other methods including DQN on SR, UR and ART. Specifically, when the VM type proportion is set to 0.7 and 0.9, our method can achieve the highest SR. In addition to that, AIRL performs the best on the average response time when VM type proportion is set to 0.3, 0.5 and 0.7. Through all these results, we can see that our AIRL is robust to the VM type proportions. It means that AIRL can always provide a good scheduling scheme for incoming jobs, no matter what the configuration of underlying VMs is.

VI. Conclusion

Although the emerging DRL has been used to optimize job scheduling in cloud, the final trained model could be still suboptimal, due to the fact that cloud job trajectories are always long and a DRL agent can only get the accumulated reward when all the jobs are executed. To further improve the learning effectiveness of current DRL-based scheduling, we propose a more advanced framework for cloud job scheduling called AIRL. Specifically, AIRL can cache job trajectories with high rewards as experts and give the DRL agent an immediate direction signal during the training process. We have presented the detailed design of AIRL and also compared its performance on QoS with many other scheduling algorithms. Our experimental results have demonstrated that AIRL can generally outperform existing approaches in the presence of different real-time workload and VM configurations.

For the future work, we will mainly focus on extending the proposed scheduling model with some more advanced features for cloud jobs. For example, we will try to use our approach to handle cloud workflows [48], the jobs in which are not independent from each other and each job could have its own resource requirements. In the meantime, we also plan to apply our method in more complex cloud computing environments such as that the VM resources are dynamically provisioned.

References

[1] X. Chen, L. Cheng, C. Liu, Q. Liu, J. Liu, Y. Mao, and J. Murphy, "A WOA-based optimization approach for task scheduling in cloud computing systems," *IEEE Systems Journal*, vol. 14, no. 3, pp. 3117–3128, 2020.
[2] Z.-H. Zhan, X.-F. Liu, Y.-J. Gong, J. Zhang, H. S.-H. Chung, and Y. Li, "Cloud computing resource scheduling and a survey of its evolutionary approaches," *ACM Computing Surveys*, vol. 47, no. 4, p. 63, 2015.
[3] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation computer systems*, vol. 25, no. 6, pp. 599–616, 2009.
[4] T. D. Braun, H. J. Siegel, N. Beck, L. L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen *et al.*, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," *Journal of Parallel and Distributed computing*, vol. 61, no. 6, pp. 810–837, 2001.
[5] K. Hussain, M. N. M. Salleh, S. Cheng, and Y. Shi, "Metaheuristic research: a comprehensive survey," *Artificial Intelligence Review*, vol. 52, no. 4, pp. 2191–2233, 2019.
[6] P. Singh, M. Dutta, and N. Aggarwal, "A review of task scheduling based on meta-heuristics approach in cloud computing," *Knowledge and Information Systems*, vol. 52, no. 1, pp. 1–51, 2017.

[7] L. Cheng, B. F. van Dongen, and W. M. P. van der Aalst, "Scalable discovery of hybrid process models in a cloud computing environment," *IEEE Transactions on Services Computing*, vol. 13, no. 2, pp. 368–380, 2020.

[8] L. Wang and E. Gelenbe, "Adaptive dispatching of tasks in the cloud," *IEEE Transactions on Cloud Computing*, vol. 6, no. 1, pp. 33–45, 2018.

[9] Q. Liu, L. Cheng, A. L. Jia, and C. Liu, "Deep reinforcement learning for communication flow control in wireless mesh networks," *IEEE Network*, vol. 35, no. 2, pp. 112–119, 2021.

[10] M. Cheng, J. Li, and S. Nazarian, "DRL-cloud: Deep reinforcement learning-based resource provisioning and task scheduling for cloud service providers," in *Proceeding of 23rd Asia and South Pacific Design Automation Conference*, 2018, pp. 129–134.

[11] Y. Wei, L. Pan, S. Liu, L. Wu, and X. Meng, "DRL-scheduling: An intelligent QoS-aware job scheduling framework for applications in clouds," *IEEE Access*, vol. 6, pp. 55 112–55 125, 2018.

[12] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica *et al.*, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[13] M. van Otterlo, "Solving relational and first-order logical markov decision processes: A survey," in *Reinforcement Learning*. Springer, 2012, pp. 253–292.

[14] A. Arunarani, D. Manjula, and V. Sugumaran, "Task scheduling techniques in cloud computing: A literature survey," *Future Generation Computer Systems*, vol. 91, pp. 407–415, 2019.

[15] M. Kumar, S. C. Sharma, A. Goel, and S. P. Singh, "A comprehensive survey for scheduling techniques in cloud computing," *Journal of Network and Computer Applications*, vol. 143, pp. 1–33, 2019.

[16] S. Kayalvili and M. Selvam, "Hybrid SFLA-GA algorithm for an optimal resource allocation in cloud," *Cluster Computing*, vol. 22, no. 2, pp. 3165–3173, 2019.

[17] J.-W. Lin, C.-H. Chen, and C.-Y. Lin, "Integrating qos awareness with virtualization in cloud computing systems for delay-sensitive applications," *Future Generation Computer Systems*, vol. 37, pp. 478–487, 2014.

[18] X. Zhu, L. T. Yang, H. Chen, J. Wang, S. Yin, and X. Liu, "Real-time tasks oriented energy-aware scheduling in virtualized clouds," *IEEE Transactions on Cloud Computing*, vol. 2, no. 2, pp. 168–180, 2014.

[19] P. Zhang and M. Zhou, "Dynamic cloud task scheduling based on a two-stage strategy," *IEEE Transactions on Automation Science and Engineering*, vol. 15, no. 2, pp. 772–783, 2017.

[20] C. Delimitrou and C. Kozyrakis, "QoS-aware scheduling in heterogeneous datacenters with paragon," *ACM Transactions on Computer Systems*, vol. 31, no. 4, pp. 1–34, 2013.

[21] H. Khazaei, J. Misic, and V. B. Misic, "Performance analysis of cloud computing centers using m/g/m/m+ r queuing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 5, pp. 936–943, 2011.

[22] H. Yuan, J. Bi, W. Tan, M. Zhou, B. H. Li, and J. Li, "Ttsa: An effective scheduling approach for delay bounded tasks in hybrid clouds," *IEEE Transactions on Cybernetics*, vol. 47, no. 11, pp. 3658–3668, 2016.

[23] M. Shojafar, S. Javanmardi, S. Abolfazli, and N. Cordeschi, "FUGE: a joint meta-heuristic approach to cloud job scheduling algorithm using fuzzy theory and a genetic method," *Cluster Computing*, vol. 18, no. 2, pp. 829–844, 2015.

[24] H. Liu, A. Abraham, and A. E. Hassanien, "Scheduling jobs on computational grids using a fuzzy particle swarm optimization algorithm," *Future Generation Computer Systems*, vol. 26, no. 8, pp. 1336–1343, 2010.

[25] F. Palmieri, L. Buonanno, S. Venticinque, R. Aversa, and B. Di Martino, "A distributed scheduling framework based on selfish autonomous agents for federated cloud environments," *Future Generation Computer Systems*, vol. 29, no. 6, pp. 1461–1472, 2013.

[26] M.-R. Shie, C.-Y. Liu, Y.-F. Lee, Y.-C. Lin, and K.-C. Lai, "Distributed scheduling approach based on game theory in the federated cloud," in *2014 International Conference on Information Science & Applications*. IEEE, 2014, pp. 1–4.

[27] M. Gen and R. Cheng, *Genetic algorithms and engineering optimization*. John Wiley & Sons, 2000, vol. 7.

[28] M. Dorigo and G. Di Caro, "Ant colony optimization: a new meta-heuristic," in *Proc. 1999 Congress on Evolutionary Computation*, vol. 2, 1999, pp. 1470–1477.

[29] J. Kennedy, "Particle swarm optimization," in *Encyclopedia of machine learning*. Springer, 2011, pp. 760–766.

[30] S. Mirjalili and A. Lewis, "The whale optimization algorithm," *Advances in Engineering Software*, vol. 95, pp. 51–67, 2016.

[31] L. Cheng, Y. Wang, Q. Liu, D. H. Epema, C. Liu, Y. Mao, and J. Murphy, "Network-aware locality scheduling for distributed data operators in data centers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 6, pp. 1494–1510, 2021.

[32] W. Zheng, M. Tynes, H. Gorelick, Y. Mao, L. Cheng, and Y. Hou, "Flowcon: Elastic flow configuration for containerized deep learning applications," in *Proc. 48th International Conference on Parallel Processing*, 2019, pp. 1–10.

[33] W. Zheng, Y. Song, Z. Guo, Y. Cui, S. Gu, Y. Mao, and L. Cheng, "Target-based resource allocation for deep learning applications in a multi-tenancy system," in *Proc. 2019 IEEE High Performance Extreme Computing Conference*, 2019, pp. 1–7.

[34] Y. Mao, W. Yan, Y. Song, Y. Zeng, M. Chen, L. Cheng, and Q. Liu, "Differentiate quality of experience scheduling for deep learning applications with docker containers in the cloud," *arXiv preprint arXiv:2010.12728*, 2020.

[35] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[36] Q. Liu, L. Cheng, T. Ozcelebi, J. Murphy, and J. Lukkien, "Deep reinforcement learning for IoT network dynamic clustering in edge computing," in *Proc. 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, 2019, pp. 600–603.

[37] H. Peng and X. S. Shen, "Deep reinforcement learning based resource management for multi-access edge computing in vehicular networks," *IEEE Transactions on Network Science and Engineering*, 2020.

[38] J. Ho and S. Ermon, "Generative adversarial imitation learning," in *Advances in Neural Information Processing Systems*, 2016, pp. 4565–4573.

[39] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne, "Imitation learning: A survey of learning methods," *ACM Computing Surveys*, vol. 50, no. 2, pp. 1–35, 2017.

[40] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[41] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *International conference on machine learning*. PMLR, 2016, pp. 1995–2003.

[42] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[43] T. Gangwani, Q. Liu, and J. Peng, "Learning self-imitating diverse policies," in *7th International Conference on Learning Representations, ICLR*, 2019.

[44] Y. Guo, J. Oh, S. Singh, and H. Lee, "Generative adversarial self-imitation learning," *arXiv preprint arXiv:1812.00950*, 2018.

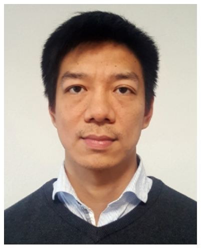
[45] B. D. Ziebart, J. A. Bagnell, and A. K. Dey, "Modeling interaction via the principle of maximum causal entropy," in *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, J. Fürnkranz and T. Joachims, Eds. Omnipress, 2010, pp. 1255–1262.

[46] D. Li, C. Chen, J. Guan, Y. Zhang, J. Zhu, and R. Yu, "Dcloud: deadline-aware resource allocation for cloud computing jobs," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 8, pp. 2248–2260, 2015.

[47] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[48] Q. Wu, M. Zhou, Q. Zhu, Y. Xia, and J. Wen, "Moels: Multiobjective evolutionary list scheduling for cloud workflows," *IEEE Transactions on Automation Science and Engineering*, vol. 17, no. 1, pp. 166–176, 2019.

Yifeng Huang is a master student in North China Electric Power University (NCEPU) in Beijing. He received his B.Eng. degree from NCEPU in 2020. His research mainly focuses on cloud computing, machine learning and reinforcement learning.



Long Cheng is a Full Professor in the School of Control and Computer Engineering at North China Electric Power University in Beijing. He received the B.E. from Harbin Institute of Technology, China in 2007, M.Sc from University of Duisburg-Essen, Germany in 2010 and Ph.D from National University of Ireland Maynooth in 2014. He was an Assistant Professor at Dublin City University, and a Marie Curie Fellow at University College Dublin. He also has worked at organizations such as Huawei Technologies Germany, IBM Research Dublin, TU Dresden and TU Eindhoven. He has published more than 60 papers in journals and conferences like TPDS, TON, TC, TSC, TASE, TCAD, TCC, TBD, TITS, TVLSI, JPDC, IEEE Network, IEEE Systems Journal, CIKM, ICPP, CCGrid and Euro-Par etc. His research focuses on distributed systems, deep learning, cloud computing and process mining. Prof Cheng is a Senior Member of the IEEE and an associate editor of Journal of Cloud Computing.

Lianting Xue is a master student in the School of Control and Computer Engineering at North China Electric Power University (NCEPU) in Beijing. She received her B.Eng. degree from NCEPU in 2020. She was a visiting student in Computer Science at Illinois Institute of Technology in Chicago. Her research interests are cloud computing and deep learning.

Cong Liu is a Full Professor with the Shandong University of Technology, China. He received the B.S. and M.S. degrees in computer software and theory from the Shandong University of Science and Technology, Qingdao, China, in 2013 and 2015, respectively, and the Ph.D. degree in computer science and information systems from the Eindhoven University of Technology, Eindhoven, The Netherlands, in 2019. He has more than 80 publications in journals and conferences like TSC, TSMC, TASE, TBD, TCC, TITS, IoTJ, FGCS, DSS, ICWS, ICPC, EuroVIS, and etc. His current research interests include business process management, process mining, Petri nets, machine learning and big data systems.

Yuancheng Li received the Ph.D. degree in pattern recognition and intelligent system from University of Science and Technology of China, Hefei, China, in 2003. From 2004 to 2005, he was a Postdoctoral Research Fellow with the Digital Media Lab, Beihang University, Beijing, China. Since 2005, he has been with the North China Electric Power University (NCEPU), Beijing, China, where he is currently a Full Professor and the Vice Dean with the School of Control and Computer Engineering. From 2009 to 2010, he was a Postdoctoral Research Fellow with the Cyber Security Laboratory, College of Information Science and Technology, Pennsylvania State University, State College, PA, USA. He has authored or coauthored more than 120 articles. His current research interest includes Cyber-Physical System (CPS) and CPS security, cloud computing, big data and machine learning.

Jianbin Li received the BS degree from Tsinghua University in 1992, and the MS degree from Analysis and Forecast Center of the State Seismological Bureau in 1995. He was the dean of the Institute of Information Security and Big Data at Central South University. He is currently a Full Professor with the North China Electric Power University in Beijing. His research interests mainly include information security, big data and deep learning.

Tomas Ward is AIB Chair of Data Analytics at the School of Computing, Dublin City University. As a member of the Science Foundation Ireland-funded research centre Insight – Ireland's Data Analytics research centre, Prof Ward studies how human health, performance and decision-making can be better understood through new ways of sensing and interpreting our physiology and behaviour. Prior to his position at Dublin City University he was a professor in the Department of Electronic Engineering at Maynooth University and led a research group focussing on neural engineering. Tomas holds B.E. (1994), M.Eng.Sc. (1996) and Ph.D (2000) degrees in engineering from University College Dublin. A Senior Member of the IEEE since 2011, Tomas has authored more than 240 peer-reviewed scientific publications. He is actively engaged in entrepreneurship and dissemination through commercialisation. He has licensed a range of technologies to industry since 2009 including sensor streaming technologies for e-health, over the air programming and mobile health applications. His current commercialisation activity is focussed on the development of mobile experience sampling technologies. Tomas is a keen advocate of hacker spaces and is a co-founder of Dublin Maker – the showcase of the maker movement in Ireland.