Deep Reinforcement Learning for Energy and Time Optimized Scheduling of Precedence-Constrained Tasks in Edge-Cloud Computing Environments

Amanda Jayanetti[a,*], Saman Halgamuge[b] and Rajkumar Buyya[a]

[a]Cloud Computing and Distributed Systems (CLOUDS) Lab, School of Computing and Information Systems, The University of Melbourne, Australia
[b]School of Electrical Mechanical and Infrastructure Engineering, The University of Melbourne, Australia

ARTICLE INFO

Keywords:
workflow scheduling
edge-computing
deep reinforcement learning
proximal policy optimization
energy

ABSTRACT

The wide-spread embracement and integration of Internet of Things (IoT) has inevitably lead to an explosion in the number of IoT devices. This in turn has led to the generation of massive volumes of data that needs to be transmitted, processed and stored for efficient interpretation and utilization. Edge computing has emerged as a viable solution which complements cloud thereby enabling the integrated edge-cloud paradigm to successfully satisfy the design requirements of IoT applications. A vast majority of existing studies have proposed scheduling frameworks for individual tasks and only very few works have considered the more challenging problem of scheduling complex workloads such as workflows across edge-cloud environments. Workflow scheduling is an NP hard problem in distributed infrastructures. It is further complicated when scheduling framework needs to coordinate workflow executions across resource constrained and highly distributed edge-cloud environments. In this work, we leverage Deep Reinforcement Learning for designing a workflow scheduling framework capable of overcoming the aforementioned challenges. Different from all existing works we have designed a novel hierarchical action space for promoting a clear distinction between edge and cloud nodes. Coupled with this a hybrid actor critic based scheduling framework enhanced with proximal policy optimization technique is proposed to efficiently deal with the complex workflow scheduling problem in edge-cloud environments. The results of extensive simulation experiments clearly demonstrate the superiority of our work in terms of enhancing the energy efficiency of workflow executions while also maintaining the total execution time in par with comparison algorithms.

1. Introduction

Recent advances in IoT (Internet of Things) facilitates a degree of intelligence infused connectivity between physical devices (Things) and the external environment that has revolutionized the manner in which digital services in the world operates. Accordingly, IoT has become an integral component that greatly enhances the convenience and efficiency of not only industrial operations, but also the day to day activities of individuals.

Despite numerous benefits offered by the Cloud computing paradigm, the traditional cloud computing architectures are largely agnostic to certain design requirements of emerging IoT applications. These include ultra low response time requirements, location-awareness, privacy and security concerns associated with sending data through public cloud platforms and so on. The inherently centralized architecture of the cloud computing paradigm necessitates the transmission of data between IoT devices and cloud over congestion prone wide area networks thus introducing additional delays. Furthermore, the unprecedented growth of IoT devices continues to impose a significant strain on cloud due to a multitude of factors including the need for processing and storing massive volumes of generated data, complexities associated with the transmission of huge volumes of data over networks with limited bandwidths and so on.

Edge computing extends computational resources to the edge of the networks thus enabling data to be processed and analyzed closer to the sources of generation. This enables the integrated cloud-edge paradigm to meet the QoS (Quality of Service) demands of latency-sensitive and bandwidth-hungry IoT applications while fully leveraging the benefits of public cloud platforms for compute-heavy processing and storage requirements. Furthermore, processing data at the edge of the networks cuts-down the volume of data that should be transmitted and stored in cloud, thus greatly reducing the overhead imposed on networks. Along with the promise of ultra-low latency processing, reduced bandwidth usage and so on, the edge computing paradigm itself introduces a fresh set of challenges. Scheduling dynamic workloads with diverse QoS requirements among heterogeneous and resource constrained edge nodes and cloud is one such challenge that should be efficiently addressed for fully harnessing the power of this novel computing paradigm.

In the existing literature, there are contradictory arguments about the energy-efficiency of edge computing infrastructures compared to centralized cloud datacenters. Some studies have concluded that edge computing solutions are more energy-efficient [18] while others have suggested that depending on the network infrastructure it could be less energy-efficient. However, an obvious fact is that, although a single node (terminal, edge) may not consume a high level of power, the combined energy consumption of billions of

*Corresponding author
amjayanetti@student.unimelb.edu.au (A. Jayanetti); saman@unimelb.edu.au (S. Halgamuge); rbuyya@unimelb.edu.au (R. Buyya)
ORCID(s):

edge nodes (e.g. IoT devices, service nodes) will impose a non-negligible impact on the underlying infrastructures, thereby threatening the end-to-end sustainability of the entire computing paradigm. Furthermore, owing to the need for decentralized deployment as well as other design requirements such as portability, ease of installation and maintenance, a significant proportion of edge nodes may be powered through batteries or energy harvesting devices with limited capacities.

Both academia as well as industry have rendered significant research efforts for efficiently addressing the aforementioned challenges. However, a vast majority of proposed scheduling techniques are aimed at independent task oriented workloads. Only very few studies have considered more complex workloads, such as those with precedence relations. In the remainder of this paper, we use the term workflow to refer to workloads with precedence-constrained tasks. Workflow is an application model that can be used to represent a wide variety of IoT applications (health care, stream processing, smart city applications etc.). Therefore, in this paper we focus on addressing the conflicting objectives of time minimization and energy optimization in scheduling DAG (Directed Acyclic Graphs) based workflows across cloud and edge computing environments. Specifically, we use Deep Reinforcement Learning (DRL) techniques which have proven to be efficient at handling highly dynamic and complex environments. Inherent characteristics of the Reinforcement Learning (RL) paradigm such as learning through experience coupled with the use of neural networks for function approximation, makes DRL an ideal candidate for handling the unpredictable dynamicity associated with edge computing environments.

We model the problem of energy and time optimized workflow scheduling in edge-cloud environments as a Markov Decision Process (MDP). Since energy-efficiency and time minimization are generally conflicting goals, it is crucial to emphasize the importance of establishing a balanced trade-off between these goals to the DRL agent. We achieve this by training the DRL agent to produce scheduling actions which minimize energy consumption of the system while meeting workflow deadlines in a best-effort manner.

The main contributions of this work are as follows:

- We propose a novel RL oriented problem formulation with a hierarchical action space for establishing a desired trade-off between the conflicting objectives of energy optimization and time minimization in workflow executions across cloud and edge computing environments. As opposed to existing studies in which all edge and cloud nodes are considered together in the action space of a single actor network, the proposed hierarchical action space promotes a clear distinction between edge and cloud nodes.
- We propose a hybrid DRL model comprising of two actor networks and one critic network. As opposed to the general case where a single actor network determines the node to which a task is assigned, the multi-actor network finely divides the responsibility of determining the tier (cloud/edge) and determining the node to separate actors, thus greatly speeding up the learning process. The critic network is used to guide both actor networks.
- For efficiently training the proposed DRL framework we use Proximal Policy Optimization (PPO) technique which is capable of overcoming the inherent sample inefficiency associated with traditional actor-critic methods with the use of a clipped surrogate objective function.
- We conduct extensive simulations for evaluating the performance of proposed algorithms. With experimental results thus obtained, we demonstrate that the proposed algorithms significantly outperform baseline scheduling algorithms.

The rest of the paper is organized as follows: In Section 2 we review background of the addressed problem along with relevant literature. In Section 3 we present the system model and formulate the objective of this work mathematically. Followed by this, the DRL oriented framework for scheduling is presented in Section 4. Section 5 and Section 6 present the evaluation of the proposed technique and conclusion of the study, respectively.

2. Related Work

In this section we review related works that uses RL for dependent and independent task scheduling in cloud and edge computing environments.

2.1. Cloud Computing Environments

A number of studies [6], [14] have used RL for addressing the problem of task scheduling in cloud computing environments. In [6] Q-learning is used for prioritizing tasks allocated to servers so that energy efficiency of cloud resources are maximized. Q-learning was also used in [14] together with queuing theory for scheduling tasks in cloud computing environments under the presence of resource constraints.

RL based scheduling algorithms are proposed in several works [1], [2], [27], [9], [15], [4] for scheduling dependent tasks of workflows in cloud computing environments. In [1] Q-learning was used for sorting the tasks of a workflow prior to provisioning resources for its execution. Multiple reinforcement learning agents were used to compute an average Q-value of each node in a workflow which was then used to sort tasks in ascending order. In the resource provisioning phase, co-operative multi-agent coordination was achieved through a Markov game for determining the tasks which should execute on a particular resource, with the objectives of optimizing energy consumption and cost. A combination of multi-agent coordination together with the on-policy RL algorithm SARSA and genetic algorithm was used for a similar workflow scheduling problem in [2]. In [27], a multi-agent reinforcement learning framework for

Work	Application Model		Edge-Cloud	Algorithm	Heterogeneous	Objectives
	Workflow	BoT				
D. Ding et al. (2020)		✓		Q-Learning	✓	energy
Z. Peng et al. (2015)		✓		Q-Learning	✓	response time
M. Cheng et al. (2018)	✓			Deep Q-Learning	✓	energy
A. Asghari et al. (2020)	✓			Q-Learning	✓	energy, cost
A. Asghari et al. (2021)	✓			SARSA, Genetic Algorithm	✓	makespan, resource utilisation
Y. Wang et al. (2019)	✓			Deep Q-Learning	✓	cost, makespan
A. Kaur et al. (2020)	✓			Deep Q-Learning	✓	makespan
Q. Yao et al. (2020)	✓			Deep Q-Learning	✓	makespan, energy
T. Sen et al. (2019)		✓	✓	Q-Learning	✓	delay, energy
S. Tuli et al. (2020)		✓	✓	Asynchronous Advantage Actor Critic	✓	delay, energy, cost
P. Gazori et al. (2019)		✓	✓	Deep Q-Learning	✓	delay, cost
G. Rjoub et al. (2020)		✓	✓	Deep Q-Learning	✓	delay, resource utilisation
Y. Zhang et al. (2020)	✓		✓	Temporal Difference Learning		delay, energy
H. Lu et al. (2020)	✓		✓	Deep Q-Learning		system cost
Proposed	✓		✓	Proximal Policy Optimization	✓	energy, makespan

Table 1
Summary of Literature Review

multi-objective workflow scheduling in cloud infrastructures is proposed. The proposed approach uses separate Deep Q Learning agents for each objective (cost and makespan) and the scheduling problem is designed as a Markov game with a correlated equilibrium. Deep Q Learning coupled was also used in [9] for workflow scheduling in cloud with the objectives of minimizing response time and makespan. A multi-stage Deep Q Learning framework has been proposed in [4] for scheduling tasks of DAG based jobs with the objectives of minimizing energy cost of cloud service providers. In the first stage, the server farm to which a task should be allocated is determined. Second stage determines the exact server to which the task is allocated for execution.

2.2. Edge-Cloud Environments

A number of studies [21], [26], [8], [17] have used RL for task scheduling in edge-cloud environments. A number of studies have used the popular TD learning based Q learning algorithm for enhancing the performance of task scheduling in edge cloud systems. Q learning was used in [21] to determine if a task should be assigned to the same edge node in which it originated, or to the nearby fog layer or to cloud to achieve the highest energy-efficiency while meeting the real-time processing requirements of the task. To reduce the dimension of the state space, they have discretized the vales of the state parameters (Bandwdith, CPU, stored energy) to a pre-defined number of levels. The use of function approximators such as neural networks for approximating the Q function is a better alternative for overcoming inherent disadvantages associated with state-space discretization. [8] proposed a Double Deep Q Learning algorithm for task scheduling in fog computing environment with the objectives of minimizing delay and computation cost. In [26], A3C (Asynchronous Advantage Actor Critic) technique is used together with R2N2 (Residual Recurrent Neural Networks for task scheduling and migration in edge-cloud environment. [17] used Deep Q Learning coupled with LSTM (Long Short Term Memory Networks) for task scheduling in cloud computing environments with the aims of optimizing resource utilization and minimizing execution delay.

Several studies [29], [10] used RL for scheduling dependent tasks in edge-cloud environment. [29] used a TD (Temporal Difference) learning based RL algorithm for scheduling dependent tasks of requests modeled as DAGs (Directed Acyclic Graphs) in an edge cloud environment with the objective of minimizing energy consumption while meeting user specified time constraints. However, this work assumes that the decision of offloading task executions to the cloud is made beforehand, and therefore only addresses the problem of scheduling tasks among multiple edge nodes. In [10], Deep Q Learning and LSTM networks were used to select the service nodes of dependent tasks in IoT applications with the objective of minimizing overall system cost.

3. System Model

In this section, we present the system model and the formulation of the workflow scheduling problem in the edge-cloud environment, which forms the basis of the DRL framework proposed in this work.

3.1. Application Model

Directed Acyclic Graph (DAG) is a popular execution model that can be used to represent a wide variety of applications. A workflow can be modelled as a DAG, $G = (V, E)$ where V represents the set of vertices and E represents the set of directed edges. Each vertex $v_i \in V$ represents a computing task $v_i = (c_i, d_{in}, d_{out})$ characterized by the CPU requirements of the task c_i, size of input data d_{in} and size of output data d_{out}. Each edge $e_{i,j} \in E$ represents a data dependency between tasks v_i and v_j denoted by $e_{i,j} = (d_{i,j})$ which indicates the size of data $d_{i,j}$ that needs to be transmitted from v_i to v_j. Accordingly, a precedence constraint exists between the two tasks and task v_i is a predecessor of task v_j and task v_j is a successor of task v_i. A task may have multiple predecessors and its execution can only be commenced when all of its predecessors have completed execution and all the data dependencies are satisfied. When all the precedence constraints of a task are satisfied, it is said to be in

ready state. The bottom most task of the workflow which has no successors is referred to as a sink task.

3.2. Network Model

By nature, a majority of edge nodes are likely to be resource constrained and therefore efficient collaboration among multiple edge nodes with heterogeneous processing capabilities is inarguably beneficial for optimizing the efficiency of the entire system. In this study we consider a cluster of edge nodes with diverse computing capabilities and energy efficiencies collaborating with each other for provisioning on-demand compute and network resources to IoT devices in the vicinity. Figure 1 illustrates a high level overview of the system architecture. We consider a master worker architecture in which a gateway node with an embedded scheduler acts as the master node and the rest of the edge nodes in the cluster act as slave nodes. The considered architecture comprises of a non-hierarchical topology in which all edge nodes have direct connectivity with the gateway node. The gateway node acts as a virtual controller for managing and scheduling resources in the edge cluster. All nodes periodically share their resource availability (CPU, Memory etc.) with the gateway node, so that the real-time status of the network and edge nodes are incorporated in the formulation of scheduling decisions. Further details on the proposed DRL framework for scheduling will be discussed in latter sections of this paper.

3.3. Delay Model

In the considered architecture, a task maybe executed in an edge node or in the cloud. The execution time of a task mainly depends on the computation delay and communication delay. Computation time (CT) of a task depends on the size of the task and the processing power of the node to which it is assigned for execution. If the task is assigned to a node with no idle capacity, then waiting time also contributes to total delay associated with task execution. Accordingly, computation time of task, t_i with size $L(t_i)$ in a server with processing rate F can be expressed as follows:

$$CT(t_i) = \frac{L(t_i)}{F} + WT(t_i) \tag{1}$$

where $WT(t_i)$ is the waiting time of the task at the server before its execution commences.

In order for the execution of task, t_i to be commenced, all the precedence constraints of the task must be satisfied. This means all the predecessors of t_i must have completed execution and the output data from predecessors required for the execution of t_i, must be available at the node to which t_i is assigned. Let t_j be an immediate predecessor of task t_i and the size of data to be transferred from t_j to t_i be $D(t_i, t_j)$. If the bandwidth between the execution nodes of t_i and t_j is B, the total transmission time (TT) can be denoted by the following equation:

$$TT(t_i, t_j) = \frac{D(t_i, t_j)}{B} \tag{2}$$

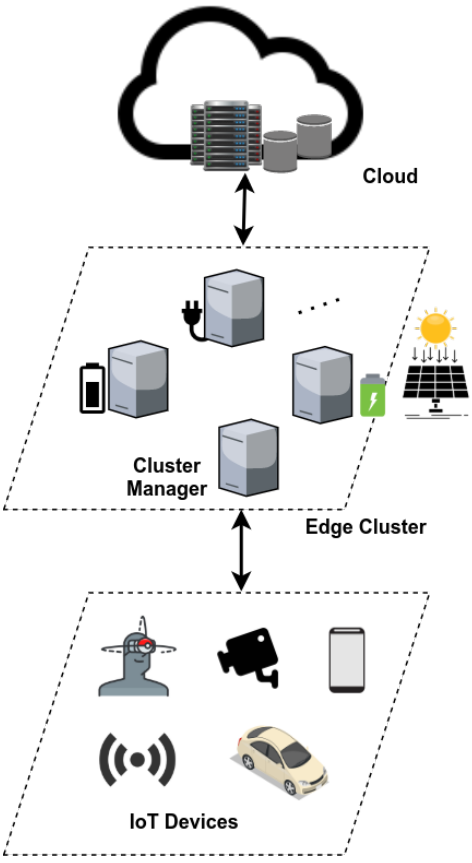


Figure 1: System Architecture

Accordingly, the earliest start time (EST) of task, t_i can be represented as follows:

$$EST(t_i) = \max_{t_j \in pred(t_i)} (FT(t_j) + TT(t_i, t_j)) \tag{3}$$

where $FT(t_j)$ is the finish time of task t_j and $pred(t_i)$ is the set of predecessors of task t_i. The finish time (FT) of task t_i can then be represented as:

$$FT(t_i) = EST(t_i) + CT(t_i) \tag{4}$$

The completion time (MT) of a workflow submitted to the system at time, ST will then be equivalent to the finish time of the task that completes execution last as represented by the following equation:

$$MT = \max_{t_i \in T}(FT(t_i)) \tag{5}$$

where T represents the set of all tasks of the workflow.

3.4. Energy Consumption Model

Energy consumed during the execution of a workflow is the aggregate of the computation energy and communication energy incurred during the execution of workflow tasks.

With CPU utilization based power consumption model [13] the energy consumed during the computation of task, t_i can be expressed as follows:

$$ECOMP(t_i) = CT(t_i) \times [U \times P_{active} + P_{idle}] \quad (6)$$

where P_{active} and P_{idle} are the power consumption rates at active and idle states of the processors and U is the current CPU utilization level of the server. Energy consumption associated with the transmission of data from the predecessor tasks is denoted as below:

$$ECOMM(t_i) = \sum_{t_j \in pred(t_i)} TT(t_i, t_j) \times P_{comm} \quad (7)$$

where P_{comm} is the power consumption associated with the transmission of data. Accordingly. the total energy consumed during the execution of a workflow can be denoted by the following equation:

$$E = \sum_{t_i \in T} (ECOMP(t_i) + ECOMM(t_i)) \quad (8)$$

3.5. Deadline Model

The primary goal of this work is to minimize energy consumption associated with workflow executions, and reduction in energy consumption is usually achieved at the expense of increased execution times. Deadlines are used in this work to establish a soft upper bound on the degree to which execution time is allowed to increase in exchange for higher energy savings.

As opposed to individual jobs, a workflow consists of multiple tasks all of which cannot be executed in parallel owing to the presence of precedence constraints among them. As discussed in latter sections of this paper, we transform the workflow scheduling problem to a task scheduling problem so that the scheduling decisions can be made in real time as tasks become ready for executions with the fulfilment of their precedence constraints. Therefore, it is important to decompose workflow deadlines to individual task deadlines, so that the scheduler can make informed decisions, taking into account the deadline of tasks. For this purpose we leverage the CP-P deadline decomposition technique presented in [30]. Accordingly deadline ($d(t_i)$) of a task (t_i) can be computed as follows:

$$d(t_i) = ST + (D - ST) * \frac{r(A) + \overline{w(A)} - r(i)}{r(A) + \overline{w(A)}} \quad (9)$$

where D is the workflow deadline, and ST is the time at which workflow is submitted. A is the source task of the workflow and $\overline{w(A)}$ is the average execution time of A. The upward rank (r_i) of task t_i [25] is computed recursively with the following equation:

$$r(i) = \max_{t_k \in succ(t_i)} (TT(t_i, t_k) + r(t_k) + \overline{w(t_k)}) \quad (10)$$

where $succ(t_i)$ is the set of successors of t_i and $TT(t_i, t_j)$ is the transmission time computed with equation 2.

3.6. Objective

The objective of this work is to minimize the completion time of workflows submitted to the system while also minimizing the total energy consumption of the entire system. Therefore the scheduling objective can be represented as a bi-objective function considering completion time and energy consumption of workflows as follows:

$$\begin{aligned} &\text{Minimize: } \sum_{i=1}^{N} \alpha M_i + (1-\alpha) E_i \\ &\text{Subject to: } M_i \leq D_i \end{aligned} \quad (11)$$

where N is the total number of workflows submitted to the system, and M_i and E_i are makespan and energy consumed during the execution of a workflow i, respectively. D_i is the deadline of the workflow. Since delay and energy are conflicting goals, in the above bi-objective function we use a normalized weight factor α for determining the degree of prominence that should be given to each goal based on system requirements.

With the deadline decomposition technique introduced in previous subsection, an individual deadline ($d(t_i)$) is assigned to each sub task (t_i). Accordingly, constraint in equation 11 can be rewritten as:

$$\forall_{t_i \in T} \quad FT(t_i) \leq d(t_i) \quad (12)$$

The use of deadline constraints in this manner allows the expansion of makespan within predefined upper bounds, so that further gains in energy efficiency can be achieved. Accordingly for a system that schedules a set of workflows (W), the objective function can be reformulated as:

$$\begin{aligned} &\text{Minimize: } \sum_{i=1}^{N} E_i \\ &\text{Subject to: } \forall_{w \in W} \forall_{t_i \in T} \quad FT(t_i) \leq d(t_i) \end{aligned} \quad (13)$$

4. DRL based Application Scheduling Framework

In this section, we provide a brief overview of the RL paradigm. Followed by this, we present an RL-oriented formulation of the energy and delay optimized workflow scheduling problem in the edge-cloud environment. We then describe the proposed DRL framework for efficiently handling the workflow scheduling problem.

4.1. Background

Reinforcement learning is a branch of Machine Learning which operates by training an agent to learn a desired behavior in an interactive environment based on the experiences it

Algorithm 1 Workflow Analyzing and Dispatching

```
upon event Submission of a workflow do
    Decompose workflow deadline to individual task deadlines
    for Each task in the workflow do
        if task is in Ready state then
            Dispatch the task to task scheduler
        Update global waiting-task map

upon event Receipt of a task completion notification do
    if Completed task is a Sink task then
        Send results to user
    else
        Use the waiting-task map to identify successors
        for Each successor of completed task do
            if task is Ready then
                Dispatch the task to task scheduler
            Update global waiting-task map
return
```

encounters. Essentially, the agent receives a reward for each action that it performs in a particular state and this reward serves as an indication of the success of the chosen action in that state. For instance, taking the action, a_t in the current state, s_t transitions the environment to a new state, s_{t+1} and the agent receives a reward, r_t. With sufficient training the agent learns to perform actions which result in the highest accumulated reward over time. Markov Decision Process (MDP) is commonly used for modelling the environment in which the RL agent operates. Accordingly, state transitions and rewards are considered to be governed by Markov Property, which assumes that the next state and reward depends solely on the current state and the action taken by the agent in the current state.

Goal of an RL agent is to maximize the expected cumulative discounted reward:

$$E[\sum_{t=0}^{\infty} \gamma^t r_t] = E[r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + ...] \quad (14)$$

where γ is the discounting factor, and $\gamma \in (0, 1)$. A policy, $\pi(a_t|s_t)$ is the strategy which dictates the course of actions to be followed by an agent to achieve the desired goal.

Owing to the high dimensionality of the complex environment associated with our problem, it is impossible to store data (states, actions) in a tabular format due to space constraints associated with storing experiences as well as the in-feasibility for an agent to explore all states during the training process. The use of neural networks as function approximators has emerged as a remarkably successful way of overcoming the aforementioned problems. Accordingly, the policy $\pi(a_t|s_t)$ can be modelled as a parameterized function with respect to an adjustable parameter θ, as $\pi_\theta(a_t|s_t)$. In order to evaluate the performance of the policy, we define a performance objective $J(\theta)$, which can be expressed in terms of cumulative discounted reward as shown in Equation 15.

$$J(\theta) = E_{\pi_\theta}[\sum_{t=1}^{\infty} \gamma^t r_t] \quad (15)$$

Policy gradients are a branch of RL algorithms that directly learn the parameterized policy. This is done by estimating the gradient of $J(\theta)$ with respect to each policy parameter, and updating the policy parameter in the direction of the gradient as shown below.

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta J(\theta_t) \quad (16)$$

REINFORCE [28] is a popular policy gradient algorithm based on the Monte Carlo method. It uses cumulative discounted rewards (v_t) from sample trajectories as an unbiased estimate of $Q(s_t, a_t)$ and updates policy parameters via gradient ascent as shown in Equation 17. This however gives rise to slow convergence and high variance in gradient estimates due to the possibility of high deviations between trajectories. Therefore, we used Actor-Critic technique as the basis of our scheduling framework.

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta \log \pi_\theta(a_t|s_t) v_t \quad (17)$$

4.2. RL Oriented Problem Formulation

Now we propose an RL oriented formulation of the workflow scheduling problem in edge-cloud environment. As opposed to individual jobs, a workflow cannot be executed at once owing to the complex precedence relations amongst workflow tasks. Although, some studies have attempted to determine in advance, the servers in which all the workflow tasks are to be executed [22], such approaches are less appropriate in a highly dynamic edge-cloud environment as the conditions may have significantly changed from the time scheduling decisions were made to the time the tasks (particularly the ones towards the bottom of the workflow) are actually scheduled for execution. Therefore, in this work we design the RL model in a manner such that all scheduling decisions are made in real time when workflow tasks are actually ready for execution.

For this we transform the complex workflow scheduling problem to a task scheduling problem, such that each scheduling decision corresponds to an allocation of a task whose precedence constraints are met, to an edge or cloud node for execution. Accordingly, the scheduler in gateway node maintains a map of pending tasks which await the completion of their predecessors for commencing execution. Whenever a task completes execution in an edge node or in the cloud, the gateway node is notified. With the receipt of this notification, the scheduler uses the proposed DRL based resource scheduling framework to determine where to execute any successors of the completed task which may now be in ready state. In the context of this problem, the DRL agent

is the task scheduler which resides in the cluster manager. Algorithm 1 summarizes the sequence of events involved in the execution of aforementioned steps.

State Space All the worker nodes periodically share their resource capacities, power consumption rates, queue statuses etc. with the gateway node as described in Section 3.2. Accordingly, a comprehensive state-representation which includes the real time status of the network together with the resource requirements and deadline of the task to be scheduled is provided to the DRL agent as described below. Specifically, the following properties will be incorporated in the state:

1. Total CPU and Memory requirements of the task, t_i
2. Deadline of task, t_i. The ultimate objective of the scheduling problem is to ensure the QoS requirements (e.g. deadlines) of workflows are satisfied while energy consumption of the system is minimized. Since the workflow scheduling problem is mapped to a task scheduling problem, from the perspective of the DRL agent, the goal would be meet the deadlines of individual tasks whilst minimizing system energy consumption. Therefore we decompose workflow deadlines to individual task deadlines so that the scheduling actions of the agents could be rewarded or penalized depending on their propensity to meet the deadline of the task in consideration.
3. An array $(d_1, d_2, ..., d_j)$ each element of which represents the amounts of data to be transferred from each node, j to the node in which the current task will be allocated before its execution can commence. Specifically, the amount of data to be transferred from node j is the sum of total input data from all predecessors that executed in node j. If none of task's predecessors executed in node j, then $d_j = 0$.
4. Total capacity and utilization of CPU, Memory and Bandwidth of each node
5. CPU frequency in Million Instructions per Second (MIPS) of each node
6. Rate of power consumption at idle and active states of each node
7. Approximate time at which a newly scheduled task can commence execution at each node (This is equivalent to the sum of execution times of tasks queued for execution at the node and the time remaining for tasks which are already in execution to complete)

Action Space We formulate a hierarchical action space for determining the task allocation node in cloud-edge environment. As opposed to the commonly used approach [26], [21] where all edge and cloud nodes are considered together in the same action space, the proposed hierarchical action space formulation enables the RL framework to clearly distinguish between cloud and edge nodes. Furthermore, it substantially reduces the action space of each agent, hence greatly expediting the training process.

Accordingly, a complete action produced by the RL framework can be represented as (a_1, a_2) where a_1 is the action which indicates the layer (cloud/edge) to which the task under consideration (t_i) should be allocated, and a_2 indicates the node (in the tier given by a_1) to which t_i should be assigned for execution. Therefore the action space can be defined as follows:

$$A = \{(a_1, a_2) | a_1 \in \{Cloud, Edge\} \ \varsigma \ a_2 \in \{1, 2, ..., N_{a_1}\}\} \tag{18}$$

where N_{a_1} is the total number of nodes that belong to the tier given by action a_1. We then adapt the hybrid actor critic technique proposed in [7] for parameterized action spaces, to the hierarchical action space in our problem. Details on the proposed DRL framework will be presented in the next section.

Note that the state space only includes details of one ready task, and each action is for mapping this task to a node in either cloud or edge. But in reality at each actual time step, the Ready Task Queue (RTQ) will have one or more tasks ready to be scheduled for execution. If the details of all ready tasks are to be incorporated, the size of state space as well as action space would increase which in turn could impede the speed of agent's learning process. Therefore, in each actual time-step, we use the DRL agent multiple times to determine the nodes in which all tasks in RTQ are to be scheduled [11].

Reward It is imperative to design a reward that is inline with the objective of the scheduling problem, so that with sufficient training the agent learns to optimize the objective while meeting any constraints. The problem addressed in this work requires the agent to minimize the energy consumption of the system whilst ensuring QoS requirements as defined by the deadlines are met in a best effort manner. Accordingly, we define the reward with the following two components:

1. R_1: Total energy consumption of the system during the time elapsed since last action. Negative sign is required since we need the DRL agent to minimize energy consumption. Note that in the training process of the DRL agent we only considered computation energy consumption since energy consumed for communication is negligible in comparison to energy consumed for computations.
2. R_2: A constant positive or negative reward depending on whether the selected node is capable of meeting task deadline, $d(t_i)$ or not:

$$R_2 = \begin{cases} +1, & \text{if } d(t_i) > FT(t_i) \\ -1, & \text{otherwise} \end{cases} \tag{19}$$

where $FT(t_i)$ is the estimated completion time of task t_i at the selected node. It can be calculated as in equation 4.

Accordingly, the reward received by the DRL agent for each scheduling decision is computed as:

$$Reward = R_1 + R_2 \tag{20}$$

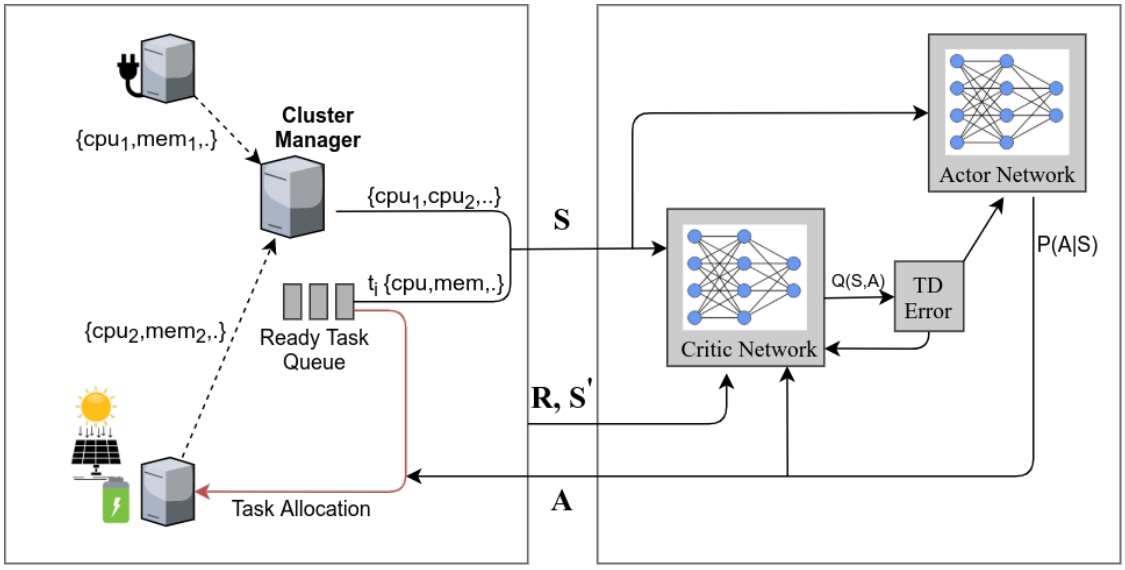


Figure 2: Traditional actor-critic based scheduling

Algorithm 2 Actor-Critic based Scheduling Framework with PPO

Initialize actor networks $\pi(a|s,\theta_1)$, $\pi(a|s,\theta_2)$ and critic network $V(s|\omega)$ with random weights
Initialize the training parameters: α, β, γ
for episode = 1 to N **do**
  Reset the environment
  for step = 1 to T **do**
    Input the state of the environment to actor networks $\pi(a|s,\theta_1)$, $\pi(a|s,\theta_2)$
    Select action a_1 (tier) from first actor network
    Select action a_2 (node) from second actor network
    Execute the combined action (a_1, a_2) and observe the corresponding reward r_t and next state of the system s_{t+1}
    Store the most recent transition (s_t, a_t, r_t, s_{t+1}) in memory D
  Compute advantage estimates $\hat{A}_1$ to $\hat{A}_T$
  for j = 1 to K **do**
    Randomly sample a mini-batch of samples of size S from D
    for i = 1 to S **do**
      Update critic network:
$\omega \leftarrow w + \beta \hat{A}_i \nabla V(s_t|\omega)$
      Update first actor network:
$\theta_1 \leftarrow \theta_1 + \alpha \hat{A}_i \nabla \ln \pi(a_1|s,\theta_1)$
      Update second actor network:
$\theta_2 \leftarrow \theta_2 + \alpha \hat{A}_i \nabla \ln \pi(a_2|s,\theta_2)$
  Clear memory D
return

Algorithm 3 Online Scheduling

upon event Submission of a task **do**
  Enqueue task in waiting-task queue
while Waiting-task queue is not empty **do**
  Deque a task from queue
  Get the latest status updates of all worker nodes
  Get resource requirements and predecessor relations of task
  Formulate the state space
  Action (a_1, a_2) = Agent(state)
  Submit the task description with the location of input data to the tier and worker node specified in Action
return

4.3. Actor-Critic based Scheduling Framework with Proximal Policy Optimization

Actor-critic is a branch of policy gradient algorithms that have proven to be efficient at overcoming the limitations of vanilla policy gradients by combining the advantages of value based methods and policy based methods. As the name implies, actor-critic algorithms consist of two main components; an actor and a critic. Actor is responsible for learning the policy, $\pi_\theta(a_t|s_t)$ which determines the action to be taken in each state for achieving the desired objective whereas critic is responsible for providing constructive criticism on the actions taken by the actor. It does so by learning the value function. The critic network can either be trained to learn the state-action value function, $Q(s_t, a_t|\omega)$ or the state-value function, $v(s_t|\omega)$. In this work, we train the critic to learn the latter since it proved to be a better fit for the problem modeled in this work. Note that we have used θ and ω to represent the set of adjustable parameters of the actor and critic networks respectively.

Critic network is initialized with arbitrary weights which

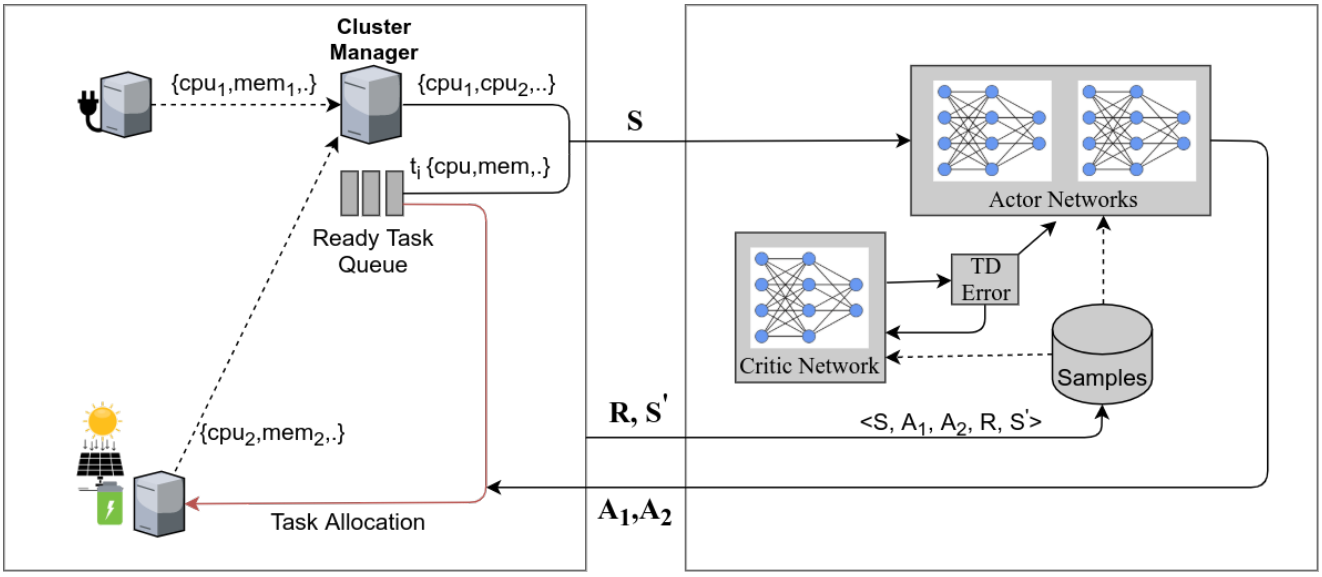


Figure 3: Proposed scheduling framework

are updated during the course of training thus allowing the critic to learn the actual state-value function. This is done by iteratively minimizing the mean squared difference (TD error) between network's predictions and target values ($R_t + v(s_{t+1}|\omega)$) as shown in equation 22. And then updating the network parameters with TD error as shown in equation 23.

$$L(\omega) = \frac{1}{2}(v_\pi(s_t|\omega) - (R_t + v_\pi(s_{t+1}|\omega)))^2 \tag{21}$$

$$\begin{aligned} \omega &\leftarrow \omega + \alpha\delta_t \nabla_\omega v_\pi(s_t|\omega) \\ where\ \delta_t &= R_t + v_\pi(s_{t+1}|\omega) - v_\pi(s_t|\omega) \end{aligned} \tag{22}$$

As training progresses, the critic network learns to more accurately predict the value of a given state. By incorporating the feedback from critic for updating policy parameters in the direction of improvements as shown in Equation 24, the actor-network also learns to produce actions that result in higher rewards. Note that we have used TD with one step look ahead as a Generalized Advantage Estimator (GAE), $\hat{A}_t$.

$$\theta \leftarrow \theta + \alpha\nabla \log \pi(a|s,\theta)\hat{A}_t \tag{23}$$

Despite the fact that actor critic methods solve problems associated with vanilla policy gradients such as high variance, the straightforward application of actor critic method did not work well for our problem. In fact these methods could take prohibitively long durations for learning complex policies due to the inherent sample inefficiency associated with them. The step size, $\alpha\nabla_\theta J(\theta_t)$ in vanilla policy gradient (Equation 16) cannot be made too large since that could lead to large policy updates that collapses performance. Trust Region Policy Optimization (TRPO) [19] techniques address the aforementioned problem by maximizing a surrogate objective function subject to a constraint (KL divergence) as shown in Equation 25. The constraint restricts the degree to which new policy is allowed to change from the old policy hence enabling the policy to monotonically improve (approximately). However, the theories which forms the basis of TRPO requires complex and computationally expensive calculations. Hence, we used Proximal Policy Optimization (PPO) [20] technique which is proven to provide the benefits of TRPO techniques, with the added advantages of less complexity, improved sample complexity and generalizability.

$$\begin{aligned} &\text{Maximize: } \hat{E}_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}\hat{A}_t\right] \\ &\text{Subject to: } \hat{E}_t[KL[\pi_{\theta_{\text{old}}}(.|s_t), \pi_\theta(.|s_t)]] \leqslant \delta \end{aligned} \tag{24}$$

If the surrogate objective function of TRPO in Equation 25 is maximized without a constraint, it could lead to large policy updates that in turn may adversely impact performance. Hence, the constraint is an imperative condition that should be satisfied when optimizing the objective. PPO attempts to find an alternate means for solving essentially the same problem, but without using an external constraint. It achieves this by limiting the degree to which new policy is allowed to change from the old policy by clipping the objective function as shown in equation 26.

$$\begin{aligned} &L^{CLIP}(\theta) = \hat{E}_t[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t] \\ &\text{where } r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \end{aligned} \tag{25}$$

As described in Section 4.2, we have proposed a novel hierarchical action space for the task scheduling problem in

edge-cloud environment. We then adapted a hybrid actor-critic technique [7] designed for parameterized action spaces to the hierarchical action space in our problem.

Figure 2 and Figure 3 illustrate high level overviews of traditional actor-critic based scheduling and the proposed hierarchical scheduling framework, respectively. As opposed to the traditional actor-critic technique which comprises of a single actor network and a single critic network, the proposed framework consists of two actors and one critic. Both actor networks use the same state space described in Section 4.2. Critic network also uses the same state space since it is trained to learn the state-value function and not the state-action value function which requires action input as well. The advantage given by the critic network is used to update the two stochastic actor networks.

Different from traditional actor-critic technique which only performs one gradient update with each experience sample, the use of PPO technique enables the proposed framework to stores experience samples in memory and perform multiple rounds of gradient updates with mini-batches of samples. As training progresses, the first actor-network learns to make the binary decision of whether to allocate a task either to cloud tier or edge tier. Since there are multiple nodes in which the task can execute in the tier selected by the first actor, the second actor-network learns to decide which node is most appropriate for task execution. Accordingly, the integrated output of which node in which tier is to be selected for task allocation is determined by the proposed hierarchical RL framework.

Pseudocode of the training process of the proposed multi-actor scheduling framework is presented in Algorithm 2. As indicated in lines 1-2 we first initialize the actor networks and critic network with random weights. Then the training parameters are also initialized. We train the DRL model for a total of N episodes (line 3), and at the beginning of each episode, the environment state is reset. Since, the problem is modeled as an input driven MDP, each time-step of the episode actually corresponds to scheduling of a task from Ready Task Queue. As indicated in lines 6-8, at each time-step the current state of the environment is given as input to the actor networks, and the output of the first actor network provides the tier to which the task should be allocated, and the second actor network's output provides the node in the selected tier to which the task should be allocated. Upon the execution of combined action (allocation of task to the selected node), the agent receives a reward and the environment transitions to a new state (line 9). Details of the transition which includes state of the environment, combined action, reward and next state are stored in memory as indicated in line 10. For each transition, the advantage estimates are also calculated and stored. At the end of each episode, we train the networks K times with randomly sampled mini-batch samples of size S (line 12-20). As opposed to techniques such as Deep Q Learning in which samples in memory are persisted over multiple episodes, with PPO technique the samples in the memory are cleared before starting the next episode of training.

Algorithm 3 summarizes the steps involved in online task scheduling process. As training the DRL model is a resource intensive and time consuming process, the DRL model is pre-trained and used in real time for obtaining the scheduling decisions. Real-time network status of all nodes together with the resource requirements of the task to be scheduled (dequeued from ready task queue) are merged for formulating the current state of the environment. It is then provided as input to the actor networks. The task is then allocated to the tier and node given by the combined action output of the actor networks.

5. Performance Evaluation

In this section we present a comprehensive analysis of the performance of the proposed DRL framework in comparison to several baseline algorithms in a number of different scenarios.

5.1. Experimental Setup

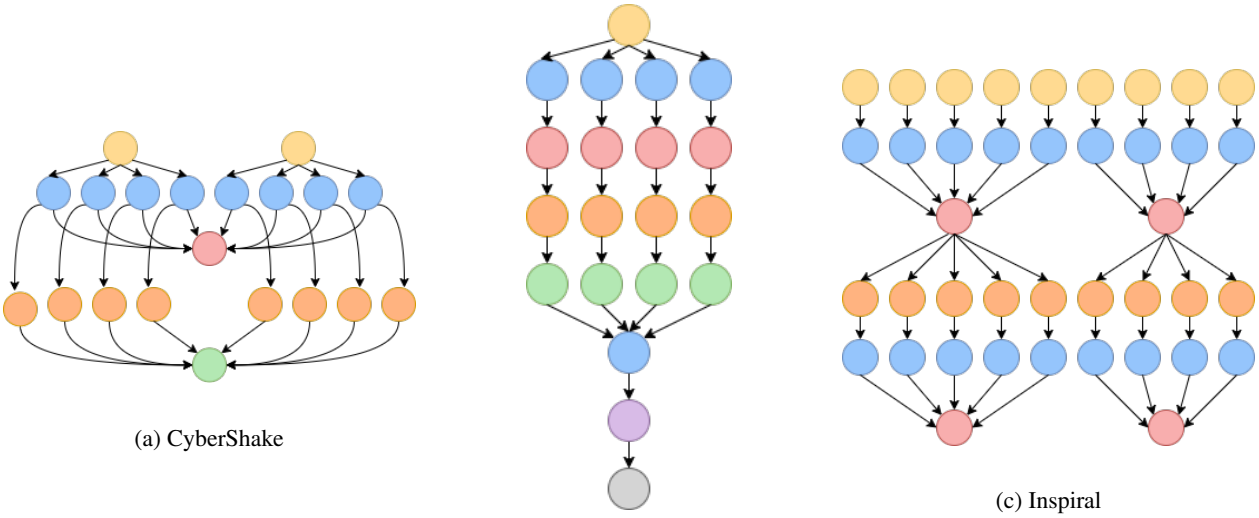
For evaluating the performance of proposed workflow scheduling framework, we used used an extension [23] of the popular CloudSim simulation toolkit. We have also implemented new modules for simulating the proposed workflow scheduling framework and interacting with the deep learning algorithms implemented using the deep learning library Keras [5].

The simulated scenario comprises of a highly heterogeneous cluster with 16 edge nodes and 8 cloud nodes. We used the SPEC benchmark [24] for obtaining the resource configurations and power consumption rates of nodes as shown in Table 2. The communication delay between edge-edge nodes and edge-cloud nodes was considered to be 1ms and 10ms respectively.

5.2. Dataset

The DAG structures of some real-world scientific workflows [3] are shown in Figure 4. Evaluation dataset was created based on synthetic workflow structures [16] provided by the popular Peagasus workflow framework. Task length in terms of number of instructions (as per CloudSim nomenclature) and the sizes of precedence constraints (in megabytes) were randomly selected from the ranges 0.5k to 1000k, and 0.1k to 10k respectively. A total of 1000 workflows comprising of 5292 tasks was used for the experiments. For simulating workflow arrival times, we used a Possion distribution.

Rather than using random deadlines we used a workflow aware process for setting realistically achievable deadlines considering the resources available in the simulated cluster. Critical path of a workflow is the longest execution path which essentially determines the total execution time of the workflow. For each workflow, we obtain the nodes in the critical path of the workflow [25]. Then we calculate the total execution time of the critical path using the processing speed of a randomly selected node in the cluster. The reason for using a random node's processing power rather than the average processing power of the cluster is to improve the diversity of the deadlines. We then add a deadline



(a) CyberShake

(b) Epigenomics

(c) Inspiral

Figure 4: DAG structures of real-world scientific workflows [3]

Layer	Server Name	Processor	Cores	MIPS	RAM (GB)	Bandwidth (GB/s)	Power (Watts)	
							Idle	Active
Cloud	Dell Inc. PowerEdge R740	Intel Xeon Platinum 8280 2.70 GHz	56	604.8k	64	1.5	50	432
Cloud	IBM System x iDataPlex dx360 M2	Intel Xeon X5570 2.933 GHz	16	187.712k	48	1	116	475
Edge	Fujitsu FUJITU Server PRIMERGY TX1320 M3	Intel Xeon E3-1230 v6 3.50 GHz	4	56k	8	1	9	51
Edge	Hewlett-Packard Company ProLiant DL385 G5	AMD Opteron processor 2356 2.3 GHz	8	55.2k	16	1	178	299
Edge	Hewlett-Packard Company ProLiant ML110 G4	Intel Xeon Processor 3040 1.86 GHz	2	14.88k	16	0.1	86	117

Table 2
Host configurations derived from SPEC benchmark [24] for experimental setup

base [8] which is a constant value for each resulting critical path execution time, to derive deadlines that are realistically achievable in the simulated environment. For this we used a trial and error method where the target being the selection of a deadline base with which EFT (Earliest Finish Time) algorithm (described in Section 5.3) can meet approximately 90% of deadlines.

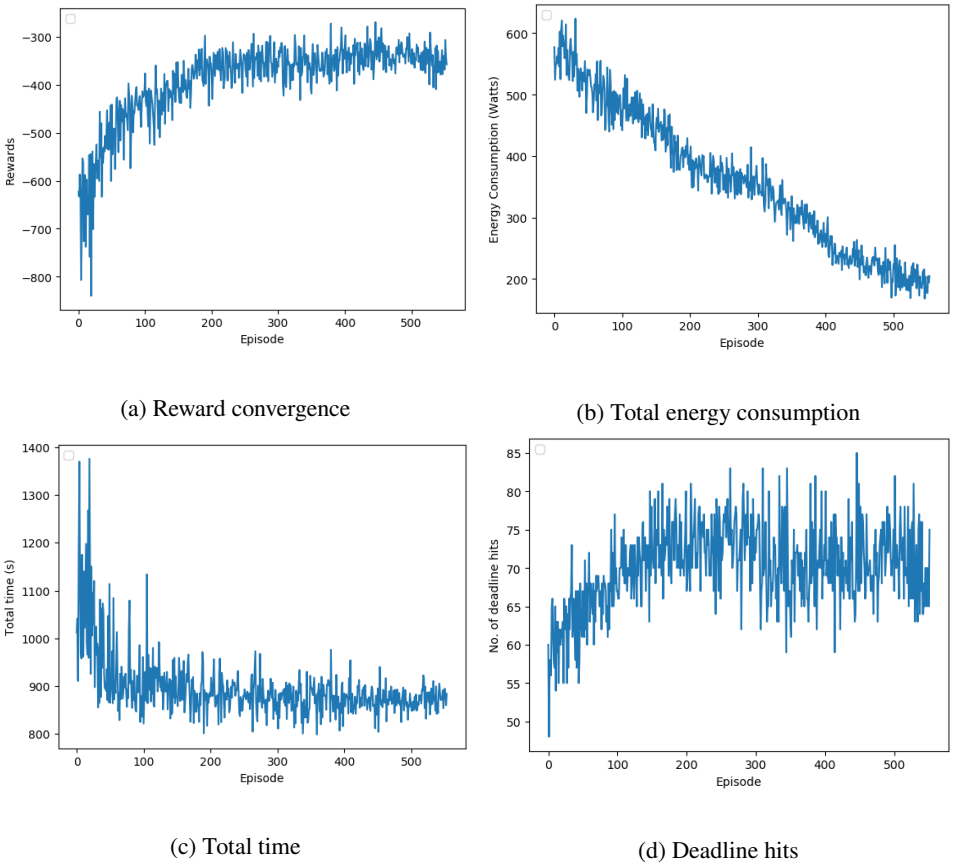
5.3. Comparison Algorithms

We used the following four algorithms for evaluating the performance of the proposed scheduling framework.

1. **Random:** This is a baseline algorithm which assigns tasks to a randomly selected node.
2. **EFT:** This algorithm is similar to the popular HEFT [25] algorithm except for the insertion based scheduling policy which is impractical in the considered edge-cloud scenario due to the lack of control over the processors of the distributed edge and cloud nodes. (i.e. once a task is allocated for execution to an edge or cloud node by the cluster manager, we do not assume it has further control over the manner in which tasks are actually scheduled for execution at the remote nodes.) It uses equations 1, 2, 3 and 4 for computing the estimated finish time of the task to be scheduled in all of the nodes, and assigns the task to the node with the earliest finish time. Where there are multiple ready tasks, tasks are prioritzed based on their upward ranks.
3. **EDA** This is an energy and delay aware algorithm which operates by assigning tasks to nodes that can reduce both delay as well as energy associated with task execution. Accordingly, it uses Equations 4 and 6 to compute the product of estimated delay and energy for executing a task at each of the nodes and selects the highest ranked node based on a score calculated as follows:

$$SCORE = FT(t_i) \times ECOMP(t_i) \tag{26}$$

4. **EES** This is an energy efficient scheduling algorithm



(a) Reward convergence

(b) Total energy consumption

(c) Total time

(d) Deadline hits

Figure 5: Learning progress with training (Convergence of DRL model)

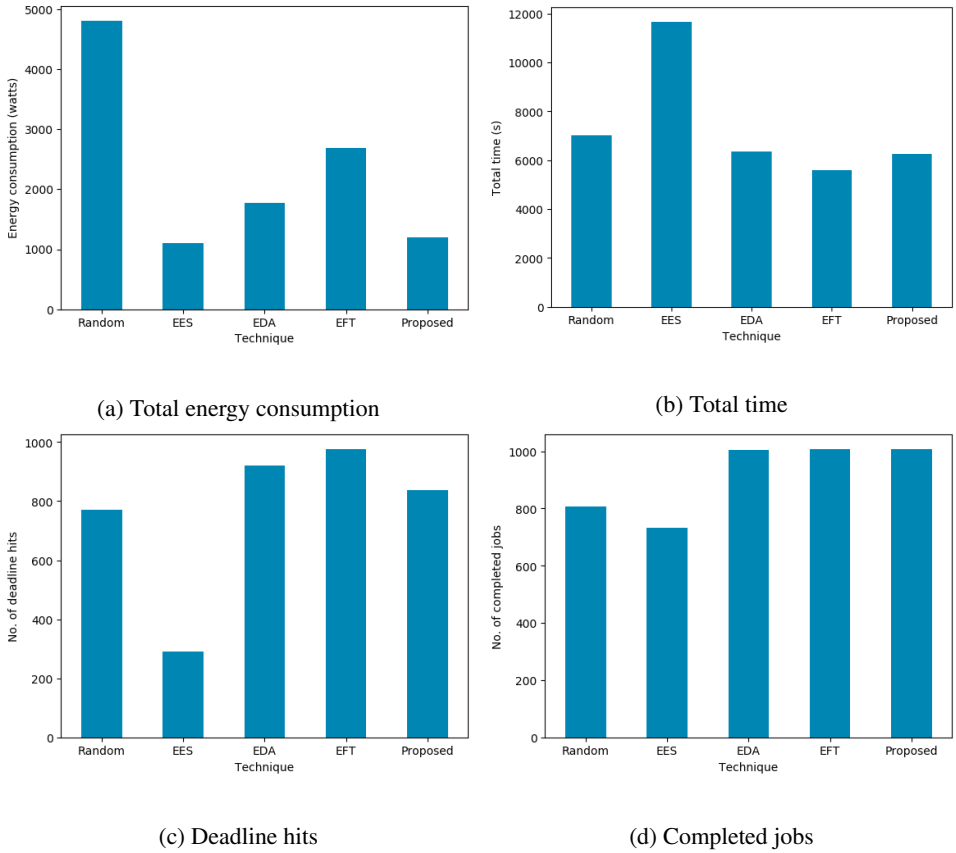
that solely aims to minimize energy consumption in a greedy manner by assigning the task to the node which requires the least amount of energy for its execution. It uses equation 6 to compute the energy consumed at each node for task execution.

5.4. Hyper-parameters and Network Configurations

Table 3 lists the hyper-parameters used for training the agents. The critic network is set to learn at a faster rate than the actor networks, since the actor networks rely on the guidance of the critic network, a critic network with a relatively faster learning rate speeds up the learning process. Furthermore, a step learning curve is used for the second actor, and thereby the first actor is set to learn at a slower rate than the second actor for the first 100 episodes. This is because during early episodes of training the reward largely depends on the actions produced by the second actor, so regardless of how good first actor's action is in a given state, the reward could still be bad due to second actor's action. Therefore, large weight updates for the first actor at early stages of training more often leads to sub-optimal convergence. Remaining hyper-parameters were chosen in a trial and error manner. We used 100 jobs in the training process of the DRL model. The model was trained 10 times with the hyper-parameters listed in the table and the model that produced best results was selected for conducting the experiments.

5.5. Analysis of Convergence of DRL Model

Figure 5 demonstrates the manner in which the agent learns to produce actions which leads to the achievement of desired objectives as training progresses. As shown in Figure 5a, total rewards accumulated during an episode gradually increase and converges to a maximum around the 550^{th} episode. As the reward is primarily designed for incentivizing the agent to minimize the energy consumption of the system, the total energy consumed by the system steadily decreases as shown in Figure 5b, and reaches a minimum around the 550th episode. As a part of the reward is designed to reward the agent for meeting task deadlines, or penalizing for failing to do so, total number of workflow deadlines hits during an episode also increases as shown in Figure 5d. Energy consumption and execution time are often contradictory goals. Therefore, we use task deadlines to convey the agent an upper bound on the degree to which execution time of a task can be compromised for a more energy-efficient allocation. As evidenced by the reduction in both energy consumption as well as execution time (Figure 5c), the agent learns to reduce both factors as training progresses. This is achieved by more frequently allocating tasks to nodes that are more efficient in terms of processing speed as well as energy consumption, so that task deadlines can also be met with relatively less energy consumption.



(a) Total energy consumption

(b) Total time

(c) Deadline hits

(d) Completed jobs

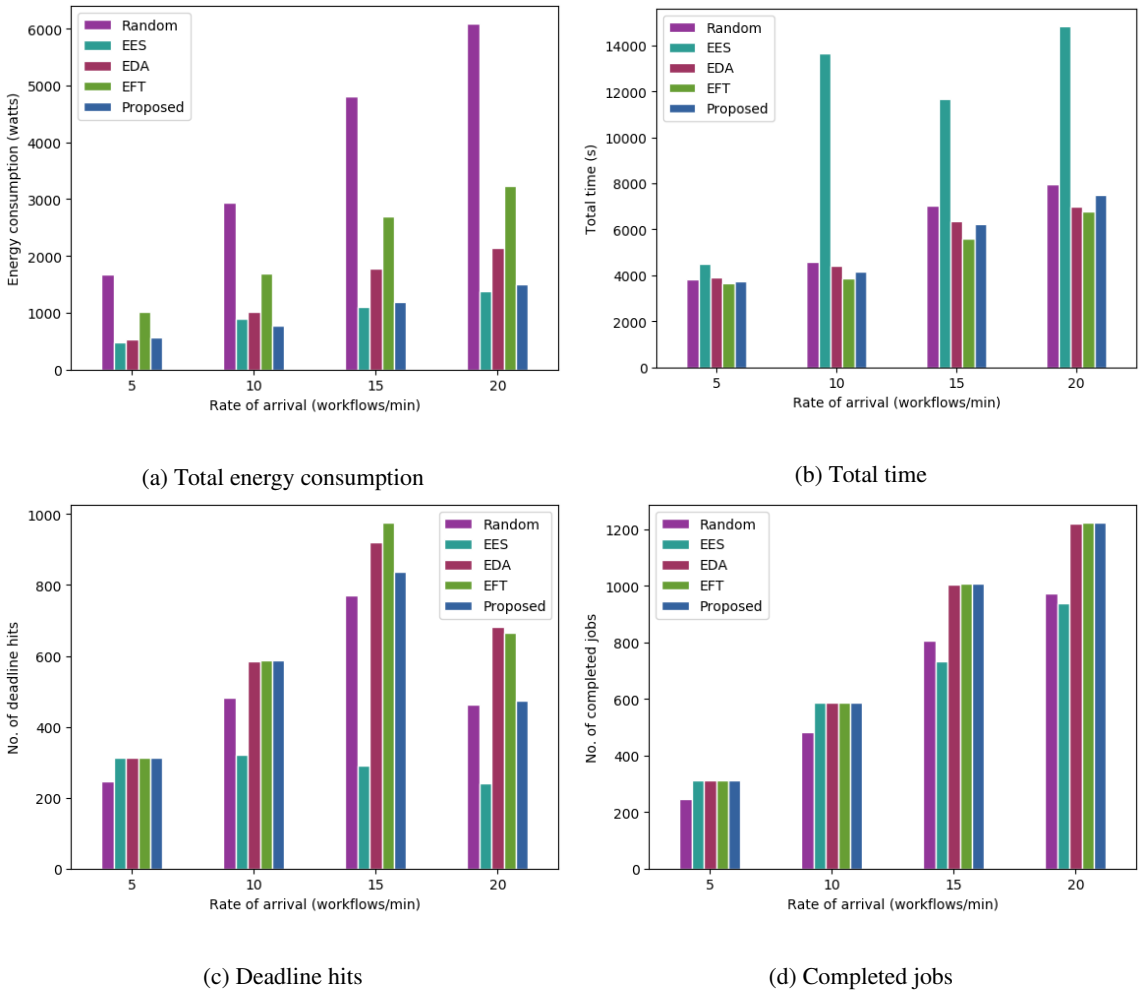
Figure 6: Comparison of performance of scheduling algorithms on experimental dataset

5.6. Analysis of Performance on Experimental Dataset

The total energy consumption incurred during the execution of workflows is demonstrated in Figure 6a. Clearly, Random algorithm has resulted in the highest energy consumption. This is due to the fact that the random task allocations among all cluster nodes, results in all the nodes of the cluster being active throughout the entire period leading to the under utilization of multiple cluster nodes. Since nodes continue to consume energy while they are in idle state as well, having all the nodes operating significantly below their capacities causes a steep degradation of energy consumption. Compared to random allocation, all other algorithms result in much better energy consumption. EES algorithm and the proposed DRL framework performs similarly with respect to energy savings, with the proposed technique consuming marginally more energy (8%). Energy consumed by EDA algorithm, though higher than EES and Proposed techniques, is much better in comparison with the EFT and Random algorithms. EFT algorithm consumes the second highest level of energy since it does not consider energy consumption of the system when making allocation decisions. The proposed DRL framework consumes 32%, 56% and 75% less energy compared to EDA, EFT and Random algorithms, respectively.

Figure 6b demonstrates the total time taken for the execution of workflows. With respect to total time, EES has performed the worst. This is expected as its sole focus is on the reduction of energy consumption without any regard to the subsequent impact on execution time. Since energy consumption and execution time are conflicting goals, the strategies employed by EES algorithm for saving energy increases the total execution time. Next highest level of execution time is incurred by the Random allocation algorithm. This is due to the fact that random allocation is completely indifferent to the location of where the predecessors of a task are hence resulting in very high communication times leading to increased total execution time. An improvement of 11% and 47% over the execution times of Random and EES algorithms is achieved by the proposed DRL framework, respectively. EDA and Proposed techniques have performed similar, with proposed technique taking slightly less time (2%). As expected, EFT has outperformed all the algorithms, since its sole focus is on minimizing execution time the allocation decisions are made such that each workflow can finish execution at the earliest time possible.

Percentage of deadlines met and jobs completed with each algorithm are demonstrated in Figure 6c and 6d respectively. EES algorithm has resulted in the lowest number of deadline hits (29%), which is significantly below the level of all other algorithms. The highest percentage of deadline hits (97%)is achieved by the EFT algorithm. This is expected since the objective of EFT algorithm is to complete the execution of each workflow within the shortest possible time,



(a) Total energy consumption

(b) Total time

(c) Deadline hits

(d) Completed jobs

Figure 7: Comparison of performance of scheduling algorithms at different workflow arrival rates

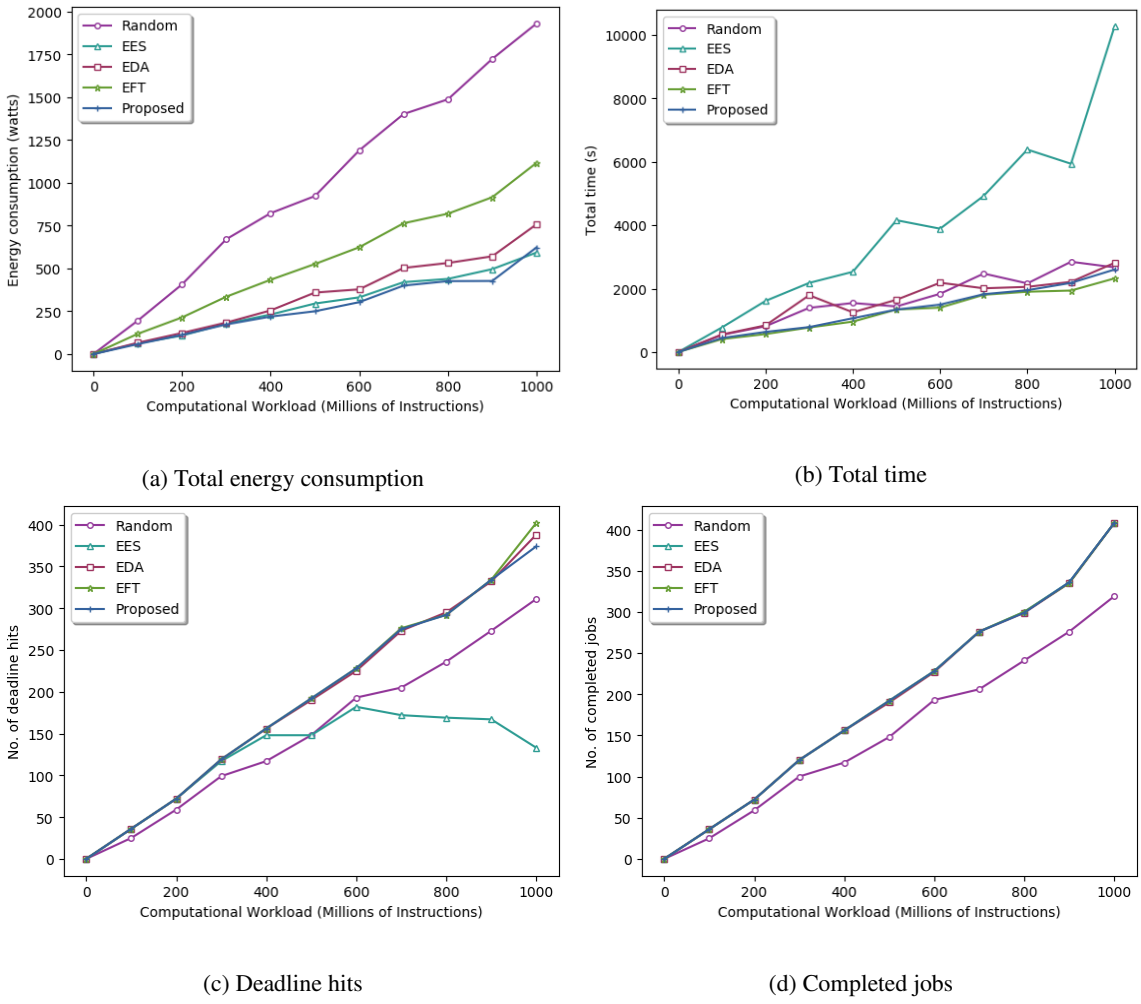
and that directly increases the probability of workflows completing execution within their deadlines. For similar reasons, EDA algorithm has achieved the second highest level of deadline hits (92%), as a part of its objective function is designed to favor nodes that contribute to minimizing the finish time of workflow tasks. The next highest number of deadline hits (84%) is achieved by the proposed DRL technique. Note that in this work we have considered deadlines to be soft deadlines, which means meeting them is desirable but not mandatory. Accordingly, the DRL model was not trained to meet all deadlines as a hard constraint, rather the training objective was formulated to primarily minimize energy consumption while using deadlines to control the degree to which makespan of workflows is allowed to increase in exchange for higher energy savings.

Proposed DRL technique, EFT as well as EDA take predecessor proximity into account in the formulation of allocation decisions which is crucial particularly when it comes to workflows with communication intensive precedence relations. Large waiting times that accompanies precedence relations agnostic scheduling decisions given by Random and EES algorithms result in timed out tasks which in turn lead to incomplete jobs as shown in Figure 6d.

5.7. Analysis of Performance at Different Workflow Arrival Rates

Figure 7 demonstrates the performance of algorithms in terms of energy consumption, total execution time, deadline hits and jobs completed as the workflow arrival rate varies. As shown in Figure 6a EES and proposed DRL technique have succeeded in keeping the energy consumption in a lower level than all other algorithms at all arrival rates. Random allocation leads to a drastic rise in energy consumption compared to other algorithms as arrival rate increases. With EFT technique as well the rise in energy consumption is more prominent compared to EDA, EES and proposed techniques. It is clear that with scheduling techniques which incorporate minimizing energy consumption as a part of their scheduling objectives, the energy consumption rises at a lower rate as arrival rate increases.

A moderate rise in total execution time can be observed in Figure 7b with all algorithms as the arrival rate increases, except with EES algorithm. Clearly, the scheduling decisions made by EES algorithm for optimizing energy consumption severely degrades the total execution time by in-



(a) Total energy consumption

(b) Total time

(c) Deadline hits

(d) Completed jobs

Figure 8: Comparison of performance of scheduling algorithms at different computational workloads

creasing the waiting times of task executions.

As indicated in Figure 7c and 7d, at moderate arrival rates EFT, EDA and proposed technique perform equally well with respect to total number of deadline hits achieved as well as total makespan. But at high arrival rates proposed DRL techniques performance degrades slightly more compared to EFT and EDA techniques. This is expected behaviour since the DRL model was trained at a moderate arrival rate, so as the arrival rate increases the learnt behavior may not be appropriate with respect to achieving certain goals since the environment in which the agent operates is changed significantly. This is in-fact a known drawback associated with DRL models trained in input driven environments [12]. In this work we have considered the workflow arrival rate to be moderate, however if burst arrival rates are also common, a feasible workaround to this problem maybe to train the model under different arrival rates [12].

5.8. Analysis of Performance at Different Computational Workloads

Figure 8 demonstrates the performance of the algorithms as the computational workload varies. In Figure 8a, Random algorithm's performance with respect to energy consumption severely degrades with increasing workload. EFT algorithm also consumes significantly more energy compared to EDA, EES and proposed techniques which consider energy efficiency as a sole or partial objective in the formulation of scheduling decisions. These algorithms are able to achieve a moderate rise in energy consumption with increasing workload as opposed to the sharp rise observable with non-energy aware scheduling algorithms. Proposed technique as well as EES clearly achieves the best results with very similar performance.

As previously discussed, the fact that EES algorithm only attempts to optimize energy consumption adversely impacts the total execution time, leading to significantly high execution times at heavy workloads. In contrast, EFT algorithm which operates with the only objective of minimizing execution time achieves the best results in execution time. EDA and Proposed DRL technique also achieves similar results with only a marginal increase in execution time compared to EFT at heavy workloads. Random algorithms performance cannot be evaluated solely based on the results in Figure 8b since it has completed less jobs compared to other

Parameter	Value
General	
Discount factor (γ)	0.2
Mini-batch size (S)	64
No. of mini-batch iterations per episode (K)	50
No. of training episodes (N)	550
Optimizer	Adam
Critic network	
Learning rate (β)	0.00005
No. of input layers	1
No. of output layers	1
No. of hidden layers	2
No. of neurons in each hidden layer	100
First actor network	
Learning rate (α)	0.00001
No. of input layers	1
No. of output layers	1
No. of hidden layers	2
No. of neurons in each hidden layer	100
Second actor network	
Learning rate (α)	0.00001
No. of input layers	2
No. of output layers	2
No. of hidden layers	4
No. of neurons in each hidden layer	100

Table 3
Hyper-parameters used for the DRL model

algorithms as indicated in Figure 8d.

In Figure 8c, EES algorithms ability to meet deadlines sharply drops with increasing workload. Random algorithm also performs poorly in comparison to EFT, EDA and proposed techniques which exhibit very similar performance. At heavy workloads, EFT algorithm marginally outperforms EDA as well as proposed techniques. As preiously discussed, this is expected since the sole focus of EFT algorithm is to speed up the execution of tasks which in turn leads to higher deadline hits.

6. Conclusions and Future Work

The problem of workflow scheduling itself is complicated due to the presence of complex precedence relations among workflow tasks. Scheduling workflows across edge-cloud environment adds an additional layer of complexity atop the general workflow scheduling problem owing to the fresh set of challenges associated with facilitating seamless executions across the highly heterogeneous and distributed edge-cloud environment.

In this work we propose a novel hierarchical state space formulation coupled with a hybrid actor-critic technique for energy-efficient resource scheduling in edge-cloud environment. The resulting DRL framework with multiple actor networks guided by a single critic network greatly reduces the size of the action space handled by each actor network while also promoting a clear distinction between edge and cloud nodes. Furthermore, we used proximal policy optimization technique to overcome the known limitations associated with traditional actor-critic methods. We also leveraged existing works to decompose workflow deadlines to individual task deadlines which were then used as soft upper-bounds during the training process, so that the DRL agent learns to establish a balanced trade-off between latency and energy consumption. Results of simulation experiments demonstrate that the DRL framework outperforms all other comparison algorithms by reducing the energy consumption of the system while maintaining the total execution time in par with other algorithms.

As part of the future work, we will implement the proposed DRL based scheduling framework in a real edge-cloud environment. We will also enhance the proposed framework by leveraging LSTM technique for enabling the DRL model to maintain an internal representation of history, which is proven to improve the ability of the DRL agents to adapt to varying conditions. Furthermore, we also intend to apply variance reduction techniques for stabilising the performance of DRL agents in the presence of highly different job arrival rates.

References

[1] Asghari, A., Sohrabi, M.K., Yaghmaee, F., 2020. A cloud resource management framework for multiple online scientific workflows using cooperative reinforcement learning agents. Computer Networks 179, 107340.

[2] Asghari, A., Sohrabi, M.K., Yaghmaee, F., 2021. Task scheduling, resource provisioning, and load balancing on scientific workflows using parallel sarsa reinforcement learning agents and genetic algorithm. The Journal of Supercomputing 77, 2800–2828.

[3] Bharathi, S., Chervenak, A., Deelman, E., Mehta, G., Su, M.H., Vahi, K., 2008. Characterization of scientific workflows, in: 2008 third workshop on workflows in support of large-scale science, IEEE. pp. 1–10.

[4] Cheng, M., Li, J., Nazarian, S., 2018. Drl-cloud: Deep reinforcement learning-based resource provisioning and task scheduling for cloud service providers, in: 2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC), IEEE. pp. 129–134.

[5] Chollet, F., et al., 2018. Keras: The python deep learning library. ascl , ascl–1806.

[6] Ding, D., Fan, X., Zhao, Y., Kang, K., Yin, Q., Zeng, J., 2020. Q-learning based dynamic task scheduling for energy-efficient cloud computing. Future Generation Computer Systems 108, 361–371.

[7] Fan, Z., Su, R., Zhang, W., Yu, Y., 2019. Hybrid actor-critic reinforcement learning in parameterized action space, in: Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI-19), pp. 2279–2285.

[8] Gazori, P., Rahbari, D., Nickray, M., 2019. Saving time and cost on the scheduling of fog-based iot applications using deep reinforcement learning approach. Future Generation Computer Systems .

[9] Kaur, A., Singh, P., Singh Batth, R., Peng Lim, C., 2020. Deep-q learning-based heterogeneous earliest finish time scheduling algorithm for scientific workflows in cloud. Software: Practice and Experience .

[10] Lu, H., Gu, C., Luo, F., Ding, W., Liu, X., 2020. Optimization of lightweight task offloading strategy for mobile edge computing based on deep reinforcement learning. Future Generation Computer Systems 102, 847–861.

[11] Mao, H., Alizadeh, M., Menache, I., Kandula, S., 2016. Resource

management with deep reinforcement learning, in: Proceedings of the 15th ACM Workshop on Hot Topics in Networks, pp. 50–56.

[12] Mao, H., Venkatakrishnan, S.B., Schwarzkopf, M., Alizadeh, M., 2018. Variance reduction for reinforcement learning in input-driven environments. arXiv preprint arXiv:1807.02264 .

[13] Pelley, S., Meisner, D., Wenisch, T.F., VanGilder, J.W., 2009. Understanding and abstracting total data center power, in: Workshop on Energy-Efficient Design.

[14] Peng, Z., Cui, D., Zuo, J., Li, Q., Xu, B., Lin, W., 2015. Random task scheduling scheme based on reinforcement learning in cloud computing. Cluster computing 18, 1595–1607.

[15] Qin, Y., Wang, H., Yi, S., Li, X., Zhai, L., 2020. An energy-aware scheduling algorithm for budget-constrained scientific workflows based on multi-objective reinforcement learning. The Journal of Supercomputing 76, 455–480.

[16] Ramakrishnan, L., Gannon, D., 2008. A survey of distributed workflow characteristics and resource requirements. Indiana University , 1–23.

[17] Rjoub, G., Bentahar, J., Abdel Wahab, O., Saleh Bataineh, A., 2020. Deep and reinforcement learning for automated task scheduling in large-scale cloud computing systems. Concurrency and Computation: Practice and Experience , e5919.

[18] Sarkar, S., Misra, S., 2016. Theoretical modelling of fog computing: a green computing paradigm to support iot applications. Iet Networks 5, 23–29.

[19] Schulman, J., Levine, S., Abbeel, P., Jordan, M., Moritz, P., 2015. Trust region policy optimization, in: International conference on machine learning, pp. 1889–1897.

[20] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O., 2017. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347 .

[21] Sen, T., Shen, H., 2019. Machine learning based timeliness-guaranteed and energy-efficient task assignment in edge computing systems, in: 2019 IEEE 3rd International Conference on Fog and Edge Computing (ICFEC), IEEE. pp. 1–10.

[22] Sharifi, M., Shahrivari, S., Salimi, H., 2013. Pasta: a power-aware solution to scheduling of precedence-constrained tasks on heterogeneous computing resources. Computing 95, 67–88.

[23] Son, J., Dastjerdi, A.V., Calheiros, R.N., Ji, X., Yoon, Y., Buyya, R., 2015. Cloudsimsdn: Modeling and simulation of software-defined cloud data centers, in: 2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, IEEE. pp. 475–484.

[24] SPEC, 2018. Spec, "standard performance evaluation corporation". URL: https://www.spec.org/benchmarks.html.

[25] Topcuoglu, H., Hariri, S., Wu, M.y., 2002. Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Transactions on Parallel and Distributed Systems 13, 260–274.

[26] Tuli, S., Ilager, S., Ramamohanarao, K., Buyya, R., 2020. Dynamic scheduling for stochastic edge-cloud computing environments using a3c learning and residual recurrent neural networks. IEEE Transactions on Mobile Computing .

[27] Wang, Y., Liu, H., Zheng, W., Xia, Y., Li, Y., Chen, P., Guo, K., Xie, H., 2019. Multi-objective workflow scheduling with deep-q-network-based multi-agent reinforcement learning. IEEE Access 7, 39974–39982.

[28] Williams, R.J., 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine learning 8, 229–256.

[29] Zhang, Y., Zhou, Z., Shi, Z., Meng, L., Zhang, Z., 2020. Online scheduling optimization for dag-based requests through reinforcement learning in collaboration edge networks. IEEE Access 8, 72985–72996.

[30] Żotkiewicz, M., Guzek, M., Kliazovich, D., Bouvry, P., 2016. Minimum dependencies energy-efficient scheduling in data centers. IEEE Transactions on Parallel and Distributed Systems 27, 3561–3574.