# Workflow Scheduling in Cloud and Edge Computing Environments with Deep Reinforcement Learning

Amanda Jayanetti

Submitted in total fulfilment of the requirements of the degree of

Doctor of Philosophy

School of Computing and Information Systems
THE UNIVERSITY OF MELBOURNE

August 2023

ORCID: 0000-0002-7178-3386

# Workflow Scheduling in Cloud and Edge Computing Environments with Deep Reinforcement Learning

Amanda Jayanetti
*Principal Supervisor: Prof. Rajkumar Buyya*
*Co-Supervisor: Prof. Saman Halgamuge*

---

**Abstract**

Cloud computing has firmly established itself as a mandatory platform for delivering computing services over the internet in an efficient manner. More recently, novel computing paradigms such as edge computing have also emerged to complement the traditional cloud computing paradigm. Owing to the multitude of benefits offered by cloud and edge computing environments, these platforms are increasingly used for the execution of workflows. The problem of scheduling workflows in a distributed system is NP-Hard in the general case. Scheduling workflows across highly dynamic cloud and edge computing environments is even more complex due to inherent challenges associated with these environments including the need to satisfy diverse contradictory objectives, coordinating executions across highly distributed infrastructures and dynamicity of the operating conditions. These requirements collectively give rise to the need for adaptive workflow scheduling algorithms that are capable of satisfying diverse optimization goals amid highly dynamic conditions.

Deep Reinforcement Learning (DRL) has emerged as a promising paradigm for dealing with highly dynamic and complex problems due to the ability of DRL agents to learn to operate in stochastic environments. Despite the benefits of DRL, there are multiple challenges associated with the application of DRL techniques including multi-objectivity, curse of dimensionality, partial observability and multi-agent coordination. In this thesis, we propose novel DRL algorithms and architectures to efficiently overcome these challenges. This thesis advances the state-of-the-art by making the following key contributions:

1. A comprehensive taxonomy and literature review on the scheduling of workflows across cloud and edge computing environments with the use of Reinforcement Learning

2. A joint host and network optimization technique for scheduling workflows in cloud data centers with the objective of minimizing energy consumption

3. A DRL technique for minimizing the energy consumption and makespan of workflow executions in edge-cloud environments

4. A multi-agent DRL technique for optimizing green energy utilization in the execution of workflows across multi-cloud environments

5. The design and implementation of a DRL agent capable of scheduling workflows in a cost efficient manner and its integration to an enterprise-grade workflow engine

6. A detailed study that outlines challenges and research directions associated with the scheduling of workflows in cloud and edge computing environments with the use of reinforcement learning

# Declaration

This is to certify that

1. the thesis comprises only my original work towards the PhD,
2. due acknowledgement has been made in the text to all other material used,
3. the thesis is less than 100,000 words in length, exclusive of tables, maps, bibliographies and appendices.

Amanda Jayanetti, August 2023

# Preface

## Main Contributions

This thesis research has been carried out in Cloud Computing and Distributed Systems (CLOUDS) Laboratory, School of Computing and Information Systems, The University of Melbourne. The main contributions of the thesis are discussed in Chapters 2-7 and are based on the following publications:

- **Amanda Jayanetti**, Saman Halgamuge, Rajkumar Buyya, "Reinforcement Learning based Workflow Scheduling in Cloud and Edge Computing Environments: A Taxonomy, Review and Future Directions", *ACM Computing Surveys (CSUR)*, 2023 (submitted, August 2023).

- **Amanda Jayanetti**, Rajkumar Buyya, "J-opt: A joint host and network optimization algorithm for energy-efficient workflow scheduling in cloud data centers", *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing*, Pages: 199–208, Auckland, New Zealand, December 2-5, 2019.

- **Amanda Jayanetti**, Saman Halgamuge, Rajkumar Buyya, "Deep reinforcement learning for energy and time optimized scheduling of precedence-constrained tasks in edge–cloud computing environments", *Future Generation Computer Systems*, Volume 137, Pages: 14–30, 2022.

- **Amanda Jayanetti**, Saman Halgamuge, Rajkumar Buyya, "Multi-agent Deep Reinforcement Learning Framework for Renewable Energy aware Workflow Scheduling in Distributed Cloud Data Centres", *IEEE Transactions on Parallel and Distributed*

*Systems*, 2024 (accepted for publication).

- **Amanda Jayanetti**, Saman Halgamuge, Rajkumar Buyya, "A Deep Reinforcement Learning Approach for Cost Optimized Workflow Scheduling in Cloud Computing Environments", *IEEE Transactions on Parallel and Distributed Systems*, 2024 (submitted, January 2024).

# Acknowledgements

I would like to thank my supervisors, Professor Rajkumar Buyya and Professor Saman Halgamuge for providing me with the opportunity of pursuing a PhD under their supervision. I appreciate the support, encouragement and guidance they provided on my PhD. I would also like to thank my advisory committee chair Professor Alistair Moffat for his valuable feedback and suggestions. I would like to express my sincere gratitude to my former advisory committee chair Professor Shanika Karunasekara, for the great assistance that was provided me to get through the challenging phases of my PhD journey.

I would also like to thank current and former members of CLOUDS Laboratory at the University of Melbourne for their support and friendship. In particular, I thank Dr. Shashikant Ilager, Dr. Muhammed Tawfiqul Islam, Dr. TianZhang He, Dr. Mohammad Goudarzi, Dr. Redowan Mahmud, Dr. Sara Kardani, Dr. Maria Rodriguez, Samodha Pallewatta, Anupama Mampage, Rajeev Muralidhar, Kwangsuk Song, Jie Zhao, Ming Chen, Siddharth Agarwal, Tharindu Bandara, Thanh-Hoa Nguyen, Yu- lun Huang, Zhiyu Wang, Kalyani Pendyala, Duneesha Fernando, Jayath Seneviratne, Chun Wei Lim, and Thakshila Mohottige.

I acknowledge the University of Melbourne for providing me with the scholarship and resources to pursue my doctoral studies. My research is also supported by a Discovery Project grant from the Australian Research Council (ARC) awarded to my principal supervisor.

I would like to express my heartfelt gratitude to my mother Dammika Jayanetti, my father Mahinda Jayanetti and my sister Achala Jayanetti for their endless support and love. I would also like to thank my lovely children Riley Nakandala and Deon Nakandala for inspiring me to be a better and stronger person than I could have ever imagined. Finally, and most importantly, I would like to thank my dearest husband Dulan Nakandala for being my pillar of strength, and for providing me with endless emotional support. My Ph.D. journey was extraordinary, and without you by my side, I could not have completed it.

*Amanda Jayanetti*

*Melbourne, Australia*
*August 2023*

# Contents

# List of Figures

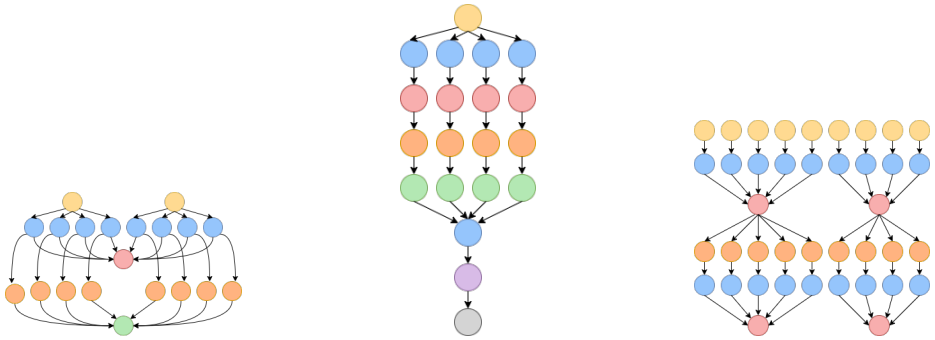
# List of Tables

# Chapter 1
# Introduction

Workflow is an application model that facilitates the representation of data in a distributed and structured manner. Therefore, the workflow application model is used for modeling the computations and data dependencies of a wide variety of applications ranging from scientific applications that are used in astronomical, biological and medicinal fields to commercial applications that are used in emerging fields such as IoT (Internet of Things). The DAG structures of some real-world scientific workflows [1] are shown in Figure 1.1.

Cloud computing has emerged as an efficient computing platform used for provisioning services over the internet. Cloud computing offers a multitude of benefits to users, including on-demand access to a pool of shared resources, rapid elasticity to suit demand variations, reduced start-up costs, elimination of infrastructure management overhead, and so on. Owing to the aforementioned benefits, cloud computing environments are widely used for the execution of workflows. Workflow scheduling in cloud computing is a well studied topic. A plethora of approaches have been proposed in academia for enhancing the performance of scientific workflow executions in cloud computing environments with respect to different optimization objectives including makespan, cost, energy-efficiency and so on.

Recently, distributed computing paradigms such as edge computing has emerged to complement the traditional cloud computing paradigm. Edge computing complements the cloud by extending computing resources to the network edge thus allowing data to be processed closer to where it is generated. While these new computing paradigms significantly enhance the capabilities of traditional cloud computing offerings with reduced latency, improved security, increased cost-effectiveness and so on, a number of



**(a)** CyberShake **(b)** Epigenomics **(c)** Inspiral

**Figure 1.1:** DAG structures of real-world scientific workflows [1]

new challenges are also introduced. In order to leverage the full potential of these technologies, adaptive and computationally efficient workflow scheduling algorithms that are capable of satisfying diverse optimization goals amid highly dynamic conditions are needed.

Reinforcement Learning (RL) is a branch of machine learning that is based on autonomous and experience-driven learning. The RL agent learns through feedback received from its interactions with the environment, and thereby improves its capabilities for making desirable decisions over time. This leads to an intelligent autonomous agent that can adapt to changing conditions. Despite the benefits of RL, the application of RL techniques to high-dimensional real-world problems has been constrained by limitations such as lack of scalability and curse of dimensionality. The emergence of deep learning led to a significant advancement in multiple areas of machine learning resulting in a dramatic improvement in tasks such as voice recognition, language processing and object detection. The combination of deep learning and reinforcement learning gave rise to the field of Deep Reinforcement Learning (DRL). DRL significantly accelerated the progress of the traditional RL paradigm through the use of deep neural networks for function approximation. The reduction of memory and computational complexities enabled the application of DRL to problems that were formerly intractable with RL.

DRL techniques have been successfully applied for solving complex decision-making and control tasks including robotics, autonomous driving, healthcare and natural language processing. The ability of DRL agents to learn from experience and utilize real-

time data for making decisions makes it an ideal candidate for dealing with the complexities associated with the problem of workflow scheduling in highly dynamic cloud and edge computing environments. Despite the benefits of DRL, there are multiple challenges associated with the application of DRL techniques including multi-objectivity, partial observability, curse of dimensionality and coordination. Off-the-shelf DRL algorithms are unlikely to be efficient at overcoming the aforementioned challenges and therefore more problem specific novel DRL techniques need to be formulated.

Accordingly, this thesis focuses on devising novel DRL techniques for addressing a set of challenges associated with scheduling workflows in cloud and edge computing environments. Firstly, we perform a baseline study on scheduling workflows in a cloud datacenter. Then we consider more complex problems associated with workflow scheduling in edge and distributed cloud environments. We propose novel single-agent as well as multi-agent DRL architectures and algorithms for efficiently addressing the unique challenges associated with the problems under consideration. Finally, we propose a reinforcement learning architecture that is generalizable to a wide range of deployments and integrate the proposed DRL technique with an industry grade workflow engine.

## 1.1 Background

In this section, a high-level overview of the main concepts related to the problem addressed in this thesis is presented.

### 1.1.1 Workflow Scheduling

A workflow can be modeled as a weighted Directed Acyclic Graph (DAG), $G = (T, E)$ where $T$ represents the set of vertices and $E$ represents the set of directed edges. Each vertex in $T$ represents a computing task, $t_n$. Each edge $e_{i,j} \in E$ represents a data dependency between tasks $t_i$ and $t_j$ such that the execution of $t_j$ cannot be commenced until the execution of $t_i$ completes. Accordingly, a precedence constraint exists between the two tasks and $t_i$ is a predecessor of $t_j$ and $t_j$ is a successor of $t_i$. 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.

The problem of scheduling workflows in a distributed system is NP-Complete in the general case. Workflow schedulers operating in distributed computing environments primarily focus on mapping tasks to nodes for execution while ensuring data dependencies are satisfied. Depending on the context of the problem the scheduling algorithm may attempt to optimize one or more performance objectives including energy, makespan, cost and QoS (Quality of Service). Scheduling algorithms can be categorised as static (offline) or dynamic (online) depending on their mode of operation. Static algorithms analyze the whole workflow and resource capacities and create a fixed schedule (i.e. task to node mapping) prior to the commencement of workflow execution, whereas dynamic algorithms operate based on information available at runtime.

### 1.1.2 Cloud and Edge Computing

Cloud services are provisioned to users through three delivery models: Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS), and users are charged for the utilisation of these services following a pay-as-you-go model. In SaaS delivery model, applications are offered to users over the internet in a fully managed manner. This eliminates the need for users to engage in downloading, installing, managing or maintaining the applications by themselves. PaaS delivery model offers a computing platform for the development of software. It relieves application developers from the burden of having to manage underlying computing infrastructures, thus allowing them to fully focus on software development activities. In IaaS delivery model, users are offered access to a pool of computing resources (servers, networking, storage etc.) that can be scaled up and down based on demand. With IaaS, users need not maintain or manage the infrastructures physically, but they are responsible for managing certain aspects such as middleware, operating systems, storage etc.

As opposed to the centralized cloud, edge computing is a fully decentralized com-

puting architecture that brings computation closer to the sources of data generation. In essence, edge computing aims at leveraging the computing, networking, and storage resources of any device (e.g. routers, switches, access points, base stations, nano data centers, etc.) that resides between the cloud and terminal devices to provision cloud-like services with minimal latency. In addition to low latency, the distributed computing architecture of edge computing allows it to facilitate real-time interactions, mobility support, location awareness, and widespread geographical coverage.

### 1.1.3 Reinforcement Learning

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 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 rewards. 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. $v_\pi(s)$ is the value function of a state, $s$ under a policy, $\pi$. It's a function for estimating the desirability of an RL agent to be in a certain state, and can be represented in terms of expected return when following a policy $\pi$ starting from state, s.

$$v_\pi(s) = E_\pi[G_t|s_t = s] \tag{1.1}$$

where, $G_t$ is the sum of discounted rewards after time, t and is expressed as:

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{1.2}$$

$\gamma$ is the discounting factor which reflects the importance of future rewards, and $\gamma \in (0, 1)$. A similar notation can be used to define the state action value function, $q_\pi(s, a)$ which represents the expected return when action, $a_t$ is taken at state, $s_t$ and policy, $\pi$ is followed thereafter.

$$q_\pi(s, a) = E_\pi[G_t | s_t = s, a_t = a] \tag{1.3}$$

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 $\theta$, 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 defined as the expected return starting from the start state of the episode, $s_0$ and thereafter following the policy $\pi_\theta$. This in fact is the value function of state, $s_0$ as expressed in Equation 1.4.

$$J(\theta) = v_{\pi_\theta}(s_0) \tag{1.4}$$

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.

$$\theta_{t+1} = \theta_t + \alpha \nabla J(\theta_t) \tag{1.5}$$

where $\alpha$ is the learning rate. From the policy gradient theorem [4], the gradient of $J(\theta)$ can be expressed in the following manner:

$$\begin{aligned}\nabla J(\theta) &= E_\pi[\sum_a q_\pi(s_t,a)\nabla\pi(a|s_t,\theta)] \\ &= E_\pi[\sum_a \pi(a|s_t,\theta)q_\pi(s_t,a)\frac{\nabla\pi(a|s_t,\theta)}{\pi(a|s_t,\theta)}] \\ &= E_\pi[q_\pi(s_t,a_t)\frac{\nabla\pi(a_t|s_t,\theta)}{\pi(a_t|s_t,\theta)}] \\ &= E_\pi[G_t\frac{\nabla\pi(a_t|s_t,\theta)}{\pi(a_t|s_t,\theta)}]\end{aligned} \tag{1.6}$$

REINFORCE [5] is a popular policy gradient algorithm which uses the above derivation for updating policy parameters via gradient ascent as shown in Equation 1.7. 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 frameworks. Figure 1.2 shows a high-level vision for a holistic DRL based scheduling framework that can be adopted in a cloud dataceneter. For the formulation of state of the environment, data from multiple sources can be collected. Real time renewable energy availability as well as predicted energy levels can be acquired from the renewable energy estimator. The network manager provides the current status of the datacenter network whereas cooling system manager provides details on thermal aspects. The utilization levels of the servers are provided by the resource manager. The state of pending workflows as well as workflows under execution is provided by the workflow analyzer. Dispatching component is responsible for managing the execution order of workflows, and for each ready task dispatched, the DRL scheduler selects the node for allocation. The selected action is executed by provisioner, and the outcome of the action is evaluated and provided as reward to the DRL model.

$$\begin{aligned}\theta_{t+1} &= \theta_t + \alpha G_t\frac{\nabla\pi(a_t|s_t,\theta_t)}{\pi(a_t|s_t,\theta_t)} \\ &= \theta_t + \alpha G_t\nabla\ln\pi(a_t|s_t,\theta)\end{aligned} \tag{1.7}$$

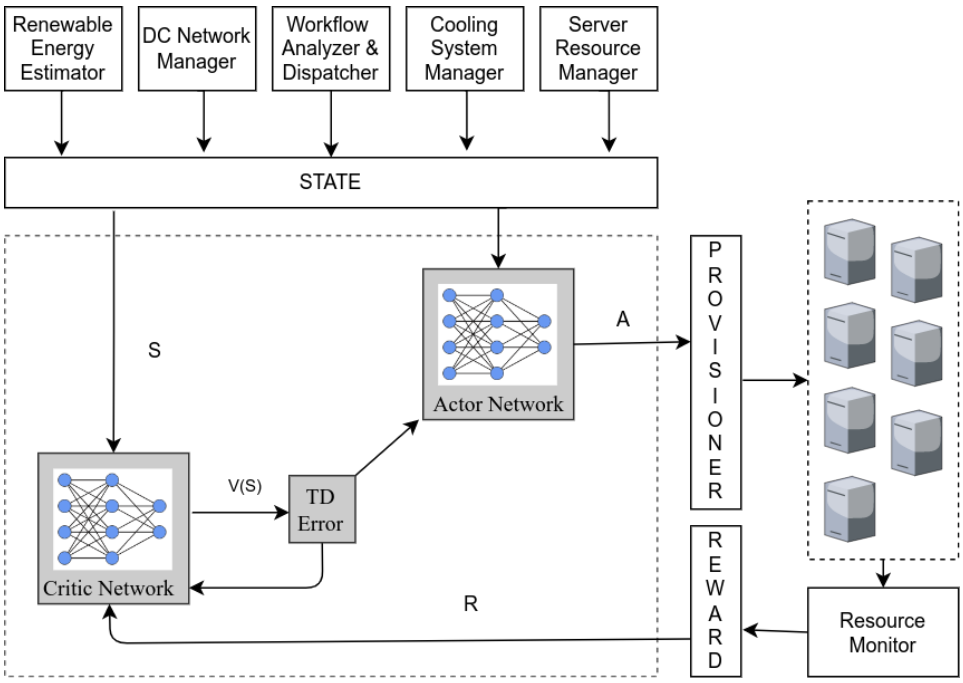



**Figure 1.2:** High level overview of workflow scheduling in a cloud datacenter with Actor-Critic method

## 1.2 Motivation

The traditional workflow scheduling problem operates with the objective of finding an allocation of tasks to execution nodes such that the precedence constraints are satisfied and the workflow completes execution within the minimum completion time. The problem of scheduling workflows in a distributed system is NP-Complete in the general case [6]. Scheduling workflows across highly dynamic cloud and edge computing environments adds a further layer of complexity atop the general problem of workflow scheduling across distributed computing environments. From a consumer point of view, more efficient workflow scheduling strategies are needed for satisfying the delay requirements of executions while also minimizing the monetary cost associated with leasing the underlying infrastructures from vendors. Whereas from a vendor perspective, efficient scheduling strategies are needed for distributing consumer workflows across their infrastructures such that the cost of operations including energy consumption of the infrastructure is minimized while also satisfying the Quality of Service (QoS) guar-

antees.

Traditional heuristic and meta-heuristic approaches to workflow scheduling have several weaknesses that limit the efficacy and applicability of these scheduling algorithms. A common drawback of heuristic techniques is that they are more prone to human errors. While the developers of heuristics may possess sufficient domain expertise to formulate efficient heuristics, these methods aren't guaranteed to be optimal. In contrast, techniques such as integer linear programming which guarantees optimal results are highly computationally complex. Existing methods also lack adaptability since they typically assume a predefined model of the operational environment. Therefore they are unable to mimic the dynamics of real-world scenarios including time variant environmental factors such as the availability of renewable energy. Another weakness of a majority of existing approaches is that they are agnostic to unique challenges of the underlying environments. For instance, as opposed to cloud servers with uninterrupted power supplies, a majority of terminal devices as well edge nodes deployed in certain locations (e.g. remote areas) tend to have limited power supplies and therefore extending the lifetime of power constrained edge nodes is imperative for preventing service disruption. Furthermore, as opposed to cloud, edge infrastructures are resource constrained and these factors need to be considered in the formulation of workflow scheduling policies. It is imperative that the scheduling algorithm is sensitive to the nature of the workflow execution environment since each environment imposes unique challenges that need to be taken into account for efficiently scheduling workflows.

Incorporating intelligence to resource allocation policies through learning techniques is an efficient way of addressing the aforementioned challenges. DRL has emerged as a promising paradigm for dealing with highly dynamic and complex problems due to the ability of DRL agents to learn to operate in stochastic environments. Well formulated DRL solutions have outperformed traditional methods in highly complex problems including robotics, and games such as Alpha-Go and Dota. DRL agents are capable of interacting with the environments where they get exposed to the real-world dynamics and thereby build an internal model which is a more accurate representation of the operating environment. This in turn leads to self-adaptability and improved decision making amid changing conditions and uncertainties.

However, straightforward application of RL algorithms in workflow scheduling problems in distributed cloud computing environments is non-trivial both due to the previously discussed challenges of the workflow scheduling problem in distributed cloud computing environments, and also due to the complexities and limitations associated with the RL paradigm in general. Some such challenges are described below.

**Curse of Dimensionality**

Curse of dimensionality refers to the problem of exponential growth in the state and action spaces when the dimensionality of the problem increases, that impedes the learning of the agent. This issue particularly bedeviled the applicability of RL algorithms to real-world problems in early stages. The emergence of deep reinforcement learning alleviated this problem to some extent. However, efficient state and action space representations are still needed for achieving faster convergence to optimal policies.

**Multi-objectivity**

Multi-objective DRL methods are inherently more complex to design as well as train compared to their single objective counterparts [7]. However, multi-objectivity is essential for efficiently capturing the diverse conflicting requirements that should be satisfied by workflow schedulers when formulating scheduling decisions in distributed cloud computing environments.

**Multi-Agent Coordination**

Single-agent scheduling frameworks maybe sufficient for satisfying the centralized scheduling requirements of the traditional cloud computing paradigm. However, such approaches are not efficient when workloads are to be scheduled across more complex distributed infrastructures such as federated clouds and emerging fog computing environments [8]. Multi-Agent Systems (MAS) in which interactions between multiple agents are leveraged, are proven to be a better fit for problem-solving in highly distributed and stochastic environments compared to its single-agent counterpart [9]. The

direct implementation of single agent RL algorithms in multiple agents leads to a non-stationary environment which prevents such methods from converging to an optimal solution [7]. Therefore the design of efficient multi-agent RL (MARL) algorithms requires overcoming multiple challenges including nonstationarity and partial observability of the environment, scalability issues that arise as the joint action space grows with the increasing number of agents, and communication between agents which is particularly challenging in partially observable environments [7].

#### Decentralized Execution

A majority of existing RL-based cloud workflow scheduling algorithms require the RL agents to be trained and executed in a centralized manner. Such centralized execution-based algorithms may not be ideal for highly distributed and stochastic computing environments [10]. However, designing and training DRL algorithms that can be executed in a decentralized manner is non-trivial due to challenges such as availability of only local information and partial intentions of other agents during execution time.

## 1.3 Evaluation Methodologies

Discrete-event simulation is one of the main methods used for evaluating the performance of algorithms in large scale experiments due to cost and management issues associated with performing such large-scale evaluations in real test beds. In this thesis an extension of the popular CloudSim [11] simulator was used for evaluating the algorithms. We have also implemented new modules for simulating the proposed workflow scheduling frameworks and interacting with the deep learning algorithms implemented using the deep learning library Keras [12]. With the use of the simulator, a series of experiments were conducted to evaluate each of the proposed techniques in a number of different scenarios. A small scale practical testbed was also used for evaluating the performance of the algorithms.

## 1.4 Research Questions and Objectives

In this thesis, we investigate the manner in which DRL techniques can be leveraged for addressing the challenges associated with scheduling workflows across distributed cloud computing environments. Initially, we perform a baseline study to investigate how energy consumption of workflow executions in a centralized datacenter can be optimized with a heuristic technique. In the subsequent works, we study more complex workflow scheduling problems, and explore how DRL techniques can be leveraged for efficiently solving those. More specifically, the following problems are investigated in this thesis:

- Q1. *How to jointly optimize host and network energy consumption of workflow executions in a centralized cloud datacenter?* Although servers are the primary source of energy consumption in cloud datacenters, datacenter networks also account for a considerable percentage. Studies have revealed that the DCN consumes 10-20% of the total data center power, and this percentage could rise much higher depending on the utilization level of the data center [13]. In this research question, we propose an energy-efficient workflow scheduling approach that jointly optimizes the power consumption of servers and networking elements in cloud data centers. The proposed technique takes into account the precedence constraints and data dependencies among workflow tasks as well as communication requirements among task instances in the formulation of topology-aware scheduling decisions.

- Q2. *How to dynamically schedule workflows across edge-cloud computing environments while satisfying complex contradictory objectives?* As opposed to cloud servers with uninterrupted power supplies, a majority of terminal devices as well fog nodes deployed in certain locations (e.g. remote areas) tend to have limited power supplies. Therefore, it is imperative to design energy-efficient resource allocation policies for workflows that are executing across cloud and fog computing environments. However, it is equally important that the optimization policies do not compromise the QoS requirements such as delay sensitivity. Accordingly, an efficient trade-off between energy consumption and execution delay needs to be established. In this research question, we propose a scheduling policy that is capable of enhancing the

energy-efficiency of workflow executions across cloud and edge computing infrastructures while also meeting workflow deadlines in a best-effort manner. We use Deep Reinforcement Learning (DRL) techniques which have proven to be efficient at handling highly dynamic and complex environments [14]. 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.

- Q3. *How to devise self-adaptable algorithms for scheduling workflows on distributed cloud datacenters while optimizing renewable energy utilization?* The use of renewable energy for powering datacenters has emerged as a promising solution for minimizing brown energy consumption. However, due to the intermittent nature of renewable energy sources, incorporating renewable energy awareness to resource allocation policies is inherently challenging [15]. The problem becomes further challenging, when the workflow executions are to be coordinated across geo-distributed cloud datacenters. Consequently, traditional heuristic and meta-heursitic based algorithms as well as single agent reinforcement learning algorithms are incapable of efficiently meeting the decentralized and adaptive control required for scheduling workflows across distributed cloud datacenters powered through renewable energy sources. In this research question, we have leveraged the recent advancements in the paradigm of MARL (Multi-Agent Reinforcement Learning) for designing and developing a multi-agent RL framework for optimizing the green energy utilization of workflow executions on distributed cloud datacenters.

- Q4. *How to devise cost-optimized workflow scheduling policies and integrate them in state-of-the-art workflow engines?* The intelligent use of a mix of on-demand and spot instances for workflow executions is a potential means of achieving high cost efficiencies without adversely affecting performance expectations. Spot instances are offered by cloud providers at steep discounts compared to their on-demand counterparts in exchange for reduced reliability. This is because the cloud providers utilize spare computing capacities available for provisioning spot instances, and

therefore when the capacity is needed back, the instances are interrupted. Furthermore, as opposed to on-demand instances with fixed prices, the prices of spot instances are not guaranteed to be fixed, as the pricing is dependent on long term supply and demand. The possibility of interruptions and pricing variations adds a layer of complexity that needs to be efficiently handled for enjoying the cost savings without compromising the underlying business requirements. In this research question, we design a DRL agent capable of establishing an efficient balance between the use of spot instances and on-demand instances for cost optimized workflow scheduling in cloud computing environments. Then, we present an end-to-end means of training and deploying the DRL agent proposed in this work in an open source container native workflow engine.

## 1.5 Thesis Contributions

A summary of contributions made by this thesis for addressing the aforementioned research problems are as follows:

1. A taxonomy on the scheduling of workflows across cloud and edge computing environments with the use of Reinforcement Learning

2. An energy-efficient workflow scheduling algorithm that jointly optimizes the power consumption of servers and networking elements in cloud data centers (addresses Q1).

3. A DRL technique for optimizing energy consumption of workflow executions across cloud and edge computing environments while meeting workflow deadlines in a best-effort manner (addresses Q2).

    - An RL framework for energy and time-optimized scheduling of precedence-constrained tasks in edge-cloud environments.
    - Energy and deadline integrated reward model for training the DRL agent to establish a desired trade-off between the conflicting objectives of energy

optimization and time minimization in workflow executions across cloud and edge computing environments.

- A novel hierarchical action space formulation. Different from existing studies in which all edge and cloud nodes are considered together in non-hierarchical action spaces, the proposed hierarchical action space promotes a clear distinction between edge and cloud nodes.
- 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 enhancing the learning process. The critic network is used to guide both actor networks.

4. A multi-agent DRL framework for optimizing the green energy utilization of workflow executions on distributed cloud datacenters (addresses Q3).

    - A hierarchical design for the workflow scheduling problem in which a global RL agent assigns tasks to datacenters and local RL agents assign tasks to nodes. Coupled with this, we present a novel formulation of the agent environment comprising of state space, action space and reward as a Partially Observable Markov Decision Process (POMDP).
    - A MARL framework capable of scheduling workflows across the the partially observable and highly distributed cloud environments in an efficient manner by sharing extra information during training, and operating solely based on local information during execution. Furthermore, we propose a shared reward structure which motivates agents to act cooperatively for achieving a common goal.
    - A novel approach to handling curse of dimensionality and thereby improving training efficiency by leveraging the hierarchical nature of distributed cloud scheduler for limiting the observations shared by agents to a local neighborhood.

5. A DRL technique for cost optimized workflow scheduling in a cloud cluster and its integration to an enterprise-grade workflow engine (addresses Q4).

    - A DRL model for cost optimized scheduling of workflows in a cloud computing environment with the use of a balanced mix of on-demand and spot instances.
    - A logical organization of the cluster in a hierarchical manner, along with a novel representation of the action selection process as a tree structure.
    - An end-to-end means of training and deploying the proposed DRL agent in a workflow engine. To the best of our knowledge, this is the first attempt at embedding an intelligent agent in an open source container-native workflow engine.

## 1.6 Thesis Organization

The structure of this thesis is shown in Figure 1.3 The remaining part of this thesis is organized as follows:

- Chapter 2 presents a taxonomy and literature review on workflow scheduling in cloud and edge computing environments with RL. This chapter is derived from:

    – **Amanda Jayanetti**, Saman Halgamuge, Rajkumar Buyya, "Reinforcement Learning based Workflow Scheduling in Cloud and Edge Computing Environments: A Taxonomy, Review and Future Directions", *ACM Computing Surveys (CSUR)*, 2023 (submitted, August 2023).

- Chapter 3 presents a joint host and network optimization algorithm for optimizing the energy-efficiency of workflow executions in cloud computing environments. This chapter is derived from:

    – **Amanda Jayanetti**, Rajkumar Buyya, "J-opt: A joint host and network optimization algorithm for energy-efficient workflow scheduling in cloud data centers", *Proceedings of the 12th IEEE/ACM International Conference on Utility*

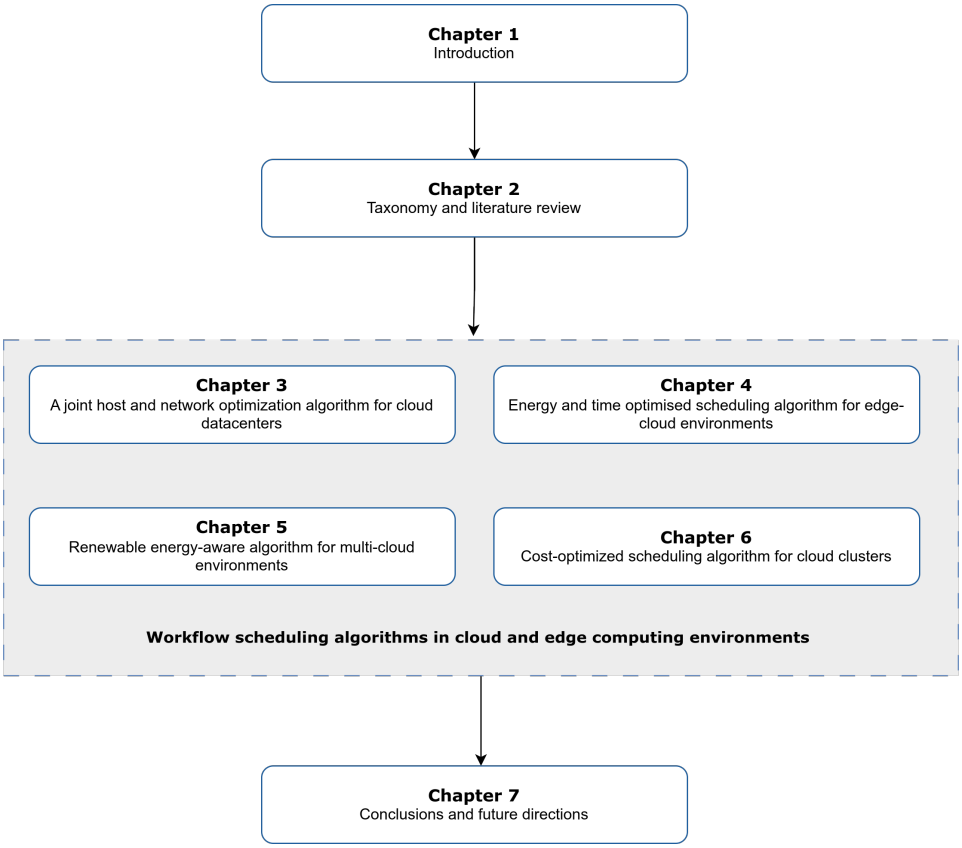



**Figure 1.3:** Thesis Organization

*and Cloud Computing*, Pages: 199–208, Auckland, New Zealand, December 2-5, 2019.

- Chapter 4 presents a technique for optimizing energy consumption of workflow executions across cloud and edge computing environments, while meeting workflow deadlines in a best effort manner with the use of DRL. This chapter is derived from:

  – **Amanda Jayanetti**, Saman Halgamuge, Rajkumar Buyya, "Deep reinforcement learning for energy and time optimized scheduling of precedence-constrained tasks in edge–cloud computing environments", *Future Generation Computer Systems*, Volume 137, Pages: 14–30, 2022

- Chapter 5 presents a multi-agent RL framework for optimizing the green energy

utilization of workflow executions on distributed cloud datacenters. This chapter is derived from:

  - **Amanda Jayanetti**, Saman Halgamuge, Rajkumar Buyya, "Multi-agent Deep Reinforcement Learning Framework for Renewable Energy aware Workflow Scheduling on Distributed Cloud Datacenters", *IEEE Transactions on Parallel and Distributed Systems*, 2024 (accepted for publication).

- Chapter 6 presents a DRL technique for optimizing cost of workflow executions, and integrates it in the open source container native Argo workflow engine.

  - **Amanda Jayanetti**, Saman Halgamuge, Rajkumar Buyya, "A Deep Reinforcement Learning Approach for Cost Optimized Workflow Scheduling in Cloud Computing Environments", *IEEE Transactions on Parallel and Distributed Systems*, 2024 (submitted, January 2024).

- Chapter 7 concludes the thesis by summarising the findings and identifies future research directions.

# Chapter 2

# A Taxonomy and Review on Workflow Scheduling with Reinforcement Learning

*Deep Reinforcement Learning (DRL) techniques have been successfully applied for solving complex decision-making and control tasks in multiple fields including robotics, autonomous driving, healthcare and natural language processing. The ability of DRL agents to learn from experience and utilize real-time data for making decisions makes it an ideal candidate for dealing with the complexities associated with the problem of workflow scheduling in highly dynamic cloud and edge computing environments. Despite the benefits of DRL, there are multiple challenges associated with the application of DRL techniques including multi-objectivity, curse of dimensionality, partial observability and multi-agent coordination. In this chapter, we comprehensively analyze the challenges and opportunities associated with the design and implementation of DRL oriented solutions for workflow scheduling in cloud and edge computing environments. Based on the identified characteristics, we propose a taxonomy of workflow scheduling with DRL. We map reviewed works with respect to the taxonomy to identify their strengths and weaknesses. Based on taxonomy driven analysis, we propose novel future research directions for the field.*

## 2.1 Introduction

A wide variety of heuristics are proposed in academia for scheduling workflows across cloud computing environments with different optimization goals. HEFT [16] is one

This chapter is derived from:

- **Amanda Jayanetti**, Saman Halgamuge, Rajkumar Buyya, "Reinforcement Learning based Workflow Scheduling in Cloud and Edge Computing Environments: A Taxonomy, Review and Future Directions", *ACM Computing Surveys (CSUR)*, 2023 (submitted, August 2023).

popular heuristic designed for makespan minimization in static workflow scheduling. This heuristic is used as a basis for task ordering in many workflow scheduling and resource provisioning algorithms. For instance, W. Zheng et. al [17] used HEFT for deriving an initial task to host mapping, post to which DVFS (Dynamic Voltage and Frequency Scaling) technique [18] was applied to selected hosts in a greedy manner. DVFS technique has been used in multiple other studies for energy-efficient workflow scheduling in cloud computing environments [19], [20]. In [21] a migration-based resource allocation policy for energy-efficient workflow scheduling across cloud VMs is proposed. Researchers have also proposed heuristics based on VM scaling [22] and task merging [23] for scheduling workflows with the objectives of enhancing energy efficiency and cost, while meeting deadline constraints.

A number of heuristics are proposed for scheduling workflows in cloud and edge environments [24], [25]. In [24] S. Ljaz et. al proposed a heuristic algorithm for energy and makespan-optimized scheduling of workflows across cloud-edge environments. Firstly, tasks ranked based on optimistic processing times are enqueued to a priority queue. For selecting nodes, a weighted bi-objective function based on minimum completion time and energy consumption is formulated, and tasks are allocated to the node with the minimum value. Deadline aware stepwise frequency scaling approach is also incorporated for further minimizing energy consumption by leveraging the unused time slots between multiple tasks scheduled on the same node.

Meta-heuristics are widely used in the design of scheduling algorithms in cloud computing environments. Particle Swarm Optimization (PSO) is a popularly used meta-heuristic in cloud workflow scheduling. For instance, in [26] PSO is used for scheduling workflows with the objective of minimizing execution costs without violating deadline constraints. Z. Li [27] also used PSO for designing a workflow scheduling algorithm for minimizing workflow execution cost while adhering to deadline and security constraints. In [28], an artificial neural network was integrated with NSGA-II meta-heuristic algorithm to develop a scheduling algorithm that is capable of adapting to the inherent dynamicity associated with cloud computing environments.

Meta-heuristics techniques are increasingly used for designing workflow scheduling algorithms across cloud and edge/fog computing environments. P. Hosseinioun

et. al [29] used an invasive weed optimization and culture evolutionary algorithm for ordering tasks with precedence constraints prior to allocating them for execution in DVFS-enabled fog nodes. The authors have leveraged slack times of tasks for setting low frequencies to processing nodes without compromising pre-defined deadlines to achieve considerable energy savings. M. Mokini et. al [30] also ordered tasks taking into account the precedence relations with the use of a Topological Sort Algorithm [31]. Afterward, Genetic Algorithm was used for scheduling tasks with the objectives of minimizing makespan and cost while optimizing resource utilization. In a work by C. Wu et. al [32] workflows are partitioned to two parts, only one of which is offloaded to the fog and cloud tiers. The other partition remains in the terminal layer in which IoT devices are residing. Partitioning is done for the purpose of minimizing data transfer among terminal and fog/cloud layers which is important for reducing energy-consumption as well as network traffic. The proposed approach uses EDA (Estimation of Distribution Algorithm) technique for scheduling the tasks among fog and cloud tiers with the objectives of minimizing energy consumption and makespan. Y. Xie et. al [33] used PSO for scheduling workflows in a hybrid cloud and edge computing environment with the objectives of minimizing execution cost and makespan. A weighted bi-objective function is used to establish the desired balance between makespan and cost. In [34], N. Bacanin et. al improved the limitations of firefly meta-heuristic algorithm by the incorporation of genetic operators and a learning prodedure based on quasi-reflection. The enhanced algorithm was then used for scheduling workflows across cloud and edge computing environments with the objectives of minimizing makespan and cost.

Despite the satisfactory results achieved by heuristic-based workflow scheduling algorithms, there are inherent limitations associated with heuristic techniques. The design of a heuristic requires advanced domain expertise and may require the collaboration of multiple experts. Regardless of the level of domain expertise possessed by the experts, the fact that heuristics are designed by humans makes them prone to human errors. Furthermore, the chosen heuristics are not guaranteed to be optimal and the selected heuristics could produce results that are far from the optimal achievable results. Although scheduling algorithms based on meta-heuristics are typically capable of overcoming the aforementioned limitations of heuristics, they are less appropriate for highly

dynamic cloud environments due to high computational costs and time complexities. Several studies have integrated meta-heuristics with popular heuristics such as HEFT for developing efficient workflow scheduling algorithms [35], [36].

Owing to the ability of RL-oriented algorithms for identifying optimal behaviors in highly dynamic and unpredictable environments, researchers are increasingly using RL for solving a wide variety of resource management problems in cloud and edge computing environments. Resource scaling is a fundamental characteristic of the cloud computing paradigm that differentiates it from previous utility computing infrastructures such as grids. The power of RL is leveraged in multiple works for designing efficient auto-scaling algorithms for cloud applications executing in traditional VMs [37] and spot instances [38], as well as more recent serverless computing environments [39]. VM scheduling and migration is another area that seems to benefit from the capabilities of RL. N. Liu et. al [40] proposed a hierarchical RL framework comprising a global tier of VM allocation and a local tier of Dynamic Power Management (DPM) [41] of servers for optimizing the energy-efficiency of the underlying cloud computing environment. RL was also used by B. Wang [42] for designing a VM scheduling and migration algorithm for cloud datacenters.

RL is increasingly becoming a popular candidate for resolving the resource management problems of upcoming edge and fog computing environments. Q learning was used in multiple works for designing load-balancing algorithms for fog computing environments [43], [44]. In [45], Deep Q Learning is used for handling mobility-aware VM migration in edge computing. Deep Q learning was also used in [46] for IoT network clustering.

## 2.2 Taxonomy

### 2.2.1 Taxonomy based on the specific problem for which RL is used

In this section, we analyse the Agent Action branch of the taxonomy shown in Figure 2.1. Reinforcement learning algorithms are used in workflow schedulers to perform many different types of actions. An action as previously discussed results in the state

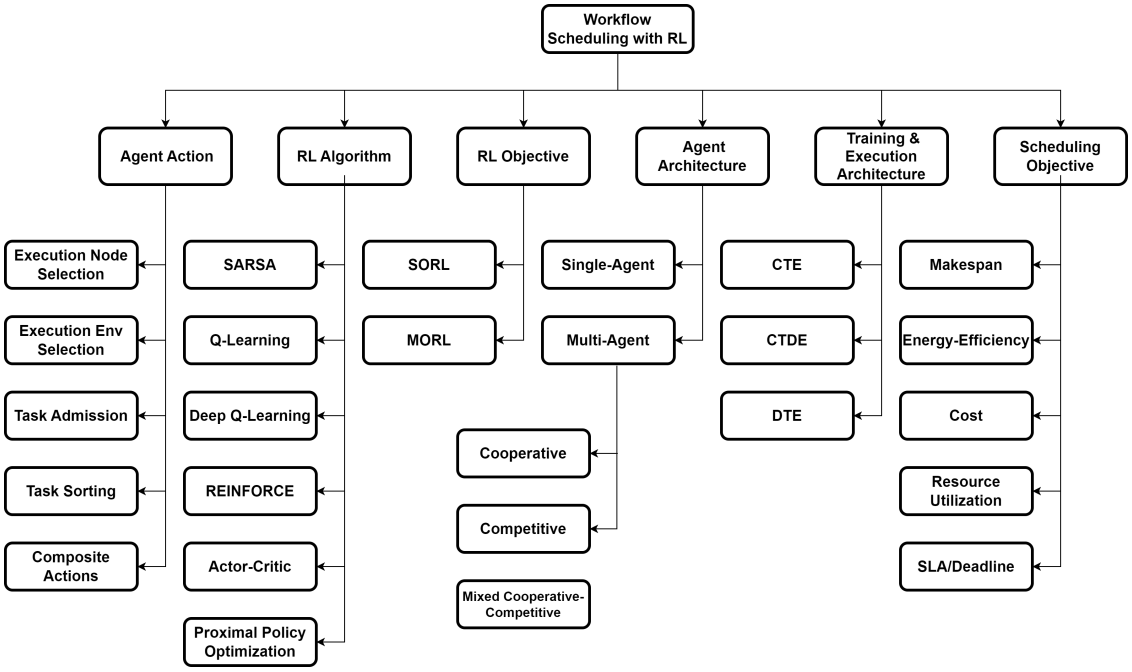



**Figure 2.1:** Taxonomy of Workflow Scheduling in Cloud and Edge Computing Environments with Reinforcement Learning [CTE - Centralized Training and Execution, CTDE - Centralized Training and Distributed Execution, DTE - Distributed Training and Execution, SORL - Single-Objective Reinforcement Learning, MORL - Multi-Objective Reinforcement Learning]

of the environment being transitioned from the current state to the next state, and the agent receives either an immediate reward or a delayed reward (usually at the end of the episode) which reflects the desirability of the action with respect to the objective of the scheduling algorithm.

An important pre-requisite for the successful application of RL in problem-solving is to identify which areas of the problem being solved would benefit from the advanced capabilities of the RL paradigm. In the case of workflow scheduling across distributed computing infrastructures, the most straightforward and common use of RL is the selection of a node for task execution, however, authors in surveyed literature have explored the power of RL in several other areas of workflow scheduling as discussed in the subsections below.

#### Execution Node Selection

In a majority of existing studies, RL is used for determining the node to which a task can be allocated for execution. Such a decision requires the consideration of multiple factors including system objective, specific resource requirements of the task which is to be allocated to the selected execution node, the current resource utilization levels of the node as well as the size of the waiting queue. A well-trained RL agent is capable of finding the right match by taking into account all the aforementioned factors and therefore, RL is heavily used for selecting the node for task executions in workflow schedulers.

The size of the action space is a parameter that influences the training speed of the algorithm as well as the decision speed [47]. Therefore, in some works, multiple actions are permitted in a single time step so that the size of the action space is constrained.

#### Execution Environment Selection

The selection of the execution environment is a high-level action for which RL is used in hierarchically designed schedulers and it is often combined with a lower-level action such as execution node selection. In schedulers that are operating across multi-cloud environments, an action of an RL agent corresponds to the selection of a particular datacenter for allocation of a task, or an entire workflow. In a cloud datacenter where execution nodes are grouped into clusters, an action would be the selection of a server cluster for task allocation. For schedulers that operate across cloud and edge computing environments, an action corresponds to the selection of the particular layer (cloud/edge) to which the task will be allocated [48].

#### Task Admission

Task admission includes the selection of a task to be scheduled. In a multi-tenant workflow scheduling system at a particular time step, there may be multiple tasks that have their precedence constraints satisfied, and therefore ready to be scheduled. In such scenarios, the selection of a task from multiple ready tasks is an important factor that impacts the resource utilization of the underlying system as well as the makespan of the

workflows that are submitted to the system. Accordingly, in order to perform the admission of a task while taking into account the system objective, resource utilization levels as well as user-defined constraints such as deadlines, the power of RL is leveraged in multiple studies. In [49], an action of the RL agent corresponds to the selection of a task to be scheduled in the cluster. The agent is also allowed to select a movement action which causes the system to run one more time-step (without admitting a new task to the system). Similar to the case in which RL is used for the selection of an execution node, in task admission as well it is important to limit the size of action space to prevent the impediment to the training process. For instance, if there are N tasks to be scheduled at a certain point, the size of the action space would be $2^N$. By allowing the agent to take multiple actions during a single time step the size of action space can be kept linear in N. This approach is adopted in [49] to prevent the expansion of action space, thus facilitating the training process. In [50], RL is used for task selection (i.e. admission) in a scenario where an RL agent is assigned to each resource, and the action of the agent is to select a task to be executed on the relevant resource.

#### Task Sorting

As opposed to independent tasks, a workflow consists of multiple tasks with complex data inter-dependencies. Depending on the structure of the workflow, completion of certain tasks may be more crucial for minimizing the makespan of the workflow. Ordering tasks of a workflow that is to be executed in a heterogeneous platform is an NP-complete problem and therefore a number of heuristics have been proposed to find approximate solutions for this problem [51]. HEFT is one such heuristic that is popularly used for task prioritization in cloud workflow schedulers [51]. HEFT is a greedy scheduling algorithm which evaluates the lengths of critical paths of tasks and ranks them accordingly, and then, the best processor is allocated to the highest ranked task yet to be scheduled. It is a static method since the workflow is processed and scheduling decisions are made offline. In QL-HEFT authors combine RL with HEFT algorithm for designing a workflow scheduling algorithm capable of minimizing makespan [52]. In the proposed approach, the upward ranks of tasks computed according to the HEFT technique are used as imme-

diate rewards for training the RL agent. A. Kaur [53] used Deep Q learning coupled with HEFT heuristic for ordering the tasks of workflows which seemed to have produced significantly better results compared to QL-HEFT. Despite improved makespan guarantees, static scheduling algorithms are not suitable for scheduling workflows across the highly dynamic cloud and edge computing environments. [54, 55] proposed online scheduling algorithms for ordering tasks of workflows using RL. In these approaches, an RL agent assigned to the workflow learns to assign values to nodes through multiple rounds of training and the values are then used for task prioritization prior to resource provisioning phase. In [55], multiple agents are assigned to parts of the workflow for speeding up the learning process involved in task prioritization. This is particularly useful when it comes to large workflows with complex inter-dependencies.

**Composite Actions**

In a majority of surveyed works, one of the aforementioned single discrete actions is the output of the RL agent. However, such action space designs are prone to the 'curse of dimensionality as the system becomes more complex and the output of the RL agent becomes multi-dimensional leading to a combinatorial increase in the number of potential actions. More advanced action space designs involving composite actions are then useful for preventing the explosion of action space leading to in-efficient learning [56]. Hierarchical RL is another means of designing RL models such that a more complex high-level problem is decomposed into multiple sub-problems thus restricting the number of discrete actions that need to be considered by lower-level agents [57]. One such work that leveraged the power of hierarchical RL is [48] in which the authors presented the design of a hierarchical action space for task allocation in edge-cloud computing environments. In the proposed hierarchical design, the upper-level agent is responsible for determining if a task should be executed in cloud or edge, and the lower-level agent then decides which node in the selected tier should be selected for task execution.

### 2.2.2 Taxonomy based on RL algorithm

In this section, we discuss the RL algorithm branch of the taxonomy shown in Figure 2.1. Reinforcement learning algorithms can be broadly classified into two categories; Model-based reinforcement learning algorithms and model-free reinforcement learning algorithms. As the name implies model-based algorithms rely on a predictive model of the environment which provides the state transition probabilities and rewards, thus enabling agents to make informed decisions and plan ahead. However, it is impractical to formulate predictive models as such for highly dynamic cloud computing environments. Accordingly, all the surveyed RL algorithms used in cloud workflow scheduling fall into the model-free category in which an agent learns completely through the experience it gathers by interacting with the environment. A number of popular RL algorithms are considered in this section of the taxonomy.

#### SARSA

SARSA is a Temporal Difference based RL algorithm that uses the following equation for learning the Q values after every state transition:

$$Q(s,a) = Q(s,a) + \alpha[R(s,a,s') + \gamma Q(s',a') - Q(s,a)] \tag{2.1}$$

Since the behavior policy is updated based on the actions taken and rewards received, SARSA is an on-policy RL algorithm.

#### Q Learning

Q learning is a popular TD (Temporal Difference) control based off-policy RL algorithm which starts with arbitrary Q values and iteratively uses the update rule in equation 2.2 for converging to the optimal Q value function ($Q^*(s,a)$).

$$Q(s,a) = Q(s,a) + \alpha[R(s,a,s') + \gamma \max_{a' \in A} Q(s',a') - Q(s,a)] \tag{2.2}$$

The optimal Q function can then be used for deriving the optimal policy as denoted

in the equation below.

$$\pi^*(a|s) = \max_{a \in A} Q^*(s, a) \tag{2.3}$$

However, Q learning relies on storing data in a tabular format which limits its suitability for complex environments with high dimensional state spaces due to space constraints associated with storing experiences as well as the in-feasibility for an agent to explore all states.

**Deep Q Learning**

DQN is a well known off-policy reinforcement learning algorithm which leverages a neural network for approximating the Q-value function. Neural network is initialized with arbitrary weights which are updated during the course of training thus allowing the network to learn the actual Q value function. This is done by iteratively minimizing the mean squared difference between network predicted Q values and target Q values as shown in equation 2.4. The parameterized Q-value function can be represented as: $Q(s, a|\theta)$.

$$\begin{aligned} L(\theta) &= \frac{1}{N} \sum_{i \in N} (Q(s_i, a_i|\theta) - (Q'(s_i, a_i|\theta))^2 \\ where\ (Q'(s_i, a_i|\theta) &= R(s_i, a_i) + \gamma \max_{a' \in A} Q(s'_i, a'_i|\theta) \end{aligned} \tag{2.4}$$

The neural network, takes as input the state of the environment ($s \in S$) and outputs the Q value of each action ($a \in A$). The action with maximum Q value can then be selected to derive the optimal policy as follows:

$$\pi^*(a|s, \theta) = \max_{a \in A} Q^*(s, a|\theta) \tag{2.5}$$

**REINFORCE**

REINFORCE [5] is a fundamental policy gradient algorithm that uses a trajectory sampled using the current policy for obtaining the cumulative discounted reward, $v_t$ which

in turn is used as an unbiased estimate of $Q(s_t, a_t)$. The policy parameters are updated with gradient ascent. Since the update takes place after a complete trajectory, REINFORCE is an off-policy algorithm. Owing to the differences between sampled trajectories, the gradient estimates could have a high variance giving rise to slow convergence. A baseline function deducted from the return is used as a means of reducing variance in gradient estimates.

#### Actor-Critic

Actor critic methods are a branch of policy gradient algorithms which combine the benefits of value based and policy based RL through the use of two interacting functions termed actor and critic. The actor takes the state of the environment as input and outputs the probability distribution of the actions to be followed for achieving a particular objective. The Critic provides an evaluation of the desirability of the action taken with respect to the set objective, and the feedback is used by actor for updating the policy parameters in the direction suggested by the critic. In essence, the actor learns the policy and critic learns the value function. Neural networks are typically used for parameterizing both functions.

#### Proximal Policy Optimization

Vanilla policy gradients suffer from sample inefficiency since a sample is only used once for updating the policy. The PPO method [58] enables performing multiple epochs of updates with mini-batches of samples through the use of a clipped surrogate objective function. Since large policy updates could lead to sub-optimal convergence, the objective function of PPO restricts the degree to which new policy is allowed to deviate from the old policy by clipping the ratio between current policy and new policy in a range, [1 - $\epsilon$, 1 + $\epsilon$].

### 2.2.3 Taxonomy based on RL objective

In this section, we analyse the RL objective branch of the taxonomy shown in Figure 2.1. Workflow scheduling problems in cloud and edge computing environments for which RL is used could have a single objective or multiple objectives that may or may not be of equal importance.

#### Single Objective RL (SORL)

A majority of RL-based workflow schedulers that have been proposed in academia belong to the single objective category. Most of the existing single-objective RL algorithms have attempted to optimize scheduling objectives such as energy, makespan, cost, SLA, and resource utilization. An in-depth discussion of these objectives is included in section 2.2.6. It is realistically impossible to map the diverse and conflicting requirements of workflow schedulers deployed in highly dynamic cloud and edge computing environments to a single objective scheduling problem. Therefore, a popular technique is to use scalarization for transforming the multi-objective scheduling problem into a single objective RL problem with the use of a weighted additive reward function and then using single-objective value-based methods such as Q-learning or policy search-based methods such as A2C for handling multiple objectives [59, 60]. Naturally, this approach leads to a single-policy solution since the underlying single-objective methods are designed to find a single optimal solution. The techniques that have used a weighted reward function, although capable of incorporating multiple objectives, do not have the same flexibility as a system with multiple objectives that includes multiple agents [61]. Determining the right weights to assign for each of the objectives requires domain expertise, and even the guesses made by experts are not guaranteed to be optimal. Furthermore, with a weighted reward function, a change in preferences cannot be accommodated flexibly. It is also costly to run multiple combinations for identifying the most appropriate distribution of weights.

#### Multi-Objective RL (MORL)

Multi-objective RL though inherently more complex to design as well as train compared to their single objective counterparts [7], may be essential for efficiently capturing the diverse conflicting requirements that should be satisfied by workflow schedulers when formulating scheduling decisions in the cloud and edge computing environments. In [61], authors argue that designing an explicitly multi-objective system from the beginning leads to the reduction of computation time and sample complexity. Particularly in the presence of highly dynamic conditions such as those in the cloud and edge computing environments, it is impractical to assume that the desired trade-offs between objectives will remain constant. RL frameworks designed explicitly in a multi-objective manner are better suited for such environments since the agents can be trained so that they can adapt to such changes, this eliminates the need to retrain the model to identify a different policy every time an external factor causes a change in objective preferences.

### 2.2.4 Taxonomy based on agent architecture

In this section, we analyse the Agent Architecture branch of the taxonomy shown in Figure 2.1. When designing an RL-oriented solution, complex scheduling problems may require coordination among multiple agents (cooperating or competing) whereas a single-agent architecture may be more suitable for other more general scenarios. Despite the advantages of multi-agent methods, they do have certain limitations such as hindrance to learning which may occur due to the actions of multiple concurrent agents making the environment non-stationary and curse of dimensionality due to exponential growth in joint state and action spaces with the number of agents [7].

#### Single-Agent Systems

As the name implies, a single agent system comprises of a single RL agent that interacts with the environment and performs an action for which a reward is received. Single-agent systems are less complex to design as well as train since they are free from the common challenges associated with multi-agent coordination described above. Despite

the success of single-agent RL methods, their applicability to a large number of real-world problems is limited since they cannot be solved with a single agent [7].

#### Multi-Agent Systems (MAS)

In multi-agent systems, multiple agents interact with the environment and learn to solve a problem concurrently. Multi-agent RL can be categorized into three groups based on whether the nature of interactions between agents is cooperative or competitive.

**Cooperative multi-agent systems** RL based schedulers of this category leverage the cooperation among multiple agents for solving parts of a larger problem. The agents may work independently on different parts of the problem or in parallel on parts of a single problem distributed among them [62]. The concurrent processing of sub-problems by multiple agents enhances the scalability of the system which is particularly important for larger problems. Also, the use of multiple agents working in parallel makes the system more robust since the system is tolerant to the failure of a single agent. For instance, in [55] both aforementioned types of multi-agent co-ordinations are leveraged to design an integrated task scheduling and resource provisioning with RL. In the task ordering phase, multiple parallel agents are used for efficiently analyzing parts of a workflow, and ordering the tasks such that tasks with higher ranks can be prioritized in the resource provisioning phase. Since different agents analyze different parts of a workflow, the search space handled by each agent is reduced, leading to faster learning. In cases where the same part of the workflow is analyzed by multiple agents, the weighted average of the Q value estimates of agents is taken. The use of parallel agents in this manner has been particularly useful for speeding up the learning process especially when workflows with a large number of complex interdependencies are involved. In the next phase of the same work, multi-agent coordination is again leveraged for provisioning resources to the ordered tasks. In this case, an agent is assigned to each resource, and the action is the selection of a task to be executed on the resource. Each agent operates with the objective of maximizing its reward, so to ensure the global objective of system-wide cost and energy minimization is achieved, a Markov game is formulated to converge the

agents to a globally optimal solution.

**Competitive multi-agent systems** Competitive multi-agent systems are generally modelled as zero sum Markov games where the sum of returns of all agents is zero [63]. In the common case where there are two competing agents [64], the reward received by one agent is equivalent to the loss of the other.

**Mixed cooperative-competitive multi-agent systems** Mixed settings are characterized by a combination of features from competitive and cooperative settings and includes a general-sum reward [7]. In mixed settings agents are self-motivated and may have rewards that are conflicting with those of others [63].

### 2.2.5 Taxonomy based on the RL agent training and execution architecture

In this section, we analyse the Training and Execution Architecture branch of the taxonomy shown in Figure 2.1. Multi-agent coordination in RL is an active research area, therefore the categorizations considered in this part of the taxonomy are not exhaustive. We have limited the scope of analysis to three main types of training and execution methods although other hybrid variants are recently being explored for multi-agent coordination [65].

#### Centralized Training and Execution

A majority of RL-based algorithms considered in this survey belong to the category of centralized training and centralized execution as they are straightforward single-agent algorithms. As the name implies, in the case of a single-agent scenario, the agent is trained and executed in a centralized manner. In multi-agent scenarios, this type of learning is typically based on a joint action and observation model in which a joint observation of all agents is mapped to a joint action by a centralized policy. This in turn could lead to an exponential growth in the action and observation spaces with the number of agents [66]. The multi-agent workflow scheduling algorithm proposed in [48] leverages hierarchical RL [57] for addressing the aforementioned problem. In the proposed approach, the action space is designed in a hierarchical manner such that the action spaces of agents at each level are confined.

#### Centralized Training and Distributed Execution

Despite the promising experimental results, centralized execution-based RL algorithms may not be ideal for highly distributed and stochastic edge computing environments [10]. In centralized training and distributed execution (CTDE) paradigm, additional information that is only available at training time is leveraged to learn decentralized policies such that during execution time the agents operate solely based on local observations and partial information about the intentions of other agents, thus alleviating the need for complex communications between agents [67]. The paradigm of CTDE is utilized in [68] for coordinating multi-agent training and execution in the scheduling of concurrent requests modeled as DAGs across edge networks. A policy learning technique based on value decomposition networks [69] is adopted for decomposing the value function of the team into individual value functions. This helps overcome problems associated with independent learning such as when the learning of a particular agent is discouraged since its exploration leads to the hindrance of another agent that has already learned a useful policy [69].

#### Distributed Training and Execution

As the name implies, in the paradigm of distributed training and execution, both learning as well as execution takes place in a decentralized manner. Popular Multi-Agent RL algorithms such as A3C [70] and more recent developments such as IMPALA [71] as well as algorithms such as Gorila (General Reinforcement Learning Architecture) [72] that leveraged parallel computations to enhance the efficiency of single-agent training falls into this category.

### 2.2.6 Taxonomy based on scheduling objective

In this section, we analyse the Scheduling Objective branch of the taxonomy shown in Figure 2.1.

### Makespan

Makespan minimization is one of the most commonly studied objectives in workflow scheduling across distributed computing infrastructures. Makespan refers to the total time it takes for a workflow to complete execution, and it is dependent on multiple factors including the size of tasks, processing speeds of nodes to which tasks are allocated for execution, the volume of data dependencies among tasks, and bandwidth of underlying networking infrastructures. Makespan minimization across heterogeneous and distributed infrastructures is a complex problem, it is even more complicated when workflows are to be scheduled across multi-tenant cloud computing environments [73]. Several RL-oriented methods have been proposed by researchers for solving this problem efficiently. One such approach is to use RL for sorting tasks of workflows prior to resource provisioning, another more common method is to incorporate makespan minimization objective to the reward of an RL agent so that resources are allocated to tasks with the long term objective of minimizing the overall makespan.

### Energy efficiency

The ever-increasing popularity of the cloud computing paradigm has resulted in the development of hyper-scale data centers that consume significantly high levels of energy. High energy consumption leads to undesirable consequences including raised levels of CO2 emissions and increased monetary costs associated with high electricity consumption [74]. Energy efficiency is equally important for the emerging edge computing paradigm since a majority of edge devices are likely to be powered with batteries with limited capacities and the collective energy consumption of rapidly growing volumes of edge nodes is estimated to be significantly high [75]. Therefore, a plethora of research efforts has been focused on enhancing the energy efficiency of cloud and edge computing infrastructures through a variety of different mechanisms including dynamic power management [41], thermal aware scheduling [76] and renewable energy utilization [15]. Accordingly, energy efficiency has been considered a primary objective in a large number of workflow scheduling algorithms proposed in academia [77]. More recently, researchers have leveraged the advanced capabilities of RL for enhancing the

energy efficiency of workflow executions across distributed cloud computing environments.

#### Cost

A majority of cloud services are billed using the pay-as-you-go approach. Cloud computing platforms offer virtual machines (VM) with different flavors (processing capabilities including the number of virtual CPUs, memory, and storage) and therefore different prices are charged for their utilization. Execution cost depends on the total number of VMs used for the execution of workflow tasks and their respective flavors, which is determined by the underlying resource provisioning and scheduling policy. In addition to minimizing overall cost as a primary objective, some studies have attempted to satisfy budget constraints in a best-effort manner, while primarily optimizing different objectives such as makespan and/or energy [78].

#### Resource Utilization

Optimized resource utilization is an intrinsic objective that goes in hand with many other objectives. For instance, optimized network and host (virtual or physical) resources lead to lower energy-efficiency, since unused resources can be put into dormant state. This in turn reduces cost of operations as well.

#### SLA/Deadline

Deadline is an important objective particularly when it comes to delay sensitive workflows such as those used in IoT (Internet of Things). In some studies deadlines are also used for establishing an upper bound to the degree to which makespan of workflows are allowed to extend for achieving objectives such as energy-efficiency [48].

**Table 2.1:** Analysis of existing literature based on proposed taxonomy for workflow scheduling with DRL [CTE - Centralized Training and Execution, CTDE - Centralized Training and Distributed Execution, DTE - Distributed Training and Execution, MORL - Multi-Objective Reinforcement Learning, SORL - Single-Objective Reinforcement Learning]

| Work | Agent Action | RL Algorithm | RL Objective | Agent Architecture | Training-Execution Architecture | Scheduling Objectives |
|---|---|---|---|---|---|---|
| A. Asghari et al [55] | Task Sorting<br>Task Admission | Q Learning | MORL | Multi-Agent Cooperative | CTDE | Makespan, Energy-Efficiency<br>Cost, Resource Utilization |
| Y. Wang et al [79] | Execution Node Selection | Deep Q Learning | MORL | Multi-Agent Cooperative | DTE | Makespan, Cost |
| A. Asghari et al [50] | Task Sorting<br>Task Admission | SARSA | MORL | Multi-Agent Cooperative | CTDE | Resource Utilization, Makespan |
| Y. Qin [78] | Execution Node Selection | Q Learning | MORL | Single-Agent | CTE | Makespan, Energy-Efficiency,<br>Cost |
| A. Nascimento et al [80] | Execution Node Selection | Q Learning | SORL | Single-Agent | CTE | Makespan |
| A. Asghari et al [54] | Task Sorting<br>Task Admission | Q Learning | MORL | Multi-Agent Cooperative | CTDE | Makespan, Energy-Efficiency,<br>Resource Utilization |
| Z. Tong et al [52] | Task Sorting | Q Learning | SORL | Single-Agent | CTE | Makespan |
| A. Kintsakis et al [81] | Execution Node Selection | REINFORCE | SORL | Single-Agent | CTE | Makespan |
| T.Dong et al [82] | Execution Node Selection | Deep Q Learning | SORL | Single-Agent | CTE | Makespan |
| Z. Pheng et al [83] | Execution Node Selection | Deep Q Learning | MORL | Single-Agent | CTE | Energy-Efficiency, Makespan |
| A. Orehan et al [84] | Execution Node Selection | Q Learning, SARSA | SORL | Single-Agent | CTE | Makespan |
| Q. Wu et al [85] | Execution Node Selection | REINFORCE | SORL | Single-Agent | CTE | Makespan |
| Y. Hu et al. [49] | Task Admission | Actor-Critic | SORL | Single-Agent | CTE | Makespan |
| H. Li et al. [86] | Composite Action<br>(Task Admission, Execution Node Selection) | Double Deep Q Learning | MORL | Multi-Agent Cooperative | CTE | Makespan, Cost |
| F. Hue et al [87] | Execution Node Selection | Deep Q Learning | SORL | Single-Agent | CTE | Makespan |
| Y. Zhang et al. [68] | Task Admission | Deep Q Learning | MORL | Multi-Agent Cooperative | CTDE | Energy, Delay,<br>Throughput |
| Y. Zhang et al. [88] | Execution Node Selection | TD-Learning | MORL | Single-Agent | CTE | Energy-Efficiency, Delay |
| Z. Hu [**?** ] | Task Sorting | Deep Neural Networks | SORL | Single-Agent | CTE | Makespan |
| Z. Tong [89] | Execution Node Selection | Deep Q Learning | SORL | Single-Agent | CTE | Makespan |
| A. Jayanetti et al. [48] | Execution Env Selection<br>Execution Node Selection | Proximal Policy Optimization | MORL | Multi-Agent Cooperative | CTDE | Makespan, Energy-Efficiency |

## 2.3 Review of Reinforcement Learning based workflow scheduling techniques

In this section, we review the RL-based workflow scheduling techniques in the context of the taxonomy presented in the previous section.

In [79] the problem of scheduling workflows in cloud computing is handled with the optimization objectives of minimizing makespan and cost. For this, the authors have proposed a multi-agent deep reinforcement learning framework. The multi-agent collaboration is modeled as a Markov game with a correlated equilibrium. As opposed to Nash equilibrium, in a correlated equilibrium, the agents are not motivated to deviate from the 'joint distribution in a unilateral manner' hence they are capable of collaborating together to optimize different objectives. Each agent attempts to optimize one of the scheduling objectives. It is assumed that the actions and rewards of agents are visible to other agents. The state of the system includes VMs that are available for execution currently, and the successors of tasks that have been scheduled for execution in the previous state. The action space consists of the probabilities of a task to be allocated to a particular VM. The reward of the makespan agent and the cost agent is designed to encourage the minimization of makespan and cost, respectively.

A DRL-based task scheduling and resource provisioning framework for workflow execution in the cloud is proposed in [50]. In the task scheduling step, an agent is assigned to each workflow for evaluating the values of nodes after a number of episodes. In this case, each node of the workflow is a state and an action is the selection of the next node (successor). A reward that depends on the computation cost of the next node and the communication cost between the current node and the next node is received by the agent and the values in the Q table are updated accordingly. The agent uses SARSA algorithm and after a number of iterations, the maximum cost from the start node to all other nodes is obtained. The tasks are then sorted in ascending order of their start times. In the resource provisioning phase, an agent is assigned to each resource. A state in this case is the sequence of tasks that was the result of the previous phase. An action corresponds to the selection of a task to be executed on the relevant resource. The reward received by the agent is the resource utilization of the resource post to the allocation

of the selected task to the resource. Since the agents are operating independently, this method formulates the reward in a manner such that long-term resource utilization is optimized rather than sub-optimal greedy allocations. To ensure model convergence, and the selection of a globally optimal solution GA is used. Accordingly, the assignment of tasks to a resource is represented as a chromosome. Tradition GA operators (crossover, mutation, etc.) are then used for achieving model convergence. The results of a comparison study on a simple use case clearly demonstrate the superiority of using cooperative agents over the random selection and independent agents.

An end-to-end RL-oriented framework for resource provisioning and scheduling workflows in cloud computing environments is proposed in [55]. The first RL model is for ordering the execution order of tasks in a workflow such that the scheduling algorithm will pick tasks for scheduling from the ordered list. For this, a multi-agent RL framework is used, where multiple agents are assigned to a workflow, thereby reducing the search space of the problem for faster convergence. A state of the environment is a node of the workflow, and the state space comprises the set of all nodes of a workflow. An action includes selecting a child node and the reward is the sum of the current execution time of the node and the data transfer time to the selected child node. Where multiple agents have traversed through the same path, the value of a node is the average value of all the agents. In the resource provisioning phase, tasks are assigned to a cluster of resources. Again RL is used to determine which tasks are assigned to which resources in the cluster. In this model, a markov game is formulated where each agent is a resource and each action corresponds to selecting a task to be executed on the resource. A local reward based on the resulting allocation's ability to meet task SLA and the state of the resource (normal, overloaded, under-loaded) post to the assignment is given to each agent. Virtual machine migration and DVFS techniques are is incorporated into the resource provisioning scheme to further enhance the energy efficiency of the system.

In [78] authors aim to minimize the makespan and energy consumption of workflows within a budget constraint. Tasks in a workflow are sorted based on a priority value calculated considering the task execution time and communication dependencies. The sorted tasks are then scheduled using Q learning algorithm. The agent environment considers VM utilization at each time step as the current state and an action corresponds

to the selection of a VM for task execution. A budget constraint is imposed on the action space to limit available actions at each time step. A multi-vector reward in which one vector is the ratio of fastest and actual finish times of task and the other vector is the ratio of least and actual energy consumption of task execution is received by the agent. Since the reward consists of two vectors (a weight selection problem arises), this work uses the Chebyshev scalarization function to secularise the Q values of state-action pairs and then selects the smallest scalarized Q value in a greedy manner. At the end of each episode, the corresponding solution is added to the Pareto set if it isn't dominated by any other solutions, and all solutions that are dominated by it are removed.

In [80], a unique approach is proposed where Q learning is used to schedule tasks which are referred to as activations, to VMs with the objective of minimizing the makespan. However, different from existing works, in this approach, an episode corresponds to scheduling the activations of a single workflow. Hence the states are limited to available, unavailable, successfully finished, and terminated with failure. The action space only comprises of two actions which include scheduling an activation to a VM or doing nothing. A reward based on the performance of an assignment of activation to a VM compared to the overall performance of the workflow is used for promoting actions that improve the efficiency of the workflow.

The proposed DRL framework in [54] is somewhat similar to that of[50]. The main difference is that work used SARSA whereas this used Q learning. Initially, DRL agents are assigned 'parts' of workflows, and they traverse the workflow to find the cost from each node to the sink node. The agents are rewarded based on the sum of the computation cost of the next node and the communication cost for sending data between the nodes. The sorted tasks are then allocated to resources using multiple RL agents each of which is assigned to a resource, and the agents operate with the objective of improving long-term resource utilization. If the task deadline is exceeded in a given resource, then the corresponding agent receives a large penalty thus encouraging agents to select tasks that leads to better resource utilization while also meeting deadlines. The reward for achieving the aforementioned goals consists of two components; the frequency of the processor to which the task is assigned and the negative value of the remaining unused frequency of the resource post to task allocation. The two components are combined

through a normalized weight factor which can be used for adjusting the priority according to system requirements. As an added advantage of the increased resource utilization, higher energy savings are also achieved.

In [52], a static task scheduling algorithm is proposed. It uses a combination of Q learning together with the popular HEFT algorithm for obtaining optimal task ordering. For each workflow, an entry task is randomly chosen which is considered the current task (current state), one of the previously unselected successor tasks is then selected and an immediate reward is calculated which is equivalent to the upward rank proposed in the HEFT algorithm. Q table is updated accordingly. The selected successor then becomes the next state. The process repeats until the Q table converges. The final Q table is then used to obtain the optimal task order. For the execution of a task, the processor which is capable of completing the task earliest is selected.

In [81], supervised learning techniques are used for predicting the probability of failure and runtime estimations of tasks at different execution sites. These predictions coupled with the cost of communicating input data to a particular site and the number of task successors are formulated as a feature vector. For each of the ready tasks, and for each of the available execution sites, such vectors are constructed and all of these together form the input sequence. An action would be the assignment of ready tasks to execution sites, and the work assumes that a task can be assigned to one site and a site can only execute one task at a time. The size of the action space would then be equivalent to the number of execution sites that are capable of executing a task. For handling the input and output sequences which are of variable size a pointer network is used in this work. The size of action space is reduced by formulating the output of the pointer network to select one task to execution site allocation at a time. The model is executed until all ready tasks are assigned to sites for execution. The reward is equivalent to the negative workflow makespan and from this, a baseline of the average execution time of a workflow based on past observation is deducted to stabilize the training process. The proposed model is trained in a simulated environment and deployed and tested in a practical environment.

[82] presents a straightforward application of DQN for scheduling tasks with precedence relations in a cloud manufacturing environment. Tasks are sorted prior to the use

of DQN for scheduling using upward ranks. State space comprises the server workloads and tasks allocated to servers and the tasks' start and finish times. The action space consists of all the servers to which a task can be allocated and the makespan difference between the current and next state forms the reward.

The work proposed in [83] uses DQN with a weighted reward function for establishing a desired tradeoff between energy consumption and makespan in scheduling tasks with precedence relations. More specifically, the makespan component of the reward is the inverse of the total wait and execution times of a node at the selected server and the energy component is the difference in energy consumption between the current and previous time steps. Min-max normalization was used to normalize the two components prior to their application of them in the weighted reward function. The state space comprises the number of VMs available in servers and the waiting time at each server for a task to be deployed, the set of servers available for task execution forms the action space.

In [84], authors proposed an RL framework for scheduling workflows in distributed computing environments. A multi-threaded java based pluggable scheduling module is presented such that multiple clients can be served by leveraging the parallel processing capabilities. The authors have implemented Q-learning and SARSA algorithms in the presented module. The scheduling environment is designed such that the state space consists of the load level of server queues defined in relation to a precision percentage, and an indication of whether a predecessor or a sibling of the task already resides in a particular server queue. An episodic reward of total execution time in comparison to a base value is awarded to the agent at the terminal state. As opposed to most workflow scheduling papers which have simply failed to include an indication of the node in which a particular task's predecessors and successors are residing, this work has incorporated that information into the state space, and that in turn enables the agent to learn to make better allocations such that the resulting allocations result in lower makespans due to minimized communication times and improved parallel executions. Since this work has used, Q-learning and SARSA it is important to prevent the expansion of state space, and that is achieved through the use of a precision percentage to indicate the number of tasks in a queue and also, rather than including the specific characteristics of

the task to be scheduled, a task classifier is used to assign a task type to each task. This type of state space discretization although lowers accuracy is crucial when algorithms such as Q-learning and SARSA are used.

[85] uses REINFORCE algorithm to schedule precedence-constrained tasks in distributed computing environments. To formulate the state space in a compact manner, they have incoporated the earliest start time of a task in each of the available servers. The earliest start time serves as an indication of both the load on the processor as well as the cost of communication. Additionally, the number of tasks that are to be scheduled is also included in the state. Where multiple tasks are ready to be scheduled, upward ranks are used for prioritizing the selection of tasks. An immediate reward of the increase in schedule length after taking the current action compared to the previous schedule length is used.

The authors of [49] use images for state-space representation, similar to the popular job scheduling framework presented in [14]. The resource availability and usage of the cluster together with resource requirements of the tasks to be scheduled and the number of scheduled tasks are included in the state. In order to include inter-task dependencies in the state, such that the agent is capable of learning better, a critical path-based technique is proposed. In the proposed approach, a workflow is divided into multiple stages and processed with a depth-first search. Using the critical path information computed during the division process, a stage number matrix and a critical path matrix are computed for each task and these matrices are also included in state representation. An action corresponds to the selection of a task to be scheduled in the cluster and in one time step multiple such selections are permitted to prevent the action space from being too large. The agent is also allowed to select a movement action that causes the system to run one more time-step (without admitting a new task to the system) An episodic reward equivalent to N/makespan where N is the number of tasks scheduled is awarded to the agent.

In [86], authors propose a multi-level multi-agent reinforcement learning framework for scheduling workflows in cloud computing environments with the objectives of minimizing makespan and cost. Furthermore, the paper also presents a mechanism for adjusting the models' attention to each objective as preferred by users during the training

process so that the diversity of the resulting solutions is enhanced. The state space includes the features of ready tasks, VMs available as well as corresponding time and cost of executions. In the multi-level scheduling strategy, first, a task is selected from amongst the ready tasks and then a VM is selected for executing the selected task. Since the number of ready tasks is variable, a pointer network is used in the first level so that an input state with variable length can be handled. The second level uses a traditional deep neural network. For each of the objectives, a separate sub-agent each with a separate reward is used at each level. The work also proposes the use of a normalized weight factor for combining the probabilities of selecting a candidate action for time and cost agents at each level (task selection and VM selection). The authors have also used a weighted double-deep Q learning network with a dynamic coefficient. Double deep Q learning (DDQN) method reduces overestimation of action values (associated with traditional deep Q learning method) and weighted double deep Q learning (WDDQN) further improves accuracy by reducing the underestimation bias associated with DDQN. To establish a desirable balance between over-estimation and underestimation issues associated with DQN and DDQN respectively, a dynamic coefficient that can be changed during the training process according to the two types of errors in estimations is proposed.

The authors of [87] combined DQN and genetic algorithm to design a scheduling algorithm with high convergence speed in an edge computing environment. More specifically, DQN is used to generate the initial population of the Genetic Algorithm and this in turn improves the convergence speed of the algorithm by eliminating the randomness of the initial population. The state space of the DQN model comprises the start and finish times of tasks in edge servers, cost of communication, and computations. An action corresponds to the selection of a server and the reward is the difference between makespan in the current and next states.

In [68], a multi-agent DRL framework for scheduling DAG-based user requests in edge computing environments is proposed. In order to handle the problem of non-stationary environments that occurs when independent DQN agents are operating in an environment, a value decomposition network coupled with Centralized Training and Distributed Execution (CTDE) training method is used. Accordingly, the authors use

the linear summation of individual Q values of agents to derive the team Q value. As opposed to more complex ways of computing the team Q value with techniques such as neural network fitting, a linear summation simplifies implementation and also provides an intuitive evaluation of each agent's contribution to the global objective. Furthermore, to avoid the problem associated with non-stationary environments, the authors have trained the model in a Centralized Training and Distributed Execution manner (CTDE). Accordingly, all agent training tuples are stored in a shared replay buffer from which random samples are selected and used for training all agents through the replay. The state of each agent comprises of resource capacity of the node, communication cost to other nodes, and the details of ready tasks. Action is the joint actions of all edge nodes, and at each epoch, only the first N tasks are selected (one edge node can only select one task). The reward is the weighted sum of the average delay and energy consumption of edge nodes and a penalty that is dependent on the number of tasks that exceeded the temporal dependence.

## 2.4 Summary

Reinforcement Learning has emerged as a promising paradigm for dealing with highly dynamic and complex problems due to the ability of reinforcement learning agents to learn to operate in stochastic environments. More recently, well formulated deep reinforcement learning solutions have outperformed traditional methods in highly complex problems including robotics, and games such as Alpha-Go and Dota. Reinforcement learning agents are capable of interacting with the environments where they get exposed to the real-world dynamics and thereby build an internal model which is a more accurate representation of the operating environment. This in turn leads to self-adaptability and improved decision making amid changing conditions and uncertainties. Despite the benefits, there are multiple challenges associated with the application of reinforcement learning techniques including multi-objectivity, curse of dimensionality, scalability and coordination. Off-the-shelf algorithms are unlikely to be efficient at overcoming the aforementioned challenges and therefore more problem specific novel reinforcement learning techniques need to be formulated.

In this chapter, we reviewed the state-of-the art of workflow scheduling algorithms with reinforcement learning in cloud and edge computing environments. Based on the analysis we identified the merits and weaknesses of existing works, and potential areas of improvements along with some of the latest developments in the field of reinforcement learning that can be pursued by the research community.

# Chapter 3

# Joint Host and Network Optimization Algorithm for Workflow Scheduling

*Existing approaches to energy-efficient workflow scheduling in cloud computing environments have primarily focused on the optimization of server utilization. The majority of works have ignored the impact of scheduling decisions on the data center network (DCN). However, studies have revealed that the DCN consumes 10-20% of the total data center power, and this percentage could rise much higher depending on the utilization level of the data center. This chapter proposes an energy-efficient workflow scheduling approach (J-OPT) that jointly optimizes the power consumption of servers and networking elements in cloud data centers. J-OPT considers precedence constraints and data dependencies among workflow tasks as well as communication requirements among task instances in the formulation of topology-aware scheduling decisions. The proposed approach is evaluated using synthetic and real world workflow traces in a simulated environment. Results of the experiments demonstrate that J-OPT outperforms state-of-the-art algorithms in terms of total power savings by 8% and 30% under high and low data center utilization levels, respectively.*

## 3.1 Introduction

While servers are the major source of power consumption in data centers, data center networks (DCNs) also account for 10%-20% of total power consumption. This percentage could rise as high as 50% in data centers with energy-proportional servers, under light job loading conditions [13]. Therefore, energy consumed by datacenter networks

This chapter is derived from:

- **Amanda Jayanetti**, Rajkumar Buyya, "J-opt: A joint host and network optimization algorithm for energy-efficient workflow scheduling in cloud data centers", *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing*, Pages: 199–208, Auckland, New Zealand, December 2-5, 2019.

and networking devices that facilitate communications among hundreds of thousands of concurrently executing instances is a non-trivial factor that contributes to increasing the overall energy consumption of data centers considerably. Furthermore, over-utilized network devices lead to the creation of congestion hotspots resulting in undesirable packet losses, and imbalanced use of network links reduces the overall utilization of the data center networks. Scheduling algorithms that are agnostic to the communication patterns of underlying workloads are unlikely to be efficient at exploiting the power savings that can be achieved by the joint optimization of compute and networking elements in cloud data centers.

Not only scientific workflows but also a wide variety of batch workloads and distributed applications can be modeled using the DAG (Directed Acyclic Graph) execution model [90]. Although a large body of literature on workflow scheduling techniques in cloud environments exist, only a minor proportion have considered energy efficiency as a primary objective.

A significant majority of historically proposed WaaS (Workflow as a Service) schedulers primarily focus on coarse-grain assignment of workflow tasks to virtual machines, and the assignment of virtual machines to physical servers [21], [91]. Several studies have adopted container virtualization [92], [93]. In most of these approaches multiple containers are grouped together on virtual machines which in turn are assigned to physical nodes. This is the predominantly used model in conventional public cloud infrastructures due to security concerns associated with allowing containers to directly execute on bare metal nodes.

With the invention of lightweight micro virtual machine (micro VM) based technologies such as AWS Firecracker [94], cloud platforms are able to enjoy the best of both worlds as micro VMs offer the speed and resource efficiency of containers while simultaneously provisioning the security and workload isolation of virtual machines. Inspired by the aforementioned developments, the architecture proposed in this work models a system in which each task executes in a micro VM.

Motivated by the aforementioned opportunities, we propose a topology-aware scheduling algorithm that jointly optimizes the utilization of computing and networking ele-

ments for energy-efficient scheduling of workflows in multi-tenant public cloud platforms. To the best of our knowledge, this is the first work to perform a thorough disaggregated analysis of the power consumption behavior of workflow executions in cloud computing environments.

The rest of the chapter is organized as follows: In section 3.2, we review the literature on state-of-the-art approaches related to the scope of this chapter. In section 3.3, we formulate the power model and energy optimization problem. In section 3.4, we present the proposed algorithm. Followed by this, we present the performance evaluation of the algorithm in section 3.5. Finally, in section 3.6, we conclude the chapter with a summary of contributions.

## 3.2 Related Work

The problem of workflow scheduling in cloud computing environments has been extensively studied in a large number of research studies [95]. Hence, we have limited the scope of this literature review to focus on a set of selected workflow scheduling algorithms that consider energy efficiency as a primary objective. We have also reviewed a number of joint host and network optimization algorithms which have not been oriented towards a specific application type.

### 3.2.1 Energy-Efficient Workflow Scheduling

A number of studies have proposed meta-heuristic based techniques for scheduling workflows in an energy-efficient manner in cloud infrastructures [96], [97], [98], [99], [100].

For instance, multi-objective Particle Swarm Optimization (PSO) has been used in a study by S. Yassa et. al. [100] to schedule workflows considering the objectives of minimizing cost, execution time and energy consumption. K. Bousselmi et. al. [96] used a workflow partitioning algorithm coupled with Cat Swarm Optimization (CSO) to minimize the energy consumption and makespan of workflows. The proposed approach aims to minimize network energy by partitioning the workflows such that the amount

of data transferred between the workflow partitions is minimized. CSO is then used for scheduling the generated partitions.

Although scheduling algorithms based on meta-heuristics are capable of attaining better solutions compared to list based and clustering based algorithms, they are less appropriate for highly dynamic cloud environments due to high computational costs and time complexities.

A number of heuristic algorithms have also been proposed for energy-efficient workflow scheduling [21], [101], [102], [103], [19], [104], [21], [17], [105]. Zotkiewicz et. al. [105] presented an energy and communication aware scheduling strategy for SaaS (Software as a Service) applications in a cloud data center by modeling the applications as dynamic workflows. The proposed scheduling strategy operates with the objectives of minimizing energy consumption and average makespan of all submitted workflows. Inter-task communication aspects are considered in the proposed approach by incoporating network awareness as a secondary condition used in the event of a tie between multiple equally energy efficient servers.

X. Qu et. al. [102] proposed an energy-efficient scheduling heuristic for data-intensive workflows considering a cloud environment in which computation and storage are disaggregated. The authors describe how factors such as bandwidth and speed of network connections impact the total data transferring time, and thereby increase the processing time and energy-consumption of data-intensive workflows. The proposed algorithm operates in two phases; VM deployment and task-VM mapping. In the stage of VM deployment, compute nodes and storage nodes among which data can be transferred with minimal energy consumption, are selected for VM deployment and data storage, respectively. Afterward, tasks are mapped to VMs based on ranks assigned to VMs as well as tasks for minimizing energy consumption.

In a study by X. Xu et. al. [21] an energy aware resource allocation has been presented in which task requests are always allocated to the host with lowest baseline energy consumption. This study relies on the assumption that all the instances of a task should be scheduled on the same physical server to minimize the cost of inter-task communication. While, this approach would incur gains when all instances can be accommodated on a single server, it will fail to support workflows with tasks which contains

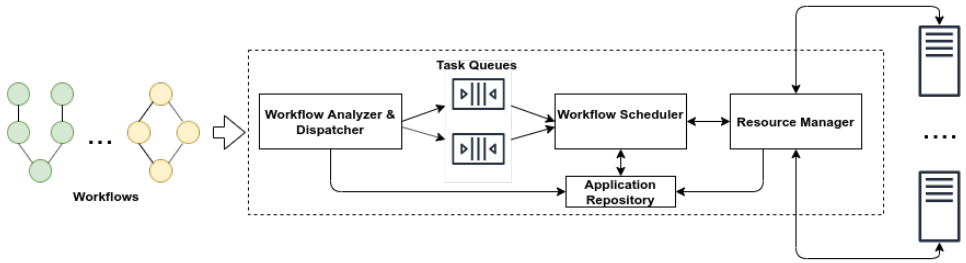



**Figure 3.1:** Proposed energy-efficient workflow scheduling model

hundreds of thousands of parallel instances. A migration based resource allocation policy is used in this work to enhance the efficiency of resource utilization. However, overheads caused by frequent migrations could adversely impact the energy efficiency.

### 3.2.2 Network Aware Energy-Efficient Scheduling

DENS methodology introduced by D. Kliazovich et. al [106], presents an energy efficient scheduling method with network awareness which monitors the status of network elements (switches, links) and incorporates their feedback to scheduling decisions. This study considers the tradeoffs between consolidating workloads on to a minimum number of servers and the impact of that on hardware reliability of servers and hotspot creation in data center networks.

S. Vakilinia [107] proposed a joint optimization approach for power minimization in large-scale multi-tenant cloud datacenters taking into account multiple factors including power consumption of servers, network communications, cost of VM migrations, heterogeneity of servers and resource constraints. This work suggests that rather than performing VM placement and migration in two steps, higher efficiencies can be achieved by jointly considering placement and migration decisions in each scheduling iteration.

NICE [108] presented by B. Cao et. al., is a joint optimization technique that consists of three subroutines, each of which is solved by a different algorithm. In the first subroutine, VMs are sorted in the order of decreasing computational resources and packed into a set of virtual hosts. The assignments of VMs to virtual hosts are readjusted in the second subroutine to minimize inter-host traffic. In the third subroutine, a greedy algorithm is used to map virtual hosts to physical hosts that result in minimal traffic and

migration costs. This study attempts to localize traffic flows in the DCN by minimizing communication distances so that unused switches can be put into dormant state to achieve additional power savings.

VMPlanner [109] proposed by W. Fang et. al. is a technique which incoporates three substeps to optimize VM placement and flow routing. Initially, VMs are grouped such that VM pairs that frequently communicate with each other are in the same group. VM groups are then mapped to server racks which leads to the minimization of inter-rack traffic. Afterward, traffic flows are consolidated into a minimum number of links so that unused networking elements can be put into dormant state. A joint host-network optimization method for energy efficient VM consolidation is presented by H. Jin et. al. in [110]. In order to achieve the objectives of optimal server placement and flow routing, the VM placement problem is converted to a routing problem and a single solution is developed to address both requirements.

All the above approaches have focused on the general problem of joint host and network optimization. They have not considered the potential power savings achievable by fine-tuning the algorithms to suit the characteristics and communication patterns of the underlying workloads.

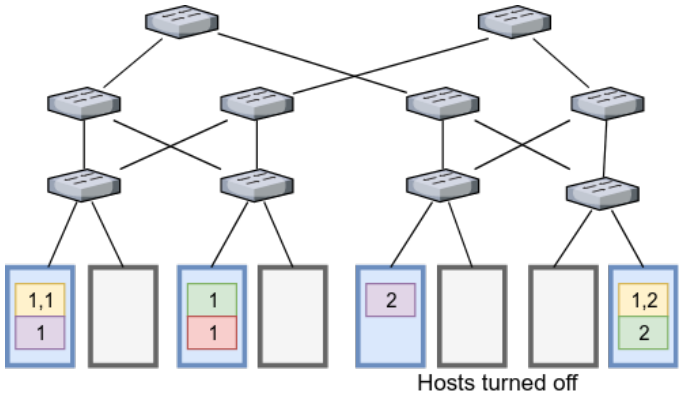
The workflow scheduling algorithms reviewed in section 3.2.1 have attempted to improve the energy efficiency of task schedules with little or no consideration about the impact of scheduling decisions on the data center network. In contrast, we attempt to enhance overall energy efficiency by jointly optimizing the utilization of servers as well as switches used in workflow executions. Our method also differs from topology-aware resource allocation methods reviewed in section 3.2.2, since we take into account the distinct features of workflows in the formulation of scheduling decisions.

## 3.3 Problem Modeling

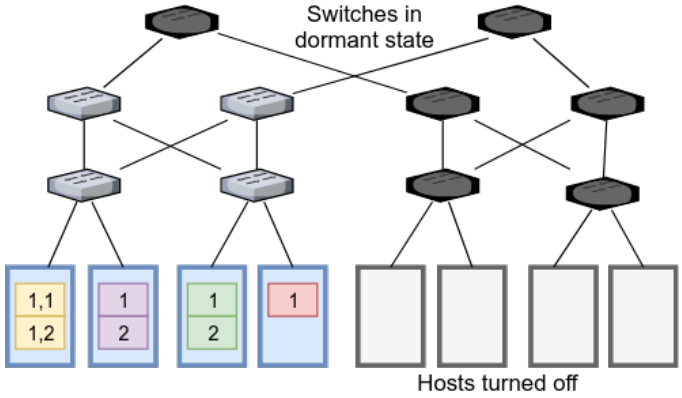
### 3.3.1 Application Model

A workflow can be modeled as a Directed Acyclic Graph (DAG) $G = (V, E)$, where $V = T_1, T_2, ..T_n$ is the set of tasks and $E$ is the set of edges which represent the data

dependencies among tasks in a workflow and the weight of an edge $e_{i,j} = (T_i, T_j)$ represents the size of data to be transmitted from $T_i$ to $T_j$. The edge $e_{i,j}$ also represents a precedence relation between tasks $T_i$ and $T_j$ such that $T_i$ is the parent task of $T_j$ and $T_j$ is the child task of $T_i$. Accordingly, the execution of $T_j$ can only start after the execution of $T_i$ is completed.




**(a)** Topology-unaware scheduling




**(b)** Topology-aware scheduling

**Figure 3.2:** A comparison of topology aware and unaware energy efficient workflow scheduling approaches

In this work, we have considered a divisible task model in which a task is composed of one or more instances which can be scheduled to execute in parallel on one or more physical resources. The execution of a task is considered to be complete when all the instances of it has finished execution, and a task is considered to be in ready state when all the instances of its parent tasks have completed execution.

### 3.3.2 System Model

The system architecture proposed in this work is oriented towards a scenario in which the WaaS platform and the underlying cloud infrastructure are managed by a single cloud service provider. Similar system models have been proposed in literature [23]. In contrast to these approaches, the architecture proposed in this work envisions a system in which each task executes within a dedicated micro virtual machine (micro VM). Figure 3.1 illustrates a high level overview of the proposed system model.

Workflow applications are submitted by tenants through an online portal. QoS (Quality of Service) and other application specific requirements desired by the tenants are specified in the application descriptions. Upon the analysis of a workflow description, the analyzer generates a workflow profile which is submitted to a centralized application repository. Application repository consists of a collection of workflow profiles. A workflow profile consists of multiple records, one for each task of the workflow. Each task record consists of the submission time of the workflow, task ID, workflow ID, execution status of the task (e.g. pending execution, ready for execution, in-execution, execution completed), rank, data dependencies of the task, resource requirements, execution details (e.g. machine IDs of executed task instances) and QoS requirements of the workflow task. Ranks are generated using the HEFT algorithm [16] described in the next section.

Workflow dispatcher periodically polls the workflow repository to retrieve details of tasks in ready state. In each iteration, the set of ready tasks are ordered based on the ranks and submitted to a global service queue. Tasks with stringent QoS requirements are submitted to a priority queue. Workflow scheduler respects the insertion order of the global task queue and operates in a FCFS (First Come First Serve) basis to ensure

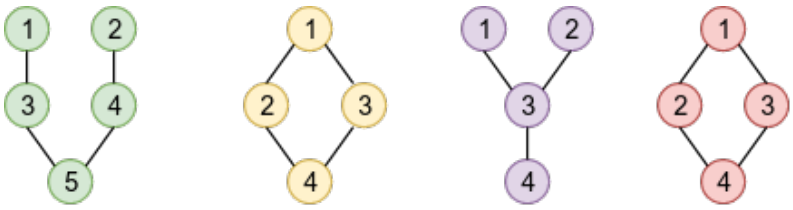



**Figure 3.3:** Example workflows

**Table 3.1:** Problem Notation

| Parameter | Value |
|---|---|
| $M$ | Total number of servers |
| $N$ | Total number of switches |
| $u_i$ | CPU utilization percentage of server $i$ |
| $P_i^{server}$ | Power consumption of server $i$ |
| $P_i^{idle}$ | Idle power consumption of server $i$ |
| $P_i^{dynamic}$ | Peak power consumption of server $i$ |
| $P_k^{switch}$ | Power consumption of switch $k$ |
| $P_k^{static}$ | Power consumption of switch $k$ without traffic |
| $P_k^{port}$ | Power consumption of each port of switch $k$ |
| $n_k$ | Number of active ports on switch $k$ |

fairness in multi-tenant environments. However, the insertion order is disregarded if a task with a higher priority arrives at the priority queue. The workflow scheduler coordinates with the resource manager and allocates resources to tasks in the service queue using the proposed topology aware resource allocation algorithm. The details required for the operation of the scheduling algorithm are retrieved from the respective task records in the application repository.

### 3.3.3 Power Model

In this section we formulate the joint server and network power optimization as a mono-objective optimization problem. Notations used in this chapter for the formulation of optimization problem is presented in Table 3.1. For the calculation of power consumption of servers, we used the CPU utilization based power model presented in [111]. Accordingly, power consumption of server i is defined as shown in the following equation.

$$P_i^{server} = \begin{cases} P_i^{idle} + (P_i^{dynamic} - P_i^{idle}).u_i, & \text{if } u_i > 0 \\ 0, & \text{otherwise} \end{cases} \tag{3.1}$$

Idle power consumption is a constant factor which incurs irrespective of the utilization level of a server and it can only be eliminated by turning off the servers. Dynamic power consumption of a server can be accurately computed by considering that the CPU utilization and the power consumption of a server follows a linear relationship [111].

The power model presented in [112] is used to compute the power consumption of switches. This model computes the power consumption of a switch as the sum of static power and the port power based on the number of active ports as shown in the following equation.

$$P_k^{switch} = \begin{cases} P_k^{static} + P_k^{port}.n_k, & \text{if switch } k \text{ is on} \\ 0, & \text{otherwise} \end{cases} \tag{3.2}$$

### 3.3.4 Problem Formulation

The focus of this work is on system wide minimization of energy consumed by servers and networking elements. Hence, the optimization objective can be formulated as:

$$\text{Minimize: } \sum_{i=1}^{M} P_i^{server} + \sum_{k=1}^{N} P_k^{switch} \tag{3.3}$$

Note that we have ignored the energy consumption of external and internal communications among tasks from the problem formulation since they are negligible in comparison to the total power consumption of servers and switches [21].

## 3.4 Proposed Algorithm

Scheduling techniques that are agnostic to the impact of resource allocation strategies on the DCN are less effective in terms of total power savings that can be achieved by workload consolidation. Aforementioned scenario is illustrated through the simple example

in Figure 3.2. Figure 3.2a and Figure 3.2b illustrate possible outcomes of a scheduling iteration in which workflows shown in Figure 3.3 are scheduled by a topology-unaware and a topology-aware energy-efficient resource allocation technique, respectively. This example demonstrates the manner in which the topology-aware resource allocation technique achieves comparatively more power savings by putting unused networking elements into dormant state.

The proposed topology-aware heuristic-based algorithm, J-OPT aims to schedule precedence constrained tasks in an energy-efficient manner by jointly optimizing the utilization of servers and networking elements. We have complemented J-OPT with Dynamic Power Management (DPM) techniques [41] which operate by switching off idle servers and switches to save energy.

### 3.4.1 Task Prioritization

Task prioritization uses the concept of upward rank presented in the well-known low complexity algorithm HEFT [16]. In each scheduling iteration, tasks of a job in ready state are ordered in decreasing order of upward rank ($rank_u$) before being submitted to the global service queue. The upward rank of a task $T_i$, is computed recursively using the following equation:

$$rank_u(T_i) = \overline{w_i} + \max_{T_j \in succ(T_i)} (\overline{c_{i,j}} + rank_u(T_j) \tag{3.4}$$

where $\overline{w_i}$ is the average computation cost of $T_i$, $\overline{c_{i,j}}$ is the average communication cost of edge $e_{i,j} = (T_i, T_j)$ and $succ(T_i)$ is the set of parent tasks of $T_i$. For the exit task (a task with no children), rank can be computed as follows:

$$rank_u(exit) = \overline{w_{exit}} \tag{3.5}$$

The average communication cost of an edge $e_{i,j}$ can be computed as follows:

$$\overline{c_{i,j}} = \overline{L} + \frac{data_{i,j}}{\overline{R}} \tag{3.6}$$

where $\overline{L}$ is the average communication start up time and $\overline{R}$ is the average communi-

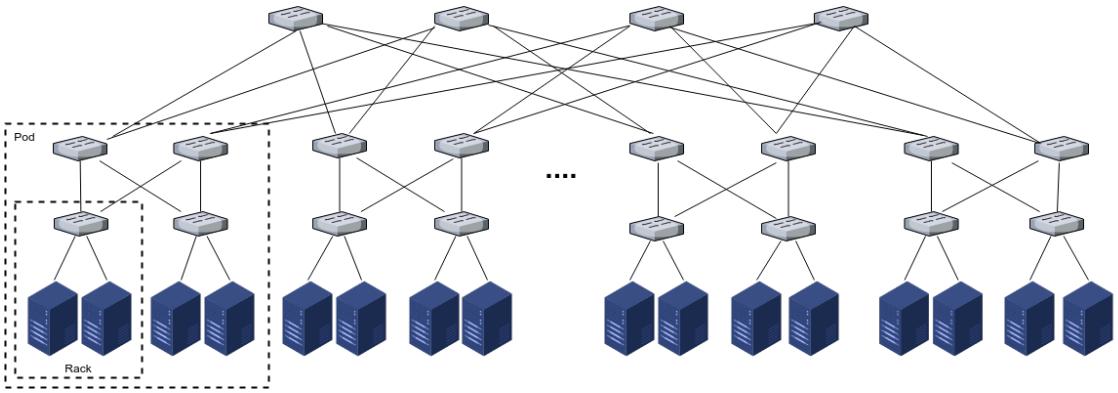



**Figure 3.4:** Fat tree network topology

cation rate among servers and $data_{i,j}$ is the amount of data to be transferred from task $T_i$ to task $T_j$.

### 3.4.2 Topology Aware Resource Allocation

In this section we present the topology-aware resource allocation technique used for mapping computing resources to tasks in the global service queue. The data center network can be represented as an undirected graph $G_{DCN}(V^{'}, E^{'})$ where $V^{'}$ is the set of vertices and $E^{'}$ is the set of edges. The set of vertices represent both servers as well as switches and the set of edges represent communication links between pairs of switches and between servers and switches.

Tasks in a workflow may have data dependencies giving rise to inter-task communications and associated communication costs. Furthermore, in the multi-instance task model considered in this work, a single task may have multiple instances that can be mapped on to one or more physical nodes and the instances may be communicating with each other during the period of execution.

To minimize the cost of communication, resource allocation algorithm should attempt to place instances of the same task on physical nodes that are as close as possible to each other in terms of the number of network hops. The placement is further complicated since it should be such that the aggregate distance to physical nodes in which the predecessor tasks with data dependencies executed is minimized. Accordingly, this resource allocation problem reduces to a variant of the proven NP hard problem in [113].

**Algorithm 1** TASK-SCHEDULING

Input: $PM$: List of servers
Input: $G_{DCN}(V^{'}, E^{'})$: Data center network
Input: $R_i$: Total resource requirements of task $T_i$
Input: $PM_{conn}$: List of hosts in which predecessors of task $T_i$ with data dependencies executed
Output: $allocMap$: Resource allocation map

```
1: subgraphList ← ∅
2: subgraphList ← NEIGHBOR-GEN(PM, PM_conn, G_DCN(V', E'))
3: for each PM_cand ∈ subgraphList do
4:     Order servers in PM_cand in descending order of desirability score
5:     while PM_cand ≠ ∅ do
6:         P ← PM with highest desirability score in PM_cand
7:         for each r ∈ R_i do
8:             if allocation(r, P) is successful then
9:                 if P ∉ PM_conn then
10:                    PM_conn ← PM_conn ∪ P
11:                R_i ← R_i − r
12:                PM_cand ← PM_cand − P
13:        if R_i = ∅ then
14:            return true
15: return false
```

Algorithm 1 operates with the local optimization perspective of placing communicating instances of a particular workflow on servers which are in close proximity such that the total number of networking elements used during the execution of a workflow is minimized. To achieve this, we introduce the concept of a dynamically expanding set of servers ($PM_{conn}$) which is provided as an input to the algorithm. $PM_{conn}$ is initialized with the set of servers in which predecessors of the current task with data dependencies executed. $PM_{conn}$ expands dynamically as more and more servers are selected for satisfying the resource requests of a newly arrived task.

Next we introduce the concept of neighbor subgraph of a server at a pre-defined hierarchical level in the DCN. Hierarchical level is defined in a topology specific manner and in this study we have considered a fat tree topology [114] with 3 hierarchical levels (Rack, Pod and DCN) as indicated in Figure 3.4. For a particular server, the neighbor subgraph at level 1 constitute the set of servers on the rack in which the server resides. Neighbor subgraph at level 2 includes the set of servers located in the pod (group of

**Algorithm 2** NEIGHBOR-GEN

Input: $PM$: List of all servers
Input: $PM_{root}$: A list of servers, the sub-graphs of which are to be generated
Input: $G_{DCN}(V^{'}, E^{'})$: Data center network
Output: A list of neighbor subgraphs of servers in $PM_{root}$

1: $currLevel \leftarrow 1$
2: $subgraphList \leftarrow \emptyset$
3: Mark all $H \in PM$ as unassigned
4: **while** $currLevel < 3$ **do**
5:  **for** each $P \in PM_{root}$ **do**
6:  $neighSubgraph \leftarrow \emptyset$
7:  **if** $P$ is unassigned **then**
8:  $neighSubgraph \leftarrow \{P\}$
9:  **for** each unassigned $H \in PM$ **do**
10:  $paths \leftarrow$ getPaths$(P, H, currLevel)$
11:  **if** $paths \neq \emptyset$ **then**
12:  $neighSubgraph \leftarrow neighSubgraph \cup \{H\}$
13:  Mark $H$ as assigned
14:  $subgraphList \leftarrow subgraphList \cup neighSubgraph$
15:  $currLevel \leftarrow currLevel + 1$
16: **for** each unassigned $H \in PM$ **do**
17:  $subgraph \leftarrow subgraph \cup \{H\}$
18: $subgraphList \leftarrow subgraphList \cup subgraph$ ▷ append the $subgraph$ of all unassigned servers to $subgraphList$
19: **return** $subgraphList$

racks) to which the server belongs, and the neighbor subgraph at level 3 comprises all the servers in the data center.

Algorithm 2 is used to obtain the set of neighbor subgraphs of the servers in $PM_{conn}$ at hierarchical levels 1 and 2 of the DCN. Depending on the topology of the data center network, Algorithm 2 can be replaced with a more sophisticated subgraph generation method. It should be noted that the sub-graph generation overhead can be completely eliminated by pre-computing the subgraphs of all servers at each hierarchical level of the DCN.

Lines 3-15 of Algorithm 1 attempts to find an allocation that can satisfy the total resource requirements of the current task by iterating through the neighbor subgraph list ($subgraphList$). In each iteration, the set of servers in a neighbor subgraph are consid-

ered as the candidate server list ($PM_{cand}$) for resource allocation. The desirability score described in section C is computed for each server in the $PM_{cand}$ list, and the server with the highest score is selected first. The algorithm attempts to assign as many resource requests as possible to the selected server. With each successful allocation, if the considered server is not currently in $PM_{conn}$, then it is added to $PM_{conn}$.

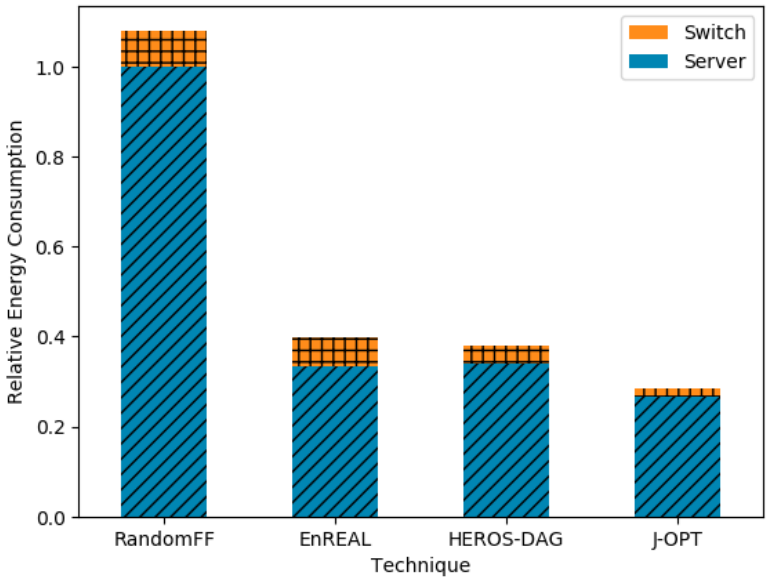
If the number of servers is denoted by $|H|$, the Algorithm 1 has a worst-case time complexity of $O(|H|^2)$ given that a sorting algorithm with worst-case time complexity of $O(|H|^2)$ is used in Line 4. Time complexity of Algorithm 2 can be ignored, since the neighbor sub-graphs can be pre-computed for eliminating the dynamic performance overhead.

As a further improvement, desirability scores can be recomputed for remaining elements of $PM_{cand}$ set per each new addition to $PM_{conn}$ (Line 10). This would be particularly useful if the workflow application to be scheduled is, for example, a SaaS application in which inter-task communication is frequent. However, inclusion of this step in the scheduling algorithm should be carefully determined based on the nature of applications to be scheduled and the scale of underlying cloud infrastructure as it leads to a significant rise in the worst case time complexity of the algorithm.

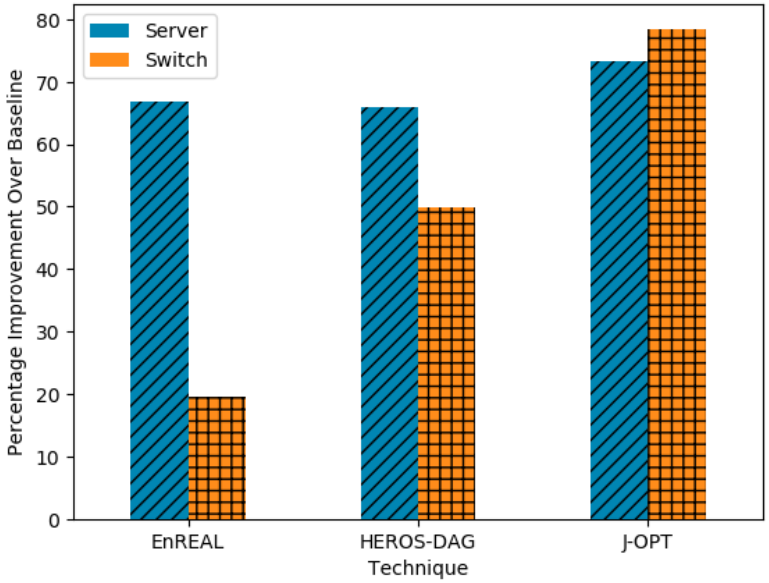
### 3.4.3 Desirability Score

The desirability score is used to rank the servers by considering the power efficiency of a server based on its current utilization and its physical location with respect to the set of servers in $PM_{conn}$ . Performance per watt is used to determine the energy efficiency of a server. Servers become most energy efficient when they are operating close to maximum capacity without being over-utilized [115]. Based on this observation, the desirability score is designed to favor servers that are more loaded compared to others without being over utilized.

It is also important that the server selection is not agnostic to the impact of the new assignment on network utilization. Hence, the new server additions to $PM_{conn}$ should minimize the use of links and switches, such that energy savings can be realized by putting the unused links and switches that do not carry traffic into power saving mode.




**(a)** Total energy consumption




**(b)** Percentage improvement over the baseline algorithm (RandomFF)

**Figure 3.5:** Energy consumption of scientific workflow executions

Accordingly, the desirability score also aims to better align the traffic distribution by favoring the selection of servers that minimize the aggregate physical distance between the final $PM_{conn}$ list in terms of the number of hops.

In order to achieve both aforementioned goals, we formulate the desirability score as a bi-objective function which combines server utilization efficiency and network utilization efficiency with the use of a normalized weight factor ($\alpha$) that indicates which factor should be given more prominence for minimizing overall energy consumption. The first term of the bi-objective function is based on HEROS [115] which in turn is based on DENS [106]. It is composed of the product of the server selection function $H_s(l)$ and communication potential function $Q(u)$ shown below:

$$H_s(l) = P_s(l) * (1 - \gamma.\frac{\langle 1 \rangle}{\langle (1 + \exp^{-\frac{\mu}{maxl_s}}(l - \beta.maxl_s)} \tag{3.7}$$

where $P_s(l)$ is the performance per watt of the server at load $l$ and the second term is a sigmoid function which is aimed at preventing the over utilization servers. Following the reference work [115], values 110, 0.9 and 1.2 were used for the coefficients $\mu$, $\beta$ and $\gamma$ in our experiments.

The communication potential is based on the actual link load $u$ and maximum link capacity $U_{max}$, and is defined as:

$$Q(u) = \exp^{-}(\frac{2u}{U_{max}})^2 \tag{3.8}$$

The second term of the desirability score is the distance function which is defined as:

$$D(i) = \sum_{j=1}^{W} d_{i,j} \tag{3.9}$$

where W is the total number of servers currently present in $PM_{conn}$ of task $T_i$, and $d_{i,j}$ is the aggregate distance in terms of the number of network hops between a candidate server and the set of servers in $PM_{conn}$.

Finally, the desirability score which represents the desirability of a server to be selected for scheduling one or more instances of a task $T_i$ is defined as:

$$\alpha.\frac{H_s(l).Q(u)}{H_f} + (1-\alpha).\frac{(D(i)+\epsilon)^{-1}}{D_f} \tag{3.10}$$

where $H_f$ and $D_f$ are two factors introduced to normalize the server selection function and distance function. $\epsilon$ is a very small positive real number.

## 3.5 Performance Evaluation

We evaluated the proposed algorithm in a simulated environment. For comparison purposes, we used three algorithms.

1. RandomFF - This is a baseline algorithm based on greedy first fit referred to as RandomFF in this chapter. RandomFF assigns requests to the first available host with adequate resources without attempting to improve either server utilization efficiency or network utilization efficiency.

2. HEROS-DAG - This algorithm is based on the independent task scheduling algorithm HEROS [115]. HEROS is designed to perform a random server selection if multiple equally desirable servers are among eligible candidates for resource allocation with respect to a decision function. We have adapted HEROS to cater to the requirements of workflows and incorporated network awareness by extending the server selection mechanism based on the physical location of the server as suggested in [105] for tie-breaking. Accordingly, in case of a tie, the server which is within a minimum hop distance to servers in which predecessors of the current task executed are selected from amongst the eligible candidates. The extended algorithm is referred to as HEROS-DAG in this chapter.

3. EnREAL - This is a state of the art algorithm [21] specifically formulated for energy-efficient scheduling of workflows in cloud computing environments. EnREAL uses a migration based resource allocation policy for achieving a high resource utilization efficiency.

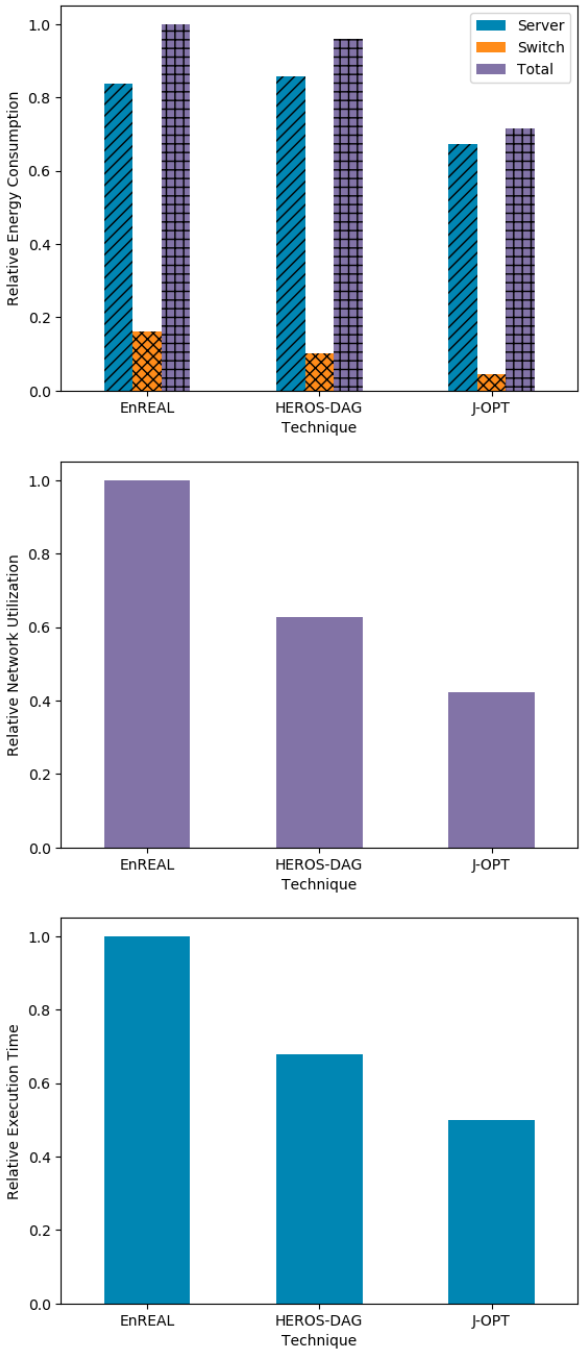



**Figure 3.6:** Performance of algorithms during heavy job loading data center occupancy state for scientific workflow executions

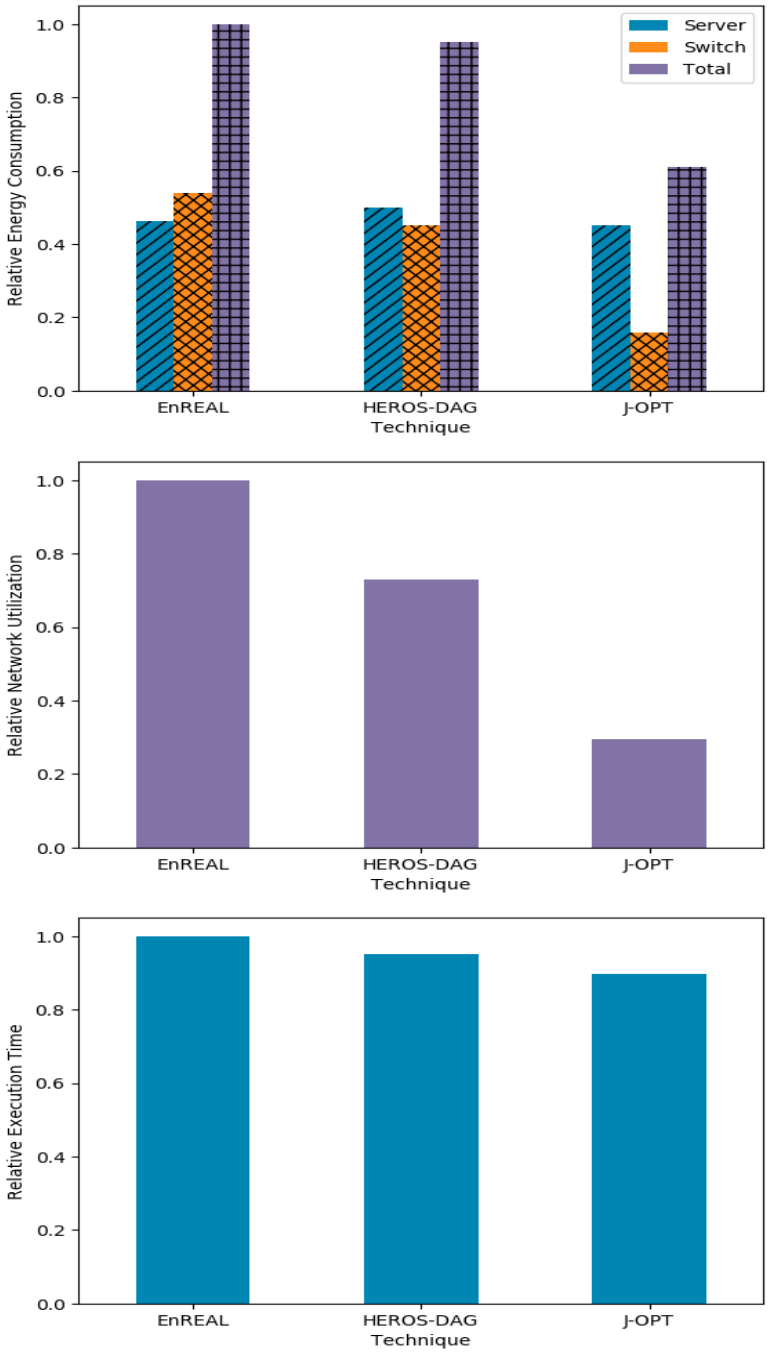



**Figure 3.7:** Performance of algorithms during light job loading data center occupancy state for scientific workflow executions

### 3.5.1 Simulation Environment

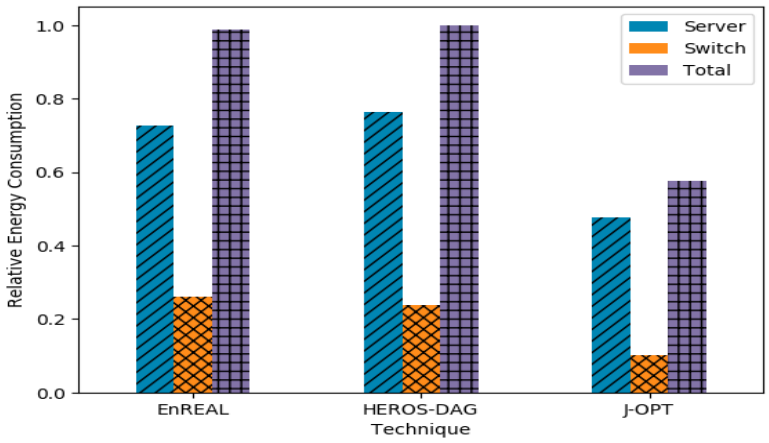
For conducting the experiments we used CloudSimSDN [11] which is a simulation tool based on the widely used CloudSim toolkit [116]. CloudSimSDN is specifically designed as a network simulation tool which facilitates SDN features such as dynamic network configuration, programmable controller and so on. We extended CloudSimSDN such that it can be used as a software-defined cloud environment in which the SDN controller can interact with the resource scheduler for performing scheduling decisions and related actions.

For the simulation, we used a homogeneous data center configuration. Equations 3.1 and 3.2 were used for modeling the power consumed by servers and switches, respectively. Fat Tree topology [114] was used for connecting servers in the simulated cloud data center.

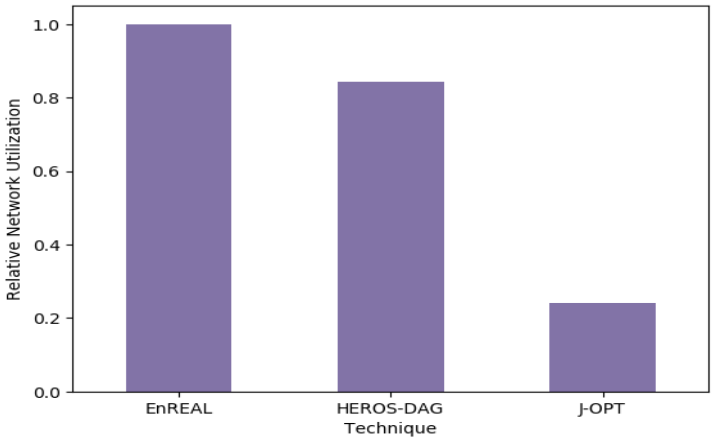
### 3.5.2 Datasets

We conducted two sets of experiments using two different DAG workloads. In the first set of experiments we used a number of well known scientific workflow benchmarks with different resource requirements: Montage, CyberShake, Inspiral, Epigenomics and SIPHT. We used the synthetic workflow traces provided by Pegasus group which are generated using traces from actual executions of scientific workflows [1]. For evaluating the performance of algorithms we created a composite workload using the aforementioned scientific workflows. To model the arrival pattern of the workflows we used a Poisson distribution.

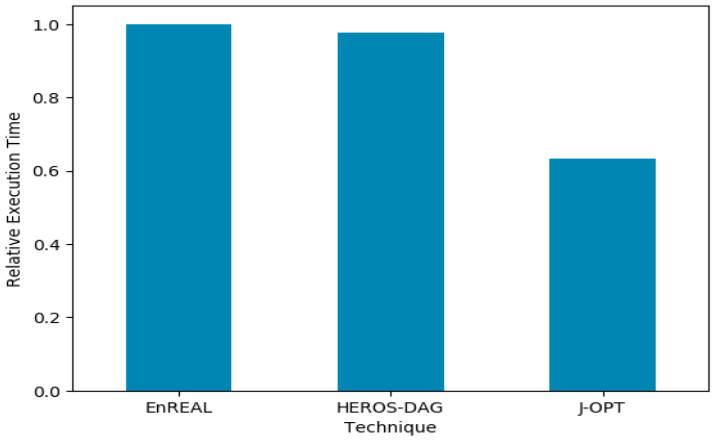
In addition to modeling scientific workflows, the DAG execution model can be used to model a wide variety of distributed applications and batch workloads. The second experiment was performed to evaluate the performance and suitability of J-OPT for scheduling non-scientific DAG workloads. For this experiment, we used a sample of 1000 workflows from workload traces extracted from a production cluster of Alibaba cloud [117].




**(a)** Total energy consumption




**(b)** Switch level network utilization




**(c)** Execution time

**Figure 3.8:** Performance of algorithms on a sample of 1000 workflows from Alibaba cluster traces

### 3.5.3 Results and Analysis

Figure 3.5a illustrates the energy consumption incurred from the execution of the composite scientific workload. Compared to the baseline RandomFF, all other algorithms have achieved significant savings in terms of total energy consumption. This is the expected behavior since EnREAL, HEROS-DAG and J-OPT incorporate multiple strategies specifically designed for enhancing the energy efficiency of scheduled workflows.

When analyzing the percentage improvements of the algorithms by disaggregating total energy consumption into its constituent components as depicted in Figure 3.5b, it can be observed that J-OPT far exceeds EnREAL and HEROS-DAG with respect to energy savings incurred from switch utilization. J-OPT has been able to achieve nearly 80% improvement over the switch energy consumption of RandomFF, while EnREAL and HEROS-DAG have only been able to achieve an improvement of 49% and 19% respectively. J-OPT has also surpassed other algorithms by 8% with respect to percentage improvement in server energy consumption over the baseline.

Although the importance of switch energy savings may appear less significant compared to total energy savings, it should be noted that as servers become fully energy proportional, the network energy consumption could rise as high as 50% of total data center energy consumption under light job loading conditions [13]. This is common in data centers since they are typically over-provisioned to meet peak load. Figure 3.6 and Figure 3.7 depict a comparison of the performance of algorithms under a heavy and light job loading scenario, respectively. Note that we have excluded the baseline algorithm's energy consumption from these comparisons since this situation occurs when servers are fully energy proportional, which is not the case with the baseline algorithm (RandomFF).

As illustrated in Figure 3.7, in the light job loading scenario network switches have consumed as much power as the servers with EnREAL and HEROS-DAG. In contrast, the topology-aware resource allocation strategy of J-OPT has been able to reduce the power consumption of network switches by a significant proportion, leading to an overall reduction of 30%-40% in the total energy consumption compared to state-of-the-art algorithms. Switch level network utilization is also reduced by a factor of 2 and 3 compared to HEROS-DAG and EnREAL, respectively.

In Figures 3.6 and 3.7, it can be observed that compared to other algorithms, J-OPT performs better with respect to total execution time of workflows. This is particularly because J-OPT allocates resources in a manner that greatly reduces the communication distance between dependent tasks as well as communicating task instances. This, in turn, reduces delays caused by bulk-data transfers leading to a significant reduction in the waiting time of data-intensive workflows.

Figure 3.8 depicts the results obtained with the application of J-OPT and other comparison algorithms to a sample workload containing 1000 workflows from Alibaba cloud traces. Note that although we have included EnREAL in this experiment, it is specifically designed for scientific workflows, and the migration based resource allocation policy of EnREAL is less appropriate for scheduling short spanned tasks in Alibaba workflows. As illustrated in Figure 3.8a, J-OPT has outperformed the comparison algorithms in terms of total energy consumption. As previously discussed, the significance of power savings achievable with J-OPT would be even more prominent under light job loading conditions. J-OPT has also achieved the lowest execution time.

The superiority of J-OPT in terms of minimizing switch level network utilization can be observed in Figure 3.8b. J-OPT has been able to reduce the switch level network utilization by over 70% compared to the other algorithms. This is because the topology-aware resource allocation mechanism of J-OPT attempts to place the tasks of a workflow in a group of closely located servers. This, in turn, minimizes the number of aggregate and core switches used during the execution of workflows.

## 3.6 Summary

In this chapter, we presented J-OPT, a novel topology-aware resource allocation technique for energy-efficient workflow scheduling in cloud data centers. J-OPT operates with the objective of minimizing total data center power consumption by jointly optimizing the utilization of servers and networking elements used in the execution of workflows. We have evaluated J-OPT in a simulated environment using synthetic and real-world workflow traces of scientific as well as commercial applications, and the results clearly demonstrate the effectiveness of the proposed algorithm compared to state-

of-the-art approaches.

Despite the satisfactory results achieved with the heuristic proposed in this chapter, there are inherent limitations associated with heuristic techniques. For instance, the design of an efficient heuristic requires advanced domain knowledge and the chosen heuristics are not guaranteed to be optimal. Although scheduling algorithms based on meta-heuristics are typically capable of overcoming the aforementioned limitations of heuristics, they are less appropriate for highly dynamic cloud environments due to high computational costs and time complexities. Therefore in the remaining chapters of this thesis, we have leveraged RL techniques for designing efficient workflow scheduling algorithms. In next chapter, we study the problem of workflow scheduling across cloud and edge computing environments, and propose a DRL framework for energy and time optimized scheduling of workflows.

# Chapter 4

# Energy and Time Optimized Scheduling of Workflows in 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. Performance of the proposed framework was compared against several baseline algorithms using energy consumption, execution time, percentage of deadline hits and percentage of jobs completed as evaluation metrics. Proposed Deep Reinforcement Learning technique performed 56% better with respect to energy consumption and 46% with respect to execution time compared to time and energy optimized baselines, respectively. This was achieved while also maintaining the energy efficiency in par with the energy optimized baseline and execution time in par with the time optimized baseline. The results thus demonstrate the superiority of the proposed technique in establishing the best-trade off between the conflicting goals of minimizing energy consumption and execution time.*

This chapter is derived from:

- **Amanda Jayanetti**, Saman Halgamuge, Rajkumar Buyya, "Deep reinforcement learning for energy and time optimized scheduling of precedence-constrained tasks in edge–cloud computing environments", *Future Generation Computer System*, Volume 137, Pages: 14–30, 2022

## 4.1 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, and complexities associated with the transmission of huge volumes of data over networks with limited bandwidths.

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, and reduced bandwidth usage, 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 [118] while others have suggested that depending on the network infrastructure and application characteristics it could be less energy-efficient in certain scenarios [119]. 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 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 [120], [10], [121], [122]. Only very few studies have considered more complex workloads, such as those with precedence relations [88], [123]. In the remainder of this chapter, 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). Therefore, in this chapter 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 [14]. 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 present an RL model for energy and time optimized scheduling of precedence constrained tasks in edge-cloud environments. We design an energy and deadline integrated reward model for training the DRL agent to establish a desired trade-off between the conflicting objectives of energy optimization and time minimization in workflow executions across cloud and edge computing environments.

- We propose a novel hierarchical action space formulation. Different from existing studies in which all edge and cloud nodes are considered together in non-hierarchical action spaces, 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 enhancing 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 chapter is organized as follows: In Section 4.2 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.

## 4.2 Related Work

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

### 4.2.1 Cloud Computing Environments

A number of studies [124], [125] have used RL for addressing the problem of task scheduling in cloud computing environments. In [124] 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 [125] 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 [55], [50], [79], [126], [78], [127] for scheduling dependent tasks of workflows in cloud computing environments. In [55] 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 [50]. In [79], a multi-agent reinforcement learning framework for multi-objective workflow scheduling in cloud infrastructures is proposed. The

| Work | Application Model | | Edge-Cloud | Algorithm | Heterogeneous | Objectives |
|---|---|---|---|---|---|---|
| | Workflow | Bag of Tasks | | | | |
| 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 4.1:** Summary of Literature Review

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 [126] 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 [127] 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.

### 4.2.2 Edge-Cloud Environments

A number of studies [120], [10], [121], [122] 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 [120] 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. [121] proposed a Double Deep Q Learning algorithm for task scheduling in fog computing environment with the objectives of minimizing delay and computation cost. In [10], A3C (Asynchronous Advantage Actor Critic) technique is used together with R2N2 (Residual Recurrent Neural Networks for task scheduling and migration in edge-cloud environment. [122] 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 [88], [123] used RL for scheduling dependent tasks in edge-cloud environment. [88] 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 [123], 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.

### 4.2.3 A Qualitative Analysis

Q learning is one of the most fundamental off-policy RL algorithms which forms the basis of many state-of-the art RL algorithms including Deep Q Learning. A large number of studies have used Q learning for dependent and independent task scheduling in cloud and edge computing environments [120], [124], [125], [55]. The scheduling algorithm proposed in [50] used the on-policy RL algorithm SARSA. An obvious advantage of Q learning and SARSA over model-based techniques such as dynamic programming is that they are model-free RL algorithms which do not require complete knowledge about the dynamics of the environment. Therefore scheduling techniques based on these RL algorithms are capable of operating in highly unpredictable cloud and edge computing environments. However, these techniques also have a number of limitations. Q learning

as well as SARSA require the RL agent to visit all states during the training process and store the state transition data in a tabular format which is both time as well as space consuming. To overcome this issue in [120] state space discretization is used. The drawback of this approach is that discretization could lead to the loss of information.

The combination of RL with deep learning which is referred to as Deep Reinforcement Learning (DRL) has successfully proven to overcome the aforementioned issue through function approximation, thereby eliminating the need for agents to visit all states during the training process and for storing state transition data in space consuming tabular formats. Deep Q Learning is one of the most popular DRL algorithms used for task scheduling in cloud and edge computing environments [79], [126], [127]. DQN based scheduling algorithms are more cost effective compared to Q learning and SARSA based techniques since they require less time for training as the agents need not visit all states of the environment. One drawback associated with DQN algorithms is that they tend to overestimate the Q values of actions since the same function approximator is used for both action evaluation and selection. Double Deep Q Learning algorithm used in [121] for task scheduling overcomes this overestimation bias by decoupling action selection from evaluation with the use of two function approximators [128].

DQN typically requires complete state information which is not readily available in mobile edge computing environments which tend to be partially observable due to high complexity and dynamicity of the environment. To address this issue DQN is coupled with LSTM networks in [123], [122]. The recurrent nature of LSTM networks facilitates the integration of long term historical data for accurately estimating the system state. An inherent weakness associated with Q learning based algorithms is the need to perform a maximization over all the actions in the action space, which is intractable with very large action spaces. Policy gradients are a branch of RL algorithms that are better suited for environments with very large action spaces. As opposed to value based RL algorithms such as Q learning and DQN that derive a policy from a learnt value function, policy gradients directly learn a parameterized policy. Despite the aforementioned advantages, vanilla policy gradients could take prohibitively long durations for learning complex policies due to the inherent sample inefficiency associated with them. To mitigate this issue [10] used Asynchronous Advantage Actor Critic (A3C) technique in

which multiple agents are trained in parallel and a global network with shared parameters are updated periodically thus speeding up the training process. As opposed to training a single agent, training multiple agents asynchronously consumes more computational power.

Regardless of the underlying RL algorithm, a common limitation in all reviewed techniques is that the service nodes are considered together in the same action space with no distinction between those that belong to cloud and edge tiers. This limits the applicability of the techniques for different scenarios. For instance, consider a commonly encountered scenario where tasks of certain workflows are restricted to execute only on edge due to security constraints. With existing single-agent RL techniques [120], [10], [121], [122], this situation cannot be handled since the trained model cannot distinguish between edge and cloud nodes. In the proposed work we design the action space in a hierarchical manner, so that the trained model can easily accommodate the aforementioned scenario with no modifications. We propose a novel multi-actor framework with a single critic for handling the hierarchical action space in an efficient manner. As evidenced by the results of extensive simulation experiments in Section 5, compared to the traditional single-actor method, the use of two actors has led to improved performance with respect to all evaluation metrics. For efficiently training the proposed DRL framework we use Proximal Policy Optimization (PPO) technique [58] which is capable of overcoming the inherent sample inefficiency associated with traditional actor-critic methods with the use of a clipped surrogate objective function. The following sections elaborate the design and implementation of the proposed scheduling framework.

## 4.3 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 chapter.

### 4.3.1 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, $G = (T, E)$ where $T$ represents the set of vertices and $E$ represents the set of directed edges. Each vertex in $T$ represents a computing task, $t_n$. Each edge $e_{i,j} \in E$ represents a data dependency between tasks $t_i$ and $t_j$ such that the execution of $t_j$ cannot be commenced until the execution of $t_i$ completes. Accordingly, a precedence constraint exists between the two tasks and $t_i$ is a predecessor of $t_j$ and $t_j$ is a successor of $t_i$. 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.

### 4.3.2 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. 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 4.1 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 chapter.

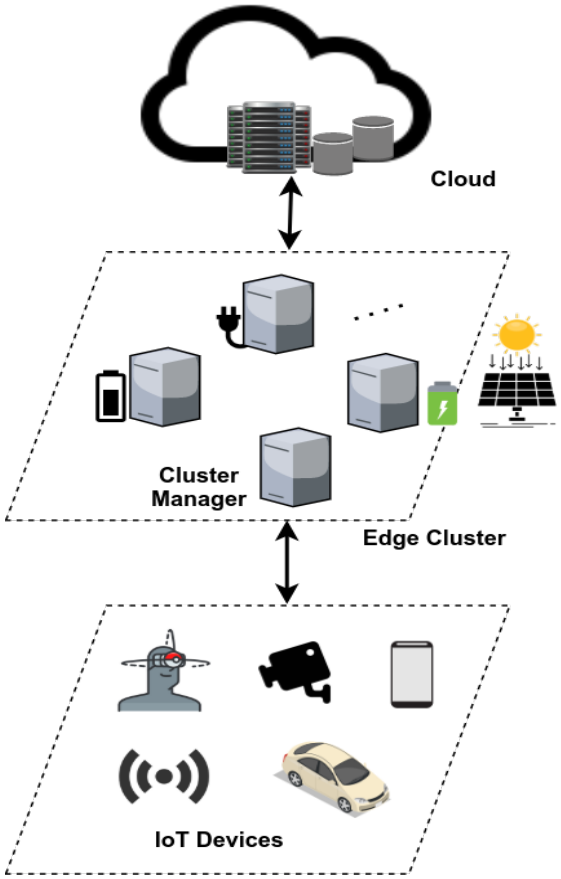



**Figure 4.1:** System Architecture

### 4.3.3 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, $t_j$ with size $L(t_j)$ in a server with processing rate $F$ can be expressed as follows:

$$CT(t_j) = \frac{L(t_j)}{F} + WT(t_j) \tag{4.1}$$

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

In order for the execution of task, $t_j$ to be commenced, all the precedence constraints of the task must be satisfied. This means all the predecessors of $t_j$ must have completed execution and the output data from predecessors required for the execution of $t_j$, must be available at the node to which $t_j$ is assigned. Let $t_i$ be an immediate predecessor of task $t_j$ and the size of data to be transferred from $t_i$ to $t_j$ 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:

$$TT(t_i, t_j) = \frac{D(t_i, t_j)}{B} \tag{4.2}$$

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

$$EST(t_j) = \max_{t_i \in pred(t_j)} (FT(t_i) + TT(t_i, t_j)) \tag{4.3}$$

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

$$FT(t_j) = EST(t_j) + CT(t_j) \tag{4.4}$$

The completion time ($MT$) of a workflow will then be equivalent to the completion time of the task that completes execution last as represented by the following equation:

$$MT = \max_{t_j \in T}(FT(t_j)) \tag{4.5}$$

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

### 4.3.4 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 [111] the energy con-

sumed during the computation of task, $t_j$ can be expressed as follows:

$$ECOMP(t_j) = CT(t_j) \times [U \times P_{active} + P_{idle}] \tag{4.6}$$

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:

$$ECOMM(t_j) = \sum_{t_i \in pred(t_j)} TT(t_i, t_j) \times P_{comm} \tag{4.7}$$

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:

$$E = \sum_{t_j \in T} (ECOMP(t_j) + ECOMM(t_j)) \tag{4.8}$$

### 4.3.5 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. This is because when the scheduling algorithm operates with the sole objective of minimizing energy consumption, tasks maybe allocated to more energy-efficient yet relatively slower servers thus lengthening task execution times. However, it's important to impose a limit on the extension of execution time to prevent the degradation of user experience. 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. This means while the primary focus of the scheduler is to minimize energy consumption, it will also attempt to formulate allocation decisions such that the workflows complete execution close to their deadlines.

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 chapter, 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.

The upward rank, $r_j$ of a task $t_j$ [16] can be computed recursively with the following equation:

$$r(j) = \max_{t_k \in succ(t_j)} (TT(t_j, t_k) + r(t_k) + \overline{w(t_k)}) \tag{4.9}$$

where $succ(t_j)$ is the set of successors of $t_j$, $\overline{w(t_k)}$ is the average execution time of $t_k$ and $TT(t_j, t_k)$ is the transmission time computed with equation 4.2. The length of critical path of a task is equivalent to its upward rank. For a workflow submitted to the system at time $ST$ with deadline $D$, the deadline $d(t_j)$ of a task $t_j$ can simply be represented as follows [105]:

$$d(t_j) = ST + r(A) + \overline{w(A)} - r(j) \tag{4.10}$$

where $A$ is the source task of the workflow and $\overline{w(A)}$ is the average execution time of $A$. Clearly the deadline derivation presented above is not appropriate for this work since the satisfaction of such deadlines will lead to the minimization of makespan rather than energy. The CP-P technique presented in [105] addresses the aforementioned problem by setting the deadlines of tasks to the maximum values. Slack time is divided evenly among all tasks by setting the deadlines proportionally to their ranks. Accordingly, the deadline of a task is calculated as below:

$$d(t_j) = ST + (D - ST) * \frac{r(A) + \overline{w(A)} - r(j)}{r(A) + \overline{w(A)}} \tag{4.11}$$

### 4.3.6 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:

$$\begin{aligned} &\text{Minimize: } \sum_{i=1}^{N} \alpha M_i + (1-\alpha) E_i \\ &\text{Subject to: } M_i \leq D_i \end{aligned} \tag{4.12}$$

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 $\alpha$ 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, $d(t_j)$ is assigned to each sub task, $t_j$. Accordingly, constraint in equation 4.12 can be rewritten as:

$$\forall_{t_j \in T} \quad FT(t_j) \leq d(t_j) \tag{4.13}$$

If task deadlines were set to minimize makespan as in equation 4.10, it will be possible to meet workflow deadlines even if some task deadlines are exceeded. In that case the constraint imposed by the condition in equation 4.13 is tighter than that in equation 4.12. However, since task deadlines are already set to maximum values by CP-P technique, the constraint imposed by the the condition in equation 4.13 is equivalent to that in equation 4.12. 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:

**Algorithm 3** 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
```

$$\begin{aligned} &\text{Minimize: } \sum_{i=1}^{N} E_i \\ &\text{Subject to: } \forall_{w \in W} \forall_{t_j \in T} \quad FT(t_j) \leq d(t_j) \end{aligned} \tag{4.14}$$

## 4.4 Deep Reinforcement Learning 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 time optimized workflow scheduling problem in the edge-cloud environment. We then describe the proposed DRL framework for efficiently handling the workflow scheduling problem.

### 4.4.1 Reinforcement Learning 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 [103], 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 3 summarizes the sequence of events involved in the execution of aforementioned steps.

**State Space** All the worker nodes periodically share their power consumption rates, queue statuses and resource capacities with the gateway node as described in Section 4.3.2. 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, $t_j$

2. Deadline of task, $t_j$. The ultimate objective of the scheduling problem is to ensure the QoS requirements (e.g. deadlines) of workflows are satisfied while minimizing energy consumption. Since the workflow scheduling problem is mapped to a task scheduling problem, from the perspective of the DRL agent, the goal would be to

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 $(d_1, d_2, ..., d_i)$ each element of which represents the amounts of data to be transferred from each node, $i$ 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 $i$ is the sum of total input data from all predecessors that executed in node $i$. If none of task's predecessors executed in node $i$, then $d_i = 0$.

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)

Accordingly, the size of state space is 3 + 11 * total number of nodes.

**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 [10], [120] 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 ($t_j$) should be allocated, and $a_2$ indicates the node (in the tier given by $a_1$) to which $t_j$ should be assigned for execution. Therefore the action space can be defined as follows:

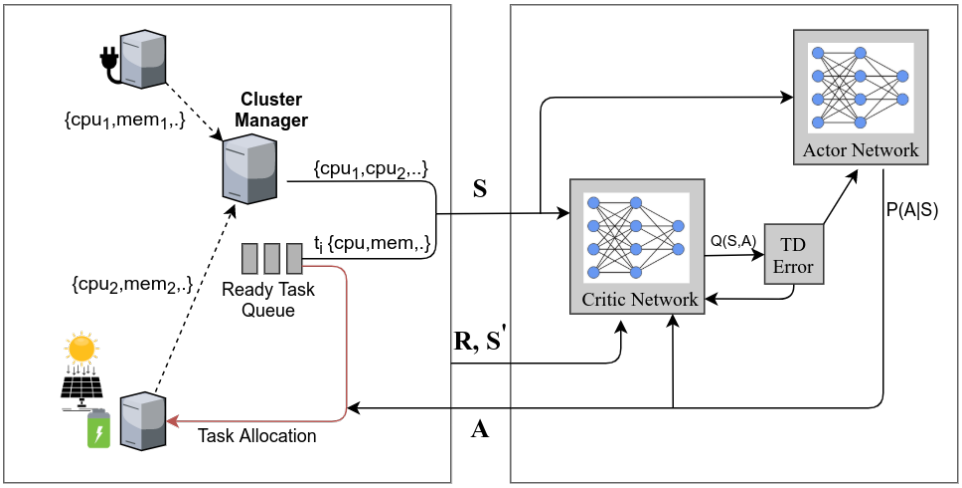



**Figure 4.2:** Traditional actor-critic based scheduling

$$A = \{(a_1, a_2) | a_1 \in \{Cloud, Edge\} \ \ \& \ \ a_2 \in \{1, 2, ..., N_{a_1}\}\} \tag{4.15}$$

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 [129] 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 [14].

**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.

2. $R_2$: A constant positive or negative reward depending on whether the selected node is capable of meeting task deadline, $d(t_j)$ or not:

$$R_2 = \begin{cases} +1, & \text{if } FT(t_j) \leq d(t_j) \\ -1, & \text{otherwise} \end{cases} \tag{4.16}$$

   where $FT(t_j)$ is the estimated completion time of task $t_j$ at the selected node. It can be calculated as in equation 4.4.

Accordingly, at each time step the reward ($r_t$) received by the DRL agent is $R_1$ and aggregate of $R_2$ for all the tasks scheduled in that time step.

### 4.4.2 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. In advantage actor critic method [130], the return, $G_t$ of REINFORCE is replaced with the advantage function, $A_\pi(s,a) = q_\pi(s,a) - v_\pi(s)$. If $\hat{A}_t$ represents an estimator of the advantage function at time t, the equation for updating policy parameters of the actor network can be denoted as follows:

$$\theta_{t+1} = \theta_t + \alpha \hat{A}_t \nabla \ln \pi(a_t|s_t, \theta_t) \tag{4.17}$$

In this work, as the estimator, $\hat{A}_t$ of the advantage function we used the Generalized

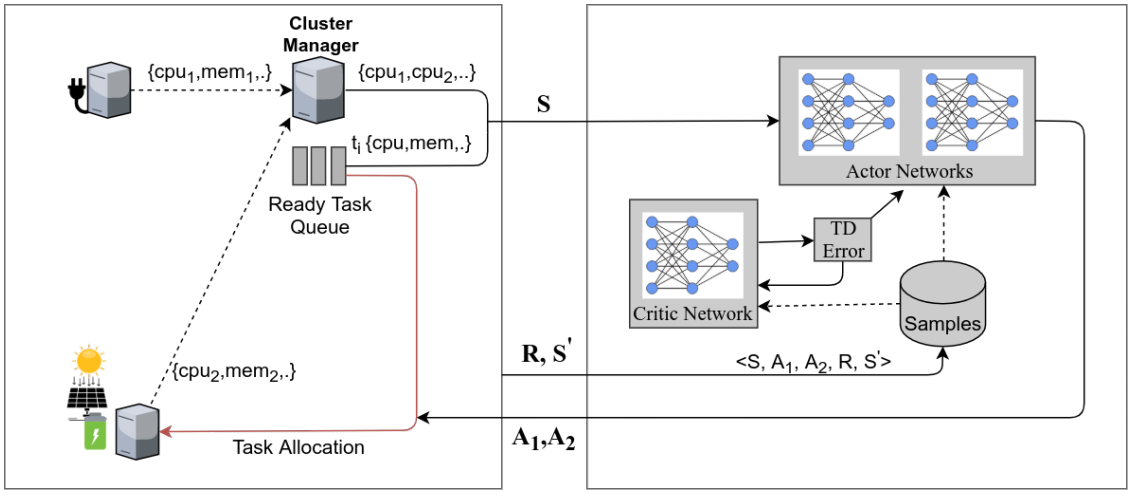



**Figure 4.3:** Proposed scheduling framework

Advantage Estimator, $GAE(\gamma, \lambda)$ presented in [130], with $\lambda = 0$. Let $V$ be an approximate value function, and $R_t$ be the return then $\hat{A}_t$ can be defined as follows:

$$GAE(\gamma, 0) : \hat{A}_t := \delta_t = R_t + \gamma V(s_{t+1}) - V(s_t) \tag{4.18}$$

As expressed in equation 4.18, $\hat{A}_t$ is equal to $\delta_t$ in the special case where $\lambda = 0$. Note that we have used $\theta$ and $\omega$ to represent the set of adjustable parameters of the actor and critic networks respectively. The critic network is trained to learn the state-value function, $v_\pi(s_t|\omega)$. It is initialized with arbitrary weights which 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 ($v_\pi(s_t|\omega)$) and target values ($R_t + v_\pi(s_{t+1}|\omega)$) as shown in equation 4.19. And then updating the network parameters with TD error as shown in equation 4.20.

$$L(\omega) = \frac{1}{2}[v_\pi(s_t|\omega) - (R_t + v_\pi(s_{t+1}|\omega))]^2 \tag{4.19}$$

$$\begin{aligned} \omega &\leftarrow \omega + \beta\delta_t \nabla v_\pi(s_t|\omega) \\ where\ \delta_t &= R_t + v_\pi(s_{t+1}|\omega) - v_\pi(s_t|\omega) \end{aligned} \tag{4.20}$$

where $\beta$ is the learning rate of the critic network. 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 4.17, the actor-network also learns to produce actions that result in higher rewards.

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 1.5) cannot be made too large since that could lead to large policy updates that collapses performance. Trust Region Policy Optimization (TRPO) [131] techniques address the aforementioned problem by maximizing a surrogate objective function subject to a constraint, $\sigma$ as shown in Equation 4.21. KL indicates the Kullback-Leibler divergence in equation 4.21. The constraint restricts the degree to which new policy, $\pi_\theta(a_t|s_t)$ is allowed to change from the old policy, $\pi_{\theta_{\text{old}}}(a_t|s_t)$ hence enabling the policy to monotonically improve (approximately). In an algorithm that alternates between sampling and optimization, the expectation $\hat{E}_t[..]$ indicates the empirical average computed over a finite batch of samples. However, the theories which forms the basis of TRPO requires complex and computationally expensive calculations. Hence, we used Proximal Policy Optimization (PPO) [58] technique which is proven to provide the benefits of TRPO techniques, with the added advantages of less complexity, improved sample complexity and generalizability.

$$\begin{aligned} &\text{Maximize: } \hat{E}_t \left[ \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \hat{A}_t \right] \\ &\text{Subject to: } \hat{E}_t[KL[\pi_{\theta_{\text{old}}}(.|s_t), \pi_\theta(.|s_t)]] \leqslant \sigma \end{aligned} \tag{4.21}$$

If the surrogate objective function of TRPO in Equation 4.21 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 4.22. This is achieved through the clip function, $\text{clip}(\mu_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t$ which removes the desirability of large policy updates that changes the $r_t(\theta)$ ratio beyond the interval $[1-\epsilon, 1+\epsilon]$.

$$\begin{aligned} &L^{CLIP}(\theta) = \hat{E}_t[\min(\mu_t(\theta)\hat{A}_t, \text{clip}(\mu_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t] \\ &\text{where } \mu_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \end{aligned} \tag{4.22}$$

As described in Section 4.4.1, 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 [129] designed for parameterized action spaces to the hierarchical action space in our problem.

Figure 4.2 and Figure 4.3 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 4.4.1. 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 store 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 to edge tier. Since there are multiple nodes 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.

**Algorithm 4** Actor-Critic based Scheduling Framework with PPO

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

Pseudocode of the training process of the proposed multi-actor scheduling framework is presented in Algorithm 4. 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 se-

**Algorithm 5** Online Scheduling

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

lected 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 12-18). 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 5 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.

## 4.5 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

| 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 4.2:** Host configurations derived from SPEC benchmark [3] for experimental setup

scenarios.

### 4.5.1 Experimental Setup

For evaluating the performance of proposed workflow scheduling framework, we used used an extension [11] 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 [12].

The simulated scenario comprises of a highly heterogeneous cluster with 16 edge nodes and 8 cloud nodes. We used the SPEC benchmark [3] for obtaining the resource configurations and power consumption rates of nodes as shown in Table 4.3. Following the simulation experiments of similar works [10] based on empirical studies, communication delay between edge-edge nodes and edge-cloud nodes was considered to be 1ms and 10ms respectively.

### 4.5.2 Dataset

Evaluation dataset was created based on synthetic workflow structures [2] provided by the popular Peagasus workflow framework. Task length in terms of number of in-

| Parameter | Value |
|---|---|
| **General** | |
| Discount factor ($\gamma$) | 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 ($\beta$) | 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 ($\alpha$) | 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 ($\alpha$) | 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 4.3:** Hyper-parameters used for the DRL model

structions (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 [16]. 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 base [121] 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 4.5.3) can meet approximately 90% of deadlines.

### 4.5.3 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 [16] 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 4.1, 4.2, 4.3 and 4.4 for computing the estimated end times 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 4.4 and 4.6 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:

$$SCORE = FT(t_i) \times ECOMP(t_i) \tag{4.23}$$

4. **EES** This is an energy efficient scheduling algorithm 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 4.6 to compute the energy consumed at each node for task execution.

5. **Single-Actor** This is a DRL model in which all edge and cloud nodes are considered together in the same non-hierarchical action space. In this comparison method we used a traditional actor critic network and trained it using the same hyper-parameters as the DRL model proposed in this work.

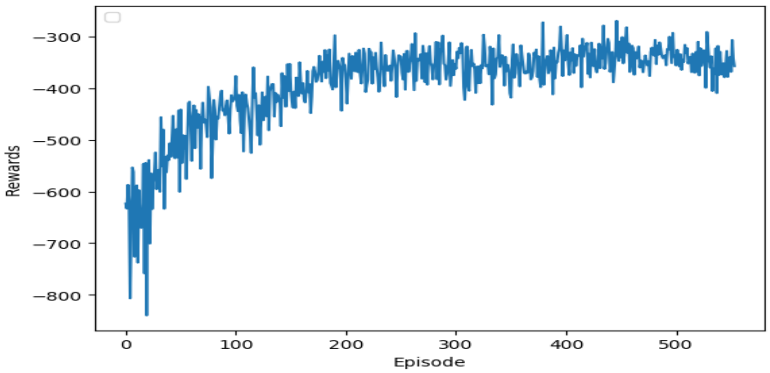
### 4.5.4 Hyper-parameters and Network Configurations

Table 4.3 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

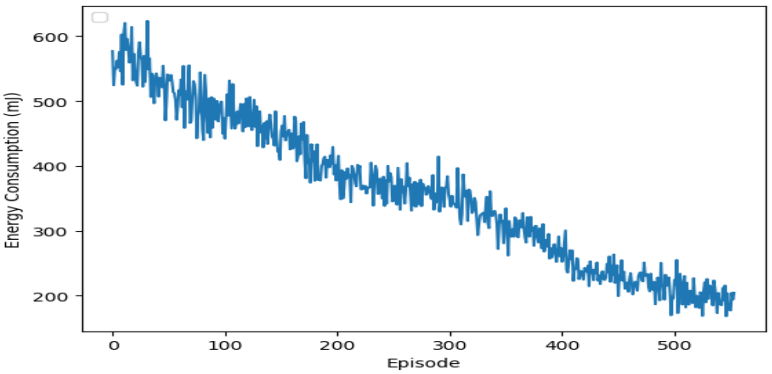
of training more often leads to sub-optimal convergence. In the training process of the DRL agent we only considered computation energy consumption in the calculation of total energy consumption of the system ($R_1$) since energy consumed for communication was negligible in comparison to energy consumed for computations. In communication intensive environments, the reward should include energy cost of communications as well. 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.

### 4.5.5 Analysis of Convergence

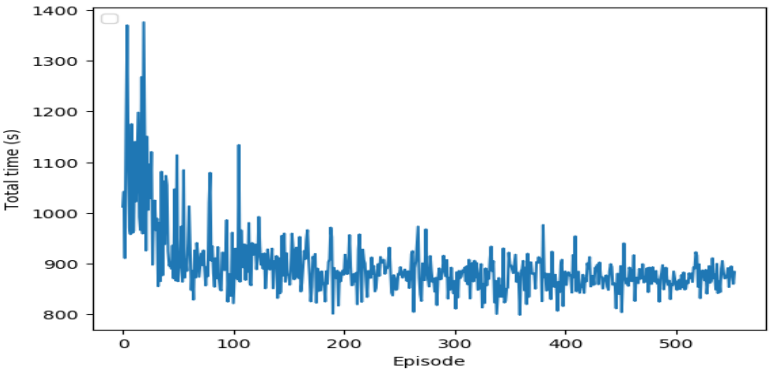
Figure 4.4 demonstrates the manner in which the agent learns to produce actions which leads to the achievement of desired objectives as training progresses. As shown in Figure 4.4a, total rewards accumulated during an episode gradually increase and converges to a maximum around the $550^{th}$ episode. The convergence of a DRL model is evaluated based on reward convergence. Oscillatory trends in other parameters can be expected as a result of ongoing exploration at early phases and equally favorable learned actions in latter phases. 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 4.4b, 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 4.4d. 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 4.4c), 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.



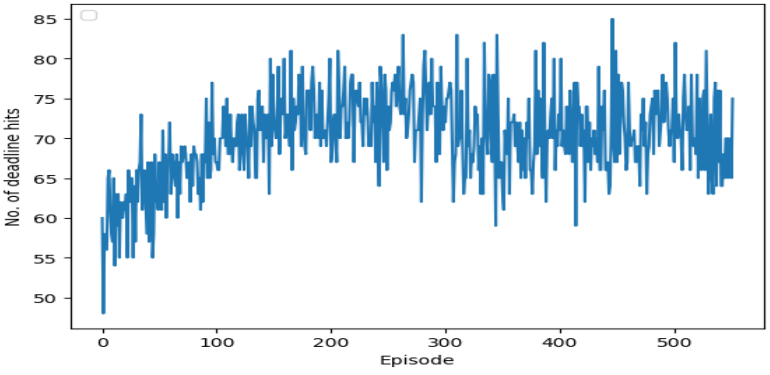

**(a)** Reward convergence




**(b)** Total energy consumption

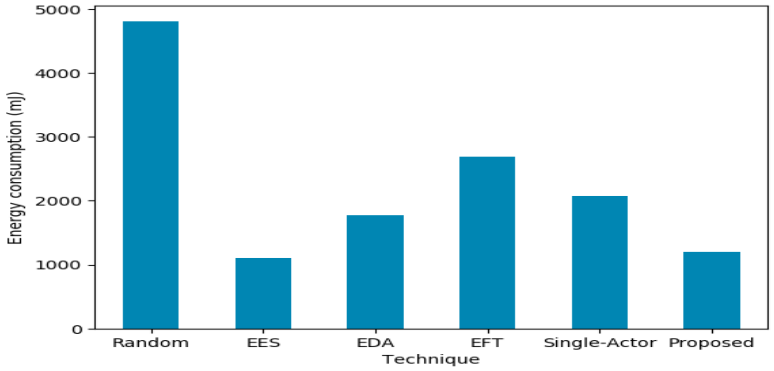



**(c)** Total time

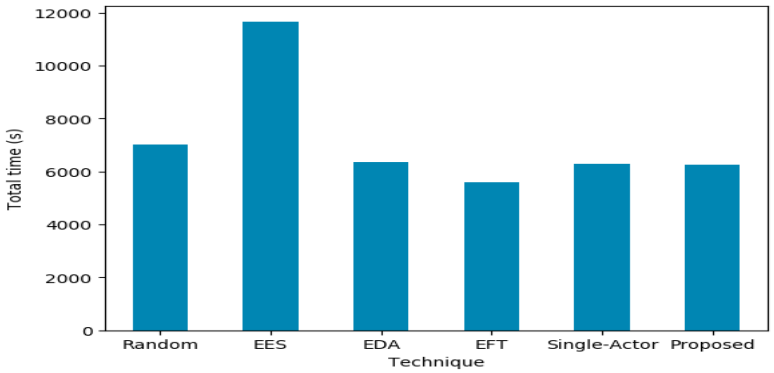



**(d)** Deadline hits

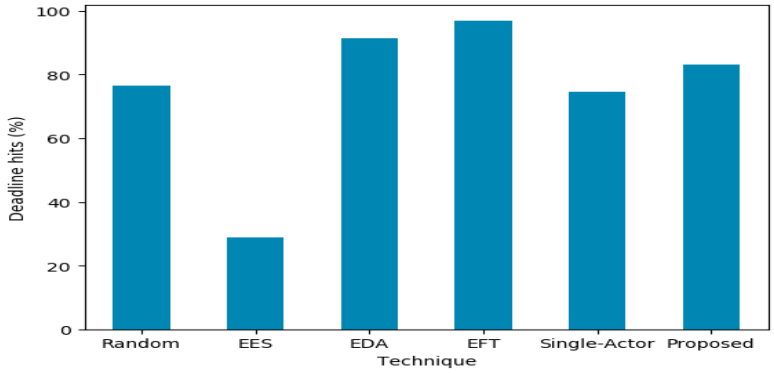
**Figure 4.4:** Learning progress with training (Convergence of DRL model)



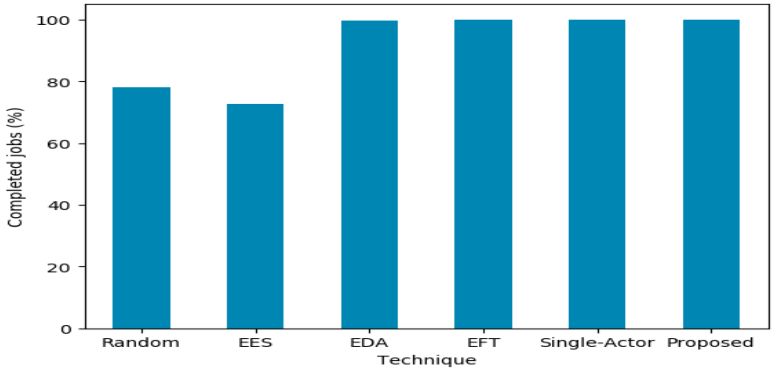

**(a)** Total energy consumption




**(b)** Total time




**(c)** Deadline hits




**(d)** Completed jobs

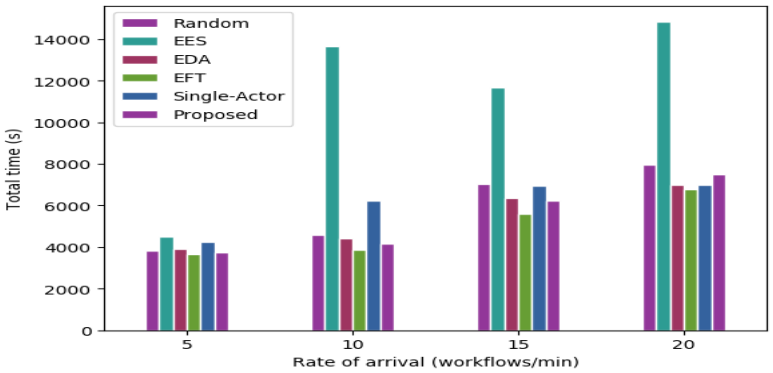
**Figure 4.5:** Comparison of performance of scheduling algorithms on experimental dataset

### 4.5.6 Analysis of Performance on Experimental Dataset

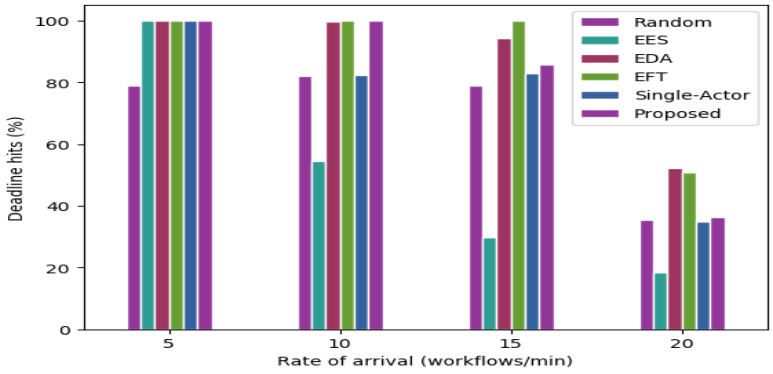
Figure 4.5 demonstrate the performance of the algorithm on the experimental dataset at an arrival rate of 15 workflows/minute. The total energy consumption incurred during the execution of workflows is demonstrated in Figure 4.5a. Clearly, Random algorithm has resulted in the highest energy consumption. 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. Single-Actor method consumes significantly more energy compared to the proposed DRL method. Since this technique and proposed DRL method are identical in every aspect except that the proposed method has a hierarchical action space and uses separate actors for determining the task allocation tier and node, the difference in energy consumption between the two methods clearly highlights the advantage of using separate actor networks.

Figure 4.5b 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 commu-

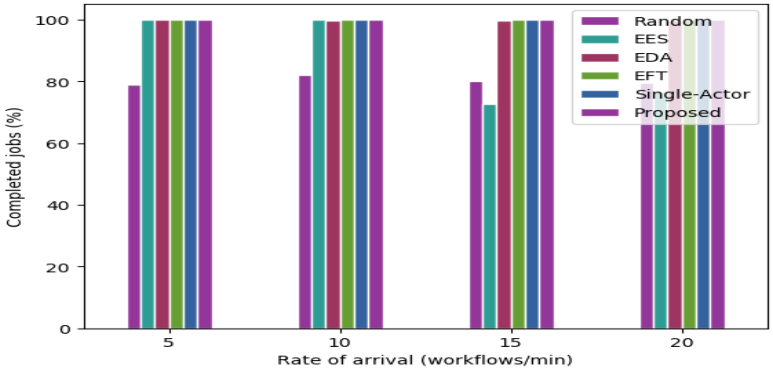

**(a)** Total energy consumption




**(b)** Total time




**(c)** Deadline hits



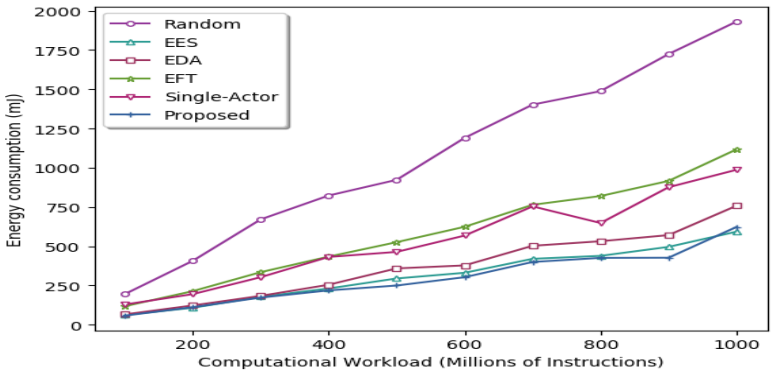

**(d)** Completed jobs

**Figure 4.6:** Comparison of performance of scheduling algorithms at different workflow arrival rates

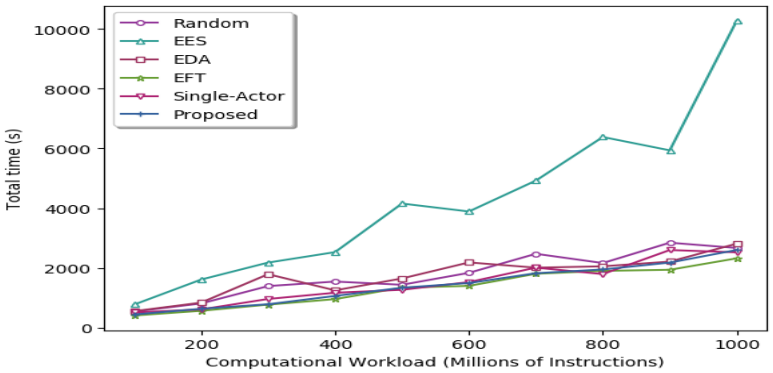
nication times leading to increased total execution time. An improvement of 11% and 47% over the execution times of Random and EES algorithms is achieved by the proposed DRL framework, respectively. EDA, Single-Actor and Proposed techniques have performed similar, with proposed technique taking slightly 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 4.5c and 4.5d respectively. EES algorithm has resulted in the lowest number of deadline hits, 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, 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 completion 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. Even-though the DRL agents of both Single-Actor method and proposed method are trained using the same reward structure, it is clear that the proposed method is more efficient at learning the training objective since it achieves more deadline hits while consuming less energy compared to the Single-Actor method.

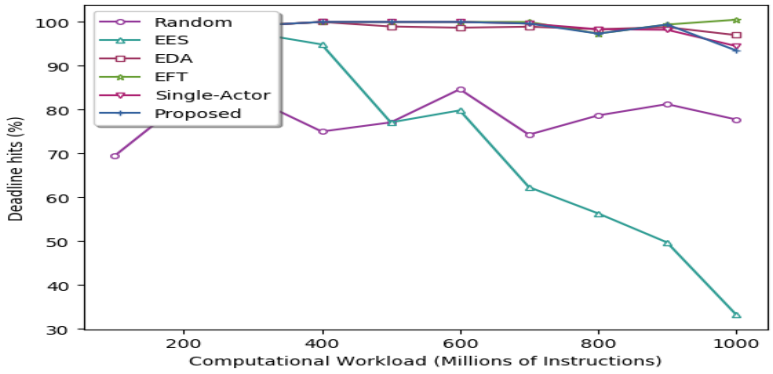
Proposed DRL method, Single-Actor method, and EFT as well as EDA techniques take predecessor proximity into account in the formulation of allocation decisions which is crucial particularly when it comes to workflows with communication intensive precedence relations. Large waiting times that accompanies precedence relations agnostic



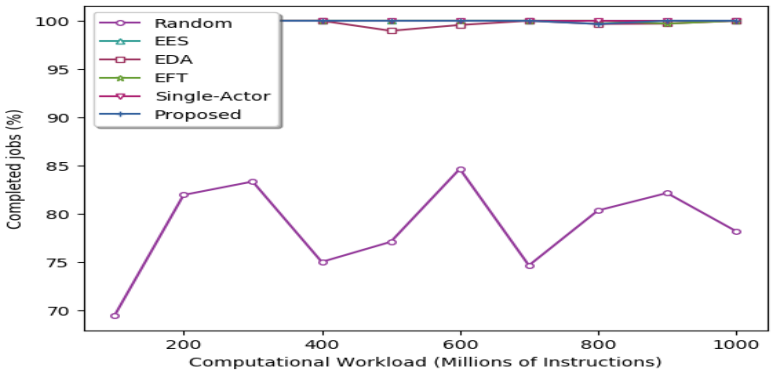

**(a)** Total energy consumption




**(b)** Total time




**(c)** Deadline hits




**(d)** Completed jobs

**Figure 4.7:** Comparison of performance of scheduling algorithms at different computational workloads

scheduling decisions given by Random and EES algorithms result in timed out tasks which in turn lead to incomplete jobs as shown in Figure 4.5d.

### 4.5.7 Analysis of Performance at Different Workflow Arrival Rates

Figure 4.6 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 4.6a 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, Single-Actor and proposed DRL technique. 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 4.6b 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 increasing the waiting times of task executions.

As indicated in Figure 4.6c and 4.6d, 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 the performance of both Single-Actor as well as proposed DRL techniques degrade slightly more compared to EFT and EDA techniques. This is expected behaviour since the DRL models were 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 [132]. 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 [132].

### 4.5.8 Analysis of Performance at Different Computational Workloads

Figure 4.7 demonstrates the performance of the algorithms as the computational workload varies. Computational workload in Figure 4.7 represents the aggregate computational workload of all workflows scheduled by the system. In Figure 4.7a, 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, Single-Actor and proposed DRL technique 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 4.7b since it has completed less jobs compared to other algorithms as indicated in Figure 4.7d.

In Figure 4.7c, EES algorithms ability to meet deadlines sharply drops with increasing workload. This is due to the fact that EES algorithm focuses solely on minimizing energy consumption, and therefore the allocation decisions it makes lead to increased execution delays. This in turn delays the completion of task executions leading to deadline misses. 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.

## 4.6 Overall Analysis

As evident through the results of experiments in Figures 4.5a, 4.6a and 4.7a, EES algorithm is the most efficient at minimizing energy consumption of workflow executions. This is because its scheduling decisions are solely focused on reducing the energy consumption and therefore, regardless of the rise in execution time it always allocates tasks to the most energy-efficient node. This however is achieved at the expense of increased execution time which is highly undesirable given the importance of low response times in use cases associated with edge computing environments. Results achieved with proposed DRL method is much similar to that with EES with respect to energy consumption. However, the proposed method is much superior to EES since it manages to minimize energy consumption without adversely impacting the total execution time as depicted in the experimental results in figures 4.5b, 4.6b and 4.7b. This is due to the fact, that the DRL agent in proposed method is trained to establish a balanced trade-off between energy consumption and execution time with the use of reward formulation presented in section 4.4.1.

The execution times of all other algorithms except EES are in a similar range with EFT technique always achieving the lowest execution times owing to the fact that its objective is solely designed to minimize response time. As expected, the percentage of deadline hits (figures 4.5c, 4.6c and 4.7c) follow a similar trend to execution time since meeting or missing deadlines is largely dependent on the time taken for task execution. Simply put, EES algorithm produces best results with respect to energy consumption and EFT algorithm produces best results with respect to execution time. Therefore the fact that the proposed DRL method has produced similar results to EES and EFT in terms of energy consumption and execution time, respectively demonstrate the superiority of the proposed technique at simultaneously achieving the conflicting objectives of minimizing energy consumption and execution time. The superior results thus obtained can be attributed to the multi-component reward used for training the DRL agent. Such behavior is particularly useful in edge computing environments where latency sensitive IoT workflows such as video surveillance are executed in edge nodes powered by batteries with limited capacities.

As previously mentioned, the Single-Actor method is similar to the proposed method in all aspects except that the proposed method has a hierarchical action space and uses separate actors for determining the task allocation tier and node. But as evident through the results of the experiments there is a significant difference between the two methods in terms of energy consumption, execution time as well as deadline hits achieved in the experimental use cases. This clearly highlights the advantage of using separate actor networks for determining the task allocation tier and node.

## 4.7 Summary

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 deep reinforcement learning 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 the 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 deep reinforcement learning framework agent learns to establish a balanced trade-off between latency and energy consumption. Results of simulation experiments demonstrate that the deep reinforcement learning 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 evidenced by the results the proposed method can be used for workflow schedul-

ing in highly dynamic and complex edge computing environments while minimizing energy consumption as well as execution time.

In the next chapter, we study the problem of workflow scheduling on distributed cloud datacenters in the presence of renewable energy. A majority of distributed cloud environments tend to be partially observable since it is unlikely that local information about all the clouds will be globally available at all times. Therefore, different from the fully observable agent environment considered in this chapter, the next chapter considers a partially observable environment in which the agent does not perceive the complete state of the environment, instead only a partial observation is visible. A decentralized multi-agent DRL framework that conforms to the paradigm of centralized training and distributed execution which is proven to be highly effective in multi-agent systems is proposed for green energy-optimized scheduling of workflows.

# Chapter 5

# Multi-agent Deep Reinforcement Learning Framework for Workflow Scheduling

*The ever-increasing demand for the cloud computing paradigm has resulted in the widespread deployment of multiple datacenters, the operations of which consume very high levels of energy. The carbon footprint resulting from these operations threatens environmental sustainability while the increased energy costs have a direct impact on the profitability of cloud providers. Using renewable energy sources to satisfy the energy demands of datacenters has emerged as a viable approach to overcome the aforementioned issues. The problem of scheduling workflows across multi-cloud environments powered through a combination of brown and green energy sources includes multiple levels of complexities. Firstly, the general case of workflow scheduling in a distributed system itself is NP-Complete. The need to schedule workflows across geo-distributed cloud datacenters adds a further layer of complexity atop the general problem. The problem becomes further challenging when the datacenters are powered through renewable sources which are inherently intermittent in nature. Consequently, traditional heuristic and meta-heursitic based algorithms and single-agent reinforcement learning algorithms are incapable of efficiently meeting the decentralized and adaptive control required for addressing these challenges. To this end, we have leveraged the recent advancements in the paradigm of MARL (Multi-Agent Reinforcement Learning) for designing and developing a multi-agent RL framework for optimizing the green energy utilization of workflow executions across multi-cloud environments. The results of extensive simulations demonstrate that the proposed approach outperforms the comparison algorithms with respect to minimizing energy consumption of workflow executions by 47% while also keeping the makespan of workflows in par with comparison algorithms. Furthermore, with the proposed optimizations, the multi-agent technique learnt 5 times faster than a generic multi-agent algorithm.*

This chapter is derived from:

- **Amanda Jayanetti**, Saman Halgamuge, Rajkumar Buyya, "Multi-agent Deep Reinforcement Learning Framework for Renewable Energy aware Workflow Scheduling on Distributed Cloud Datacenters", *IEEE Transactions on Parallel and Distributed Systems*, 2023 (under 2nd review).

## 5.1 Introduction

In recent times, cloud computing has become a frequently used platform for running resource-intensive workloads of commercial as well as scientific applications owing to its inherent ability of provisioning compute and network resources in an on-demand manner. Cloud computing services are offered by service providers via multiple geographically distributed datacenters for improving the Quality of Service (QoS) experienced by geographically dispersed consumers as well as for other requirements such as disaster recovery and high availability.

The excessive demand for cloud computing services has given rise to an exponential rise in the power consumption of cloud datacenters [74], hence adversely impacting the sustainability of the computing paradigm. Furthermore, high power consumption levels also increase the operational costs of datacenters which in turn reduces the profitability of service providers. As a step towards mitigating the adverse impacts of high power consumption and associated effects such as CO2 emissions, increasingly more renewable energy sources (such as solar power and wind power) are leveraged by cloud providers for satisfying the power requirements of cloud datacenters.

Geographically dispersed datacenters can be powered through locally available renewable energy sources thereby minimizing the use of brown energy. However, due to the intermittent nature of renewable energy sources, their utilization for powering cloud datacenters gives rise to several challenges. Variations in weather conditions and other location-dependent factors could drastically impact the levels of power generated by renewable energy sources. Distribution of workloads among geographically distributed datacenters while taking into account the renewable energy generation levels as well as diverse QoS requirements of heterogeneous workloads makes it possible to handle the intermittent nature of renewable energy sources in an efficient manner.

Workflow is a popular application model which can be used to represent a wide variety of workloads ranging from scientific to commercial applications. Cloud computing environments are widely used for the execution of resource-intensive workflows. As opposed to scheduling individual workloads, workflow scheduling in cloud computing environments is more complex due to the presence of precedence relations between

workflow tasks. The problem becomes even more challenging when workflows are to be scheduled across multiple geographically distributed datacenters some of which are operating on renewable energy sources. Only very few studies in existing literature have simultaneously attempted to tackle the aforementioned challenges. The scheduling frameworks proposed in these studies have either used heuristic or meta-heuristic techniques for achieving the desired objectives. These approaches generally suffer from problems such as high computational complexity, low adaptability to dynamically changing conditions, and so on.

Reinforcement Learning (RL) has emerged as a promising solution for dealing with highly dynamic and unpredictable problems. In particular, the more recent combination of Deep Neural Networks (DNN) with Reinforcement Learning which gave rise to the Deep Reinforcement Learning (DRL) paradigm, is proven to have the potential of solving complex problems in highly stochastic environments [133]. DRL agents operate with no prior knowledge of the environment, and they learn by interacting with the environment and gathering rewards for actions performed. The rewards enable the agents to learn desirable and undesirable behaviors in different situations, thus enabling them to identify the actions that result in the maximization of overall rewards.

DRL techniques have been used in a number of studies for handling resource management problems in cloud computing environments [134], [135]. However, a vast majority of these works have proposed straightforward adaptations of popular RL algorithms such as Deep Q Learning for designing single-agent scheduling frameworks that are more suited for satisfying the centralized scheduling requirements of the traditional cloud computing paradigm. Such approaches are not suitable when workloads are to be scheduled across more complex distributed infrastructures such as federated clouds and emerging fog computing environments [8]. Accordingly, Multi-Agent Systems (MAS) in which interactions between multiple agents are leveraged, are proven to be a better fit for problem-solving in highly distributed and stochastic environments compared to its single-agent counterpart [9].

The direct implementation of single agent RL algorithms in multiple agents leads to a non-stationary environment which prevents such methods from converging to an optimal solution [7]. Therefore the design of efficient multi-agent RL (MARL) algorithms

requires overcoming multiple challenges including nonstationarity and partial observability of the environment, scalability issues that arise as the joint action space grows with the increasing number of agents, and communication between agents which is particularly challenging in partially observable environments [7]. As the name implies, in partially observable environments, the agent does not perceive the complete state of the environment, instead only a partial observation is visible. The existing works that have proposed DRL-based scheduling techniques in multi-cloud environments have assumed full observability. However, realistically, a majority of multi-cloud environments tend to be partially observable since it is unlikely that local information about all the clouds will be globally available at all times.

To overcome the aforementioned challenges, and design an RL framework that can efficiently schedule workflows across partially observable and highly distributed multi-cloud environments, we leveraged the multi-agent coordination technique proposed by R. Lowe et. al [136]. In this technique, the traditional actor-critic method is extended to a multi-agent setting by providing the critic with extra information about the actions and observations of other agents during the training process. During execution, the agents operate based on local observations. Accordingly, the proposed technique conforms to the paradigm of centralized training and distributed execution which is proven to be highly effective in multi-agent systems [67]. A drawback of the aforementioned multi-agent coordination technique is that the Q function grows linearly with the number of agents, and this in turn could adversely impact the learning process of the agents. To overcome the potential impediment to agent learning, we leveraged domain-specific characteristics of the problem to design multi-agent coordination in a hierarchical manner. This in turn leads to the achievement of better training efficacy as evidenced by the empirical results from extensive simulations.

The following are the main contributions of this work:

- A hierarchical design for the workflow scheduling problem in which a global RL agent assigns tasks to datacenters and local RL agents assign tasks to nodes. Coupled with this, we present a novel formulation of the agent environment comprising state space, action space, and reward as a Partially Observable Markov Decision Process (POMDP)

- A MARL framework capable of scheduling workflows across the partially observable and highly distributed cloud environments in an efficient manner by sharing extra information during training, and operating solely based on local information during execution. Furthermore, we propose a shared reward structure that motivates agents to act cooperatively for achieving a common goal
- A novel approach to handling the curse of dimensionality and thereby improving training efficiency by leveraging the hierarchical nature of the proposed scheduler for limiting the observations shared by agents to a local neighborhood

The rest of the chapter is organized as follows: In section 2 we review relevant literature, and in section 3 we present the formulation of the workflow scheduling problem in multi-cloud environments. The proposed RL framework is presented in section 4 and its performance evaluation results are discussed in section 5. Finally, conclusion of this work is presented in section 6 along with future work.

## 5.2 Related Work

In this section, we review heuristic and meta-heuristic-based scheduling techniques as well as DRL-based scheduling techniques in multi-cloud environments.

A number of studies have focused on the problem of scheduling workflows across federated clouds [137–139]. In [137], a heuristic-based technique is proposed for scheduling workflows across multiple datacenters with the objective of minimizing electricity costs. They propose strategies for sorting workflows and sequencing workflow tasks, and also for allocating resources to tasks such that the resulting framework can achieve the desired objectives. [138], [139] also proposed algorithms for scheduling workflows across federated cloud datacenters with the objectives of minimizing cost and improving reliability.

Several works in existing literature have focused on the problem of scheduling workflows across geo-distributed datacenters powered through renewable energy sources [140], [141]. In [140], a hierarchical scheduling framework is proposed for scheduling workflows across multiple geo-distributed datacenters powered via renewable energy

partially. High-level scheduler allocates workflows to datacenters while a low-level scheduler assigns tasks of workflows to compute resources for achieving the overall objectives of improved energy efficiency, makespan, and deadline hits. [141] used a genetic algorithm to design a workflow scheduling framework that maximizes the use of renewable energy while minimizing the cost of electricity. The problem is formulated as a single objective optimization problem by multiplying the objectives and user-defined requirements such as deadline and budget. This work considers that the datacenters are interconnected through a privately owned Software Defined Wide Area Network (SDWAN).

Renewable energy aware scheduling of independent tasks/jobs across multi-cloud environments is also studied in a number of works [142], [143], [144]. In [142] Integer Linear Programming (ILP) is used for designing a task scheduling framework for minimizing the use of brown energy in datacenters while satisfying user-defined time constraints and electricity budgets. An obvious drawback of the proposed approach is the high time complexity associated with ILP based problem formulation. [143] used simulated annealing coupled with bees algorithm for scheduling tasks across distributed cloud computing environments with the objective of minimizing energy consumption. In [144] a job allocation algorithm is proposed to assign transactional and batch jobs to datacenters such that the power consumption of datacenters can be altered to suit variable conditions such as green energy availability. A batch job migration algorithm that migrates jobs among datacenters based on predicted green energy availability is also proposed. Jobs that are too large to be migrated are delayed until favorable energy levels are available ensuring no violation of deadlines take place. This work uses Q learning for tuning the progress of batch jobs.

While considerable research efforts have been focused on designing RL-based algorithms for scheduling workloads within a single datacenter [40], [124], [145], not much work has been done with RL in multi-cloud scheduling scenarios. Only a few works have leveraged the advanced capabilities of RL for designing workload scheduling algorithms across distributed cloud computing environments [146], [135]. In [146], Proximal Policy Optimization (PPO) [58] was used for designing a scheduling policy capable of determining if the job should be executed in a particular server of a private datacenter

or offloaded to a VM of a certain type in the public cloud depending on multiple factors including predicted renewable energy availability, deadline constraints, and cost. C. Xu et. al [135] proposed an RL-based algorithm for migrating jobs across multiple datacenters with the objective of minimizing energy cost.

The problem of scheduling workflows with complex data dependencies among tasks across multiple datacenters is inherently more complex due to the distributed nature of the underlying computing environment. A majority of existing studies have used single agent RL in workflow scheduling algorithms [80], [89] and these methods are designed to operate within a single cloud datacenter rather than across multiple datacenters. However, multi-agent RL is likely to be a better candidate for multi-cloud environments due to the need for decentralized decision-making. Regardless of the benefits of multi-agent RL in multi-cloud settings, the design of multi-agent systems are more complicated since the presence of multiple agents could make the environment nonstationary [7].

Despite the aforementioned complexities, multi-agent RL frameworks are likely to be more effective in such environments since they can be used for developing decentralized scheduling policies which are more suited for environments benefiting from decentralized control. However, none of the existing works have efficiently leveraged the power of multi-agent RL for scheduling workflows across multi-cloud environments.

## 5.3 System Model

Directed Acyclic Graphs (DAG) are used for modeling the workflows that are scheduled by the proposed hierarchical scheduling framework. The tasks of a workflow are represented by the set of nodes, $V = \{v_0, v_1..v_n\}$ and the precedence constraints between tasks are represented by the set of edges, $E = \{(v_i, v_j)|v_i, v_j \in V\}$ of a DAG, $G = (V, E)$.

Workflows are to be scheduled across a federation of geo-distributed datacenters $DC = \{dc_1, dc_2, ..dc_n\}$ which are powered through a combination of green energy from renewable energy sources and brown energy from the grid. Therefore at any instance, the total power consumption, $P_{total}$ of the datacenter federation is the sum of green power, $P_{green}$ and brown power, $P_{brown}$ consumed by the underlying cloud infrastruc-

| Work | Application Model | | Multi-Cloud | Algorithm | Renewable Energy Aware | Objectives |
|---|---|---|---|---|---|---|
| | Workflow | BoT/Job | | | | |
| [141] Z. Wen et al. (2020) | ✓ | | ✓ | Genetic Algorithm | ✓ | energy, cost |
| [140] S. Iturriaga et al. (2016) | ✓ | | ✓ | Genetic Algorithm | ✓ | energy, makespan, deadline |
| [137] L. Xiaoping et al. (2020) | ✓ | | ✓ | Heuristic | | electricity cost |
| [138] W. Zhenyu et al. (2016) | ✓ | | ✓ | Heuristic | | cost, reliability |
| [141] W. Zhenyu et al. (2016) | ✓ | | ✓ | Genetic Algorithm | | cost, reliability |
| [142] C. Gu et al. (2015) | | ✓ | ✓ | Integer Linear Programming | ✓ | minimizing carbon emissions |
| [143] Y. Haitao et al. (2020) | | ✓ | ✓ | Bees Algorithm | ✓ | energy |
| [139] Z. Wen et al. (2016) | | ✓ | ✓ | Genetic Algorithm, Heuristic | ✓ | cost, failure |
| [144] D. Cheng et al. (2020) | | ✓ | ✓ | Nonlinear programming, Q-Learning | ✓ | system goodput |
| [146] J. Zhao et. al (2021) | | ✓ | ✓ | PPO | ✓ | minimizing brown energy, deadline |
| [135] C. Xu et. al (2018) | | ✓ | ✓ | Deep Q Learning | ✓ | energy |
| [40] N. Liu et. al (2017) | | ✓ | | Deep Q Learning | | energy, latency |
| [124] D. Ding et. al (2020) | | ✓ | | Q Learning | | energy |
| [145] D. Cui et. al (2017) | | ✓ | | Q Learning | | makespan |
| [80] A. Nascimento et. al (2019) | ✓ | | | Q Learning | | makespan |
| [89] Z. Tong et. al (2020) | ✓ | | | Deep Q Learning | | makespan |
| Proposed | ✓ | | ✓ | Actor-Critic | ✓ | energy, makespan |

**Table 5.1:** A comparison of relevant literature with proposed work

tures and operations. The datacenter, $dc_i$ comprises of a set of $\lambda_i$ heterogeneous severs, $\{m_1, m_2, ..m_{\lambda_i}\}$. Power consumed by a server, $m_i$ is calculated using the CPU utilization-based power model presented in [111] as follows:

$$P_{m_i} = \begin{cases} P_{m_i}^{idle} + (P_{m_i}^{dynamic} - P_{m_i}^{idle}).u_{m_i}, & \text{if } u_{m_i} > 0 \\ 0, & \text{otherwise} \end{cases} \tag{5.1}$$

where $P_{m_i}^{idle}$ is the idle power consumption of the server which is a constant regardless of its current utilization and $P_{m_i}^{dynamic}$ is the dynamic power consumption which is dependent on the current server utilization, $u_{m_i}$. Accordingly the total power consumed by the datacenter, $dc_i$ during the $k$th time interval is computed as follows:

$$P_i^{total}(k) = \sum_{i=1}^{\lambda_i} P_{m_i}(k) \tag{5.2}$$

The objective of the proposed hierarchical scheduling framework is to minimize total brown energy utilization of the cloud datacenter federation while also optimizing workflow execution time. The total brown energy consumption at the datacenter $dc_i$ during the $k$th time interval can be represented as:

$$P_i^{brown}(k) = P_i^{total}(k) - P_i^{green}(k) \tag{5.3}$$

where $P_i^{green}(k)$ is the total green energy available at the datacenter $dc_i$ during the $k$th time interval. Accordingly, the primary objective of the global scheduler which mainly focuses on minimizing the brown energy usage during the $k$th time interval can be represented as follows:

$$\text{Min: } P_{total}^{brown}(k) = \sum_{i=1}^{N} P_i^{brown}(k) \tag{5.4}$$

Once a task is assigned to a datacenter $dc_i$, the local scheduler allocates the task to a server that jointly minimizes the total energy consumption and execution time of the task. Hence, the objective of the local scheduler during the $k$th interval can be represented as follows:

$$\text{Minimize: } \sum_{j=1}^{N} \alpha T_{t_j} + (1-\alpha) E_{t_j} \tag{5.5}$$

where $N$ is the total number of tasks executed during the $k$th time interval. $T_{t_j}$ and $E_{t_j}$ denotes the total execution time and total energy consumption associated with the execution of task $t_j$, respectively. The execution time $T_{t_j}$ includes the maximum data transfer time from predecessor nodes, computation time of the task as well as the waiting time of the task at the node ($WT_{t_j}$) before the task is actually executed. It can be computed as follows:

$$T_{t_j} = \frac{L_{t_j}}{F} + WT_{t_j} + \max_{t_i \in pred(t_j)} \left(\frac{D_{t_i,t_j}}{B}\right) \tag{5.6}$$

where, $L_{t_j}$ is the size of the task, $t_j$ and $F$ is the processing rate of the node to which task is assigned, and the ratio $\frac{L_{t_j}}{F}$ is the computation time of the task. The ratio $\frac{D_{t_i,t_j}}{B}$ is the data transfer time from the node in which predecessor $t_i$ executed to the execution node of task $t_j$, where $D_{t_i,t_j}$ is the size of data to be transferred from $t_i$ to $t_j$ and $B$ is the network bandwidth. Total energy consumed during the execution of task $t_j$ is computed as follows:

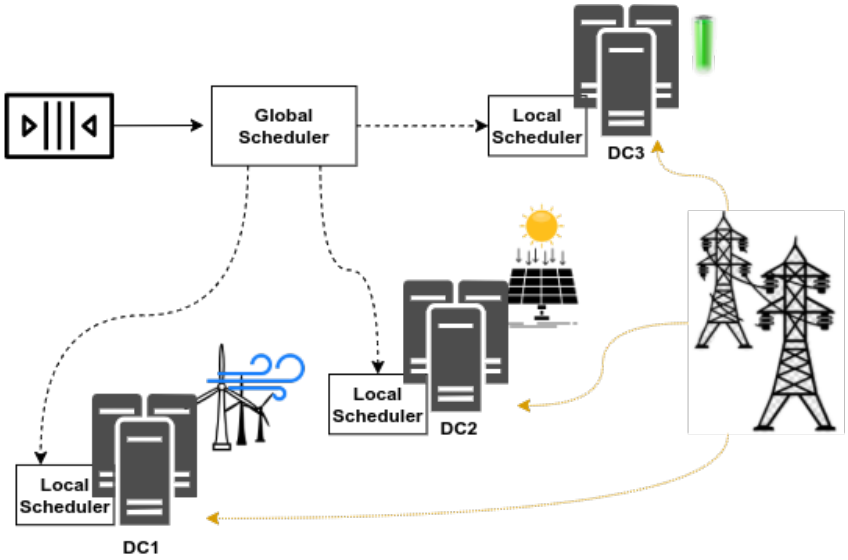



**Figure 5.1:** A high-level overview of workflow scheduling on distributed cloud data-centers

$$EE_{t_j} = T_{t_j} \times [U \times P_{active} + P_{idle}] \tag{5.7}$$

where $U$ is the current CPU utilization level of the execution node, and the rates of power consumption at active and idle states of the processors are denoted by $P_{active}$ and $P_{idle}$, respectively. Considering the power consumption associated with the transmission of data to be $P_{comm}$, the energy consumed during the transfer of data from predecessor nodes is computed as follows:

$$ET(t_j) = \sum_{t_i \in pred(t_j)} \frac{D_{t_i,t_j}}{B} \times P_{comm} \tag{5.8}$$

The total energy consumed $E_{t_j}$ is the sum of computation and communication energy, and is computed as:

$$ET(t_j) = EE_{t_j} + ET(t_j) \tag{5.9}$$

## 5.4 Reinforcement Learning

### 5.4.1 Background

**Reinforcement Learning (RL)** is a branch of the broader machine learning paradigm that operates by training an intelligent agent to learn a desired behavior in a given environment by learning through its interactions with the environment. The learning process is governed by the rewards which are received by the agent in return for the actions that it chooses to perform in the environment. Rewards serve as an indication of the degree of desirability of the action taken under the prevalent conditions toward achieving a predefined goal. RL problems are commonly modeled using the mathematical framework, Markov Decision Process (MDP).

At each decision epoch, the immediate situation the agent encounters, which is referred to as the current state ($s_t$) of the environment is taken into account by the agent for taking an action ($a_t$), which then results in a state transition from the current state ($s_t$) to the next state ($s_{t+1}$). Depending on the impact of the action on the environment, a reward ($r_t$) is given to the agent. Reward serves as a measure of the success of the agent's action in the given situation. As the agent progresses through the learning process, it learns to produce actions that result in the maximization of cumulative rewards over time. The strategy the agent employs to determine actions in this manner is called a policy ($\pi(a_t|s_t)$).

Despite the advanced capabilities of RL, the traditional RL paradigm suffers from the problem of dimensionality curse which makes its application to complex problems with very large state spaces practically infeasible. The combination of RL with deep learning which is referred to as Deep Reinforcement Learning (DRL) has successfully proven to overcome the aforementioned issue through function approximation, thereby eliminating the need for agents to visit all states during the training process and for storing state transition data in space-consuming tabular formats. Accordingly, a neural network is used for representing the policy ($\pi(a_t|s_t)$) as a parameterized function with respect to an adjustible parameter $\theta$. The resulting parameterized policy can be denoted as ($\pi_\theta(a_t|s_t)$).

**Policy Gradients** is a family of RL algorithms that operate by directly updating pol-

icy parameters for maximizing a performance objective, $J(\theta)$ that is defined as the expected cumulative discounted reward as shown in equation 5.10. This is achieved by repeatedly updating the policy parameters in the direction of the gradient of performance objective, $\nabla_\theta J(\theta)$. Gradient of performance objective, $J(\theta)$ can be expressed as in Equation 5.11.

$$J(\theta) = E_{\pi_\theta}[\sum_{t=1}^{\infty} \gamma^t r_t] \tag{5.10}$$

where $\gamma$ is a discounting factor that is used for discounting future rewards, and $\gamma \in (0, 1)$.

$$\nabla_\theta J(\theta) = E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) Q^{\pi_\theta}(s, a)] \tag{5.11}$$

where $Q^{\pi_\theta}(s, a)$ is the state-action value function (Q function) which indicates the desirability of an action, $a$ in a state, $s$ with a policy $\pi_\theta$. Different policy gradient algorithms use different techniques for estimating the Q function. For the scheduling problem addressed in this work, we use the Actor-Critic technique in a multi-agent scenario as described in the next section.

### 5.4.2 Proposed Multi-Agent Actor-Critic Scheduling Framework

In this section, we propose a multi-agent actor-critic framework for addressing the problem of scheduling heterogeneous workflows across geo-distributed datacenters whilst minimizing the use of brown energy. In actor-critic methods, the actor takes the current state of the environment as input and outputs a probability distribution over the actions that can be taken from this state. It does so by directly learning the policy function $\pi_\theta(a_t|s_t)$. Critic estimates the Q value $Q^{\pi_\theta}(s, a)$ based on the reward received for the action by the actor and the next state of the system. It then computes the Temporal Difference (TD) error which is used for updating the policy parameters of the actor-network in the direction of improvements, and for updating the network parameters of the critic so it can predict the Q function more accurately.

We propose a hierarchical scheduling framework for the workflow scheduling prob-

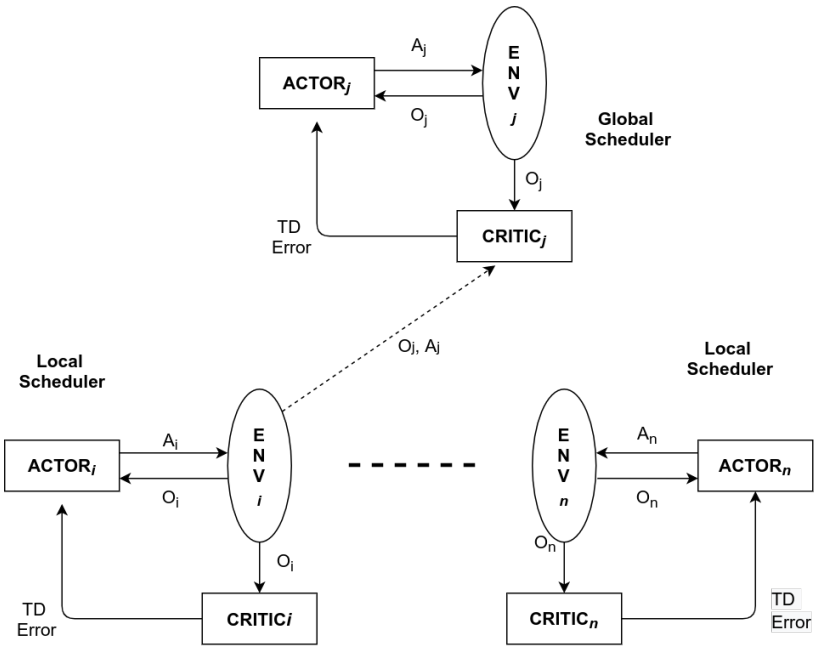



**Figure 5.2:** A multi-agent actor-critic architecture in which every critic is augmented with actions and observations of other agents

lem in which a global scheduler assigns workflow tasks to datacenters and the local scheduler in each datacenter assigns the tasks to the physical machines. Figure 5.1 shows a high-level overview of the proposed framework. A single-agent DRL framework is inappropriate for the aforementioned scenario owing to its inherently distributed nature. Therefore, in this work, we propose a multi-agent DRL framework in which one global agent acts as the global scheduler and multiple local agents act as local schedulers.

In comparison to single-agent problems, multi-agent problems are much more complex since the actions of other agents cause the environment to be non-stationary. Therefore the changes in the environment observed by an agent are not solely due to its own actions, but also due to the actions of other agents on the environment. Furthermore, due to the distributed nature of the multi-cloud environment, it is impractical to assume each agent has complete information about the real-time status of the entire environment. Partially Observable Markov Decision Process (POMDP) provides the flexibility of modeling the RL environment without requiring the agents to directly observe the actual state of the environment. Rather, what the agent receives is an observation which

is in fact a belief over the environment's actual state. In order to model the multi-agent DRL framework proposed in this work, we use Partially Observable Markov games [64]. A Markov game can be defined by a 7-tuple (N, S, $\phi$, $\{A_i\}$, P, $\{O_i\}$, $\{R_i\}$):

- N: A finite set of agents
- S: A finite set of states
- $\phi$: Initial state distribution
- $A_i$: A finite set of actions available to agent $i$
- P: State transition function which determines the probability of joint action $A_1 \times A_2 \times ... \times A_i$ in state $S_t$ leading to a transition to state $S_{t+1}$. The actions of each agent are governed by a policy $\pi_{\theta_i}$
- $O_i$: A finite set of observations of agent $i$
- $R_i$: S $\times$ $A_i \rightarrow \mathbb{R}$ is the reward function of agent i

High variance in gradient estimates is an inevitable weakness of naive policy gradients. This effect is intensified in multi-agent scenarios since the reward received by an agent may not be solely due to its own actions but also the actions of other agents [136]. Therefore in such a setting, ignoring the impact of the actions of other agents, and conditioning the rewards only on agents' own actions inevitably leads to high variability which in turn results in gradient updates with high variance. Since naive policy gradients are not capable of handling multi-agent problems efficiently, we adapt the decentralized actor and centralized critic technique proposed in [136] for the multi-agent setting in our problem. This method focuses on providing the critic with extra information, at training time, about the policies of other agents. Accordingly, in a Markov game with N agents, the gradient of the performance objective of agent $i$, $J(\theta)$ in equation 5.11 can be expressed as follows:

$$\nabla_{\theta_i} J(\theta_i) = E_{\pi_{\theta_i}}[\nabla_{\theta_i} \log \pi_{\theta_i}(a_i|O_i) Q^{\pi_{\theta_i}}(x, a_1, a_2, ..., a_N)] \tag{5.12}$$

**Algorithm 6** Actor-Critic based training process of global scheduler

Initialize actor network $\pi_\theta(a|s)$ and critic network $Q_\omega(s,a)$ with random weights
**for** episode = 1 to $N$ **do**
    Reset the environment
    Input the initial state of the environment to actor network $\pi_\theta(a|s)$
    **for** step = 1 to $T$ **do**
        Select action $a_{global}$ from the actor-network based on the current policy $\pi_\theta(a|s)$, and observe the corresponding reward $R_{global}$
        $R_{local}, a_{local}$ = SendTaskToSelectedLocalScheduler($t_j$)
        $R_t = R_{global} + R_{local}$
        $a_t = a_{global} + a_{local}$
        Update network parameters of critic
        Update network parameters of actor
**return**

In the above equation, $Q^{\pi_{\theta_i}}(x, a_1, a_2, ..., a_N)$ is the Q function estimated by the critic. Note that different from the Q function of the naive actor-critic technique which takes as inputs the state of the agent and the action, in this case along with the state of the environment (x) the critic takes as input the actions $(a_1, a_2, ..., a_N)$ of all agents as well. The state of the environment, x may include the observations of all the agents and any additional state information. With this approach, each agent could have different reward structures since Q functions are learned separately.

Figure 5.2 shows the aforementioned multi-agent coordination technique. One obvious drawback of this method is the input space of the Q function increases with the increasing number of agents. To overcome this problem we leverage the characteristics of the hierarchical scheduler design proposed in this work. Accordingly, we provide the Q function of each local agent only its observation and action since the actions and observations of other local agents have no impact on the reward it receives or the next observation. To the Q function of the global agent, at each scheduling step, we only provide the action of the local agent to which it assigns the current task in addition to its own action and observation. Figure 5.3 shows the proposed multi-agent coordination technique that restricts communications to a local neighborhood. Limiting the shared experiences to a local neighborhood in this manner allows us to prevent the expansion of the input space of all agents substantially. Where local agents have different dimensions for state space and action space, a state encoding technique [147] can be used to

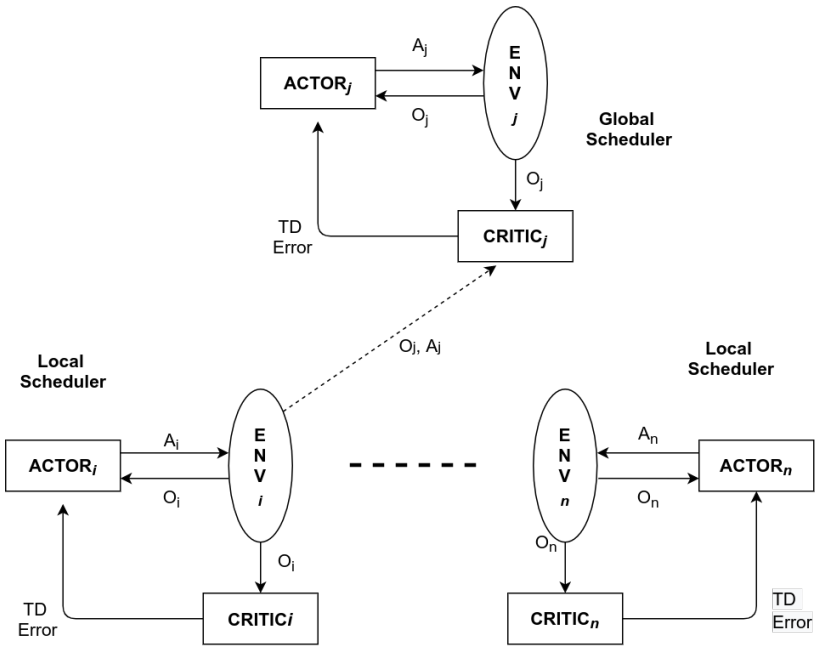



**Figure 5.3:** Proposed multi-agent actor-critic architecture where shared actions and observations are limited to a local neighborhood

extract a representation with a fixed dimensionality for providing as an input to the state information of the global agent.

**Global Scheduler** A workflow consists of multiple tasks the execution of which is constrained by precedence relations. Therefore when a workflow is submitted, all the tasks in it cannot be scheduled for execution at once. The global scheduler identifies the tasks that can be executed directly and the rest of the tasks will be pending execution until the tasks that they have precedence relations with complete execution. Upon the receipt of a task completion notification, the global scheduler executes the DRL agent multiple times with each of the tasks that can now be scheduled for execution due to their precedence relations being satisfied. The state of the global agent comprises of the green energy surplus or deficit levels of the datacenters, the average processing speed and current utilization level of the datacenters and the resource requirements of the task. Action corresponds to the selection of a datacenter (hence a local scheduler) to which the task will be submitted for execution. The reward comprises of two components; The first component corresponds to the current green energy deficit or surplus of the selected

**Algorithm 7** Actor-Critic based training process of local schedulers

Initialize actor network $\pi_\theta(a|s)$ and critic network $Q_\omega(s, a)$ with random weights
Reset the environment
Input the initial state of the environment to actor network $\pi_\theta(a|s)$
**for** every $t_j$ assigned by global scheduler **do**
    Select action $a_t$ from the actor-network based on the current policy $\pi_\theta(a|s)$
    Execute action $a_t$ and observe the corresponding reward $R_t$ and next state of the system $s_{t+1}$
    Update network parameters of critic
    Update network parameters of actor
    SendRewardAndActionToGlobalScheduler($a_t$, $R_t$)
**return**

datacenter and the second component corresponds to the reward received by the local scheduler that allocated the task to a node for execution. Incorporation of a component that reflects the desirability of the local agent's action in the global agent's reward in this manner enhances the learning process of the global agent. Algorithm 6 summarizes the steps included in the training process of the global scheduler.

**Local Scheduler** At each cloud datacenter, the tasks that are submitted to it by the global scheduler are immediately scheduled for execution. If the DC has no free capacity for task execution, then the global scheduler is notified. The allocation of a task to a node is an action performed by the local DRL agent, based on the characteristics of the task and the status of servers in the datacenter provided to it through state information. Upon the allocation of a task to a server, an immediate reward is received by the local agent which reflects the success of the allocation with respect to the objective, which in this case is energy efficiency and time minimization. The global scheduler is notified upon the completion of task execution. Since the reward received by the local agent also contributes to the global agent's reward, it is communicated along with the results of execution to the global scheduler. The state of the local agent comprises of the processing power and utilization levels of the servers and task size. Action is the selection of a server in which the task will be executed. The reward is a weighted function of the execution time of the task and corresponding energy consumption as indicated in equations 6.3 and 5.9, respectively. Algorithm 7 summarizes the steps included in the training process of the local schedulers.

| Server Name | Processor | Cores | MIPS | RAM (GB) | Bandwidth (GB/s) | Power (Watts) | |
|---|---|---|---|---|---|---|---|
| | | | | | | Idle | Active |
| Dell Inc. PowerEdge R740 | Intel Xeon Platinum 8280 2.70 GHz | 56 | 604.8k | 64 | 1.5 | 50 | 432 |
| PowerEdge C6100 | Intel Xeon X5675 3.06 GHz | 48 | 587.52k | 64 | 1.5 | 227 | 895 |
| Dell PowerEdge R820 | Intel Xeon E5-4650L 2.60 GHz | 32 | 332.8k | 64 | 1 | 110 | 446 |
| IBM System x iDataPlex dx360 M2 | Intel Xeon X5570 2.933 GHz | 16 | 187.712k | 48 | 1 | 116 | 475 |
| Dell PowerEdge R710 | Intel Xenon 5675 3.06 GHz | 12 | 146.88k | 64 | 1 | 62 | 227 |

**Table 5.2:** Host configurations derived from SPEC benchmark [3] for experimental setup

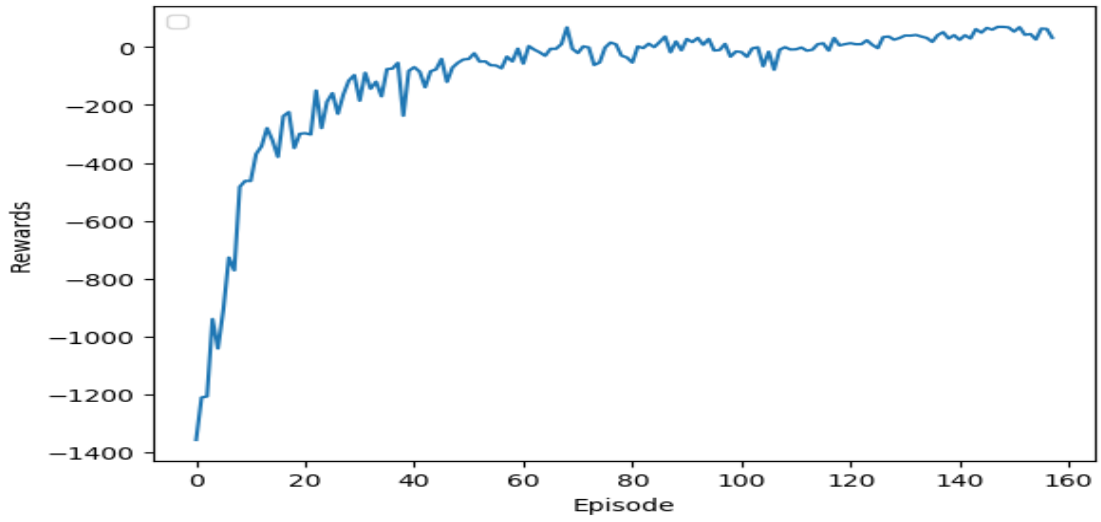
## 5.5 Performance Evaluation

### 5.5.1 Experimental Setup

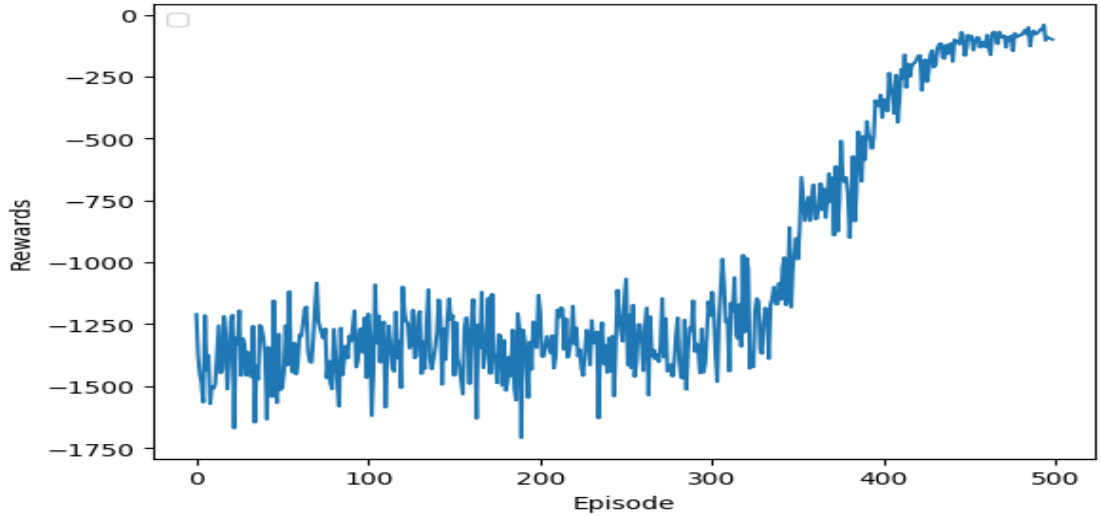
The proposed multi-agent deep reinforcement learning-oriented workflow scheduling framework was tested using an extension of the CloudSim simulation toolkit [11]. For deep learning-related implementation, Keras library was integrated with the simulation environment [12]. For the simulation environment, we used 10 datacenters each with 25 heterogeneous servers. The resource specifications of the servers including processing power, memory, power consumption etc. are derived from the popular SPEC benchmark [3]. Table 5.2 indicates the specifications of the servers used in the experiments. Renewable energy data for the simulations were obtained from the actual solar energy generated by multiple photovoltaic (PV) sites installed at the Gatton campus of the University of Queensland [**?** ].

### 5.5.2 Dataset

For evaluation of the proposed algorithm, a dataset comprising of 1000 workflows was derived from the synthetic workflow structures provided by the popular Peagusus workflow framework [2]. The size of data dependencies among tasks and the sizes of tasks were randomly selected from the ranges 0.1k to 10k and 0.5k to 1000k respectively. A




**(a)** Reward convergence with proposed DRL framework




**(b)** Reward convergence with a generic multi-agent DRL technique

**Figure 5.4:** Comparison of learning efficiency of the proposed framework and a generic algorithm (as evidenced through the number of episodes required for reward convergence)

Poisson distribution was used for modeling workflow arrival times.

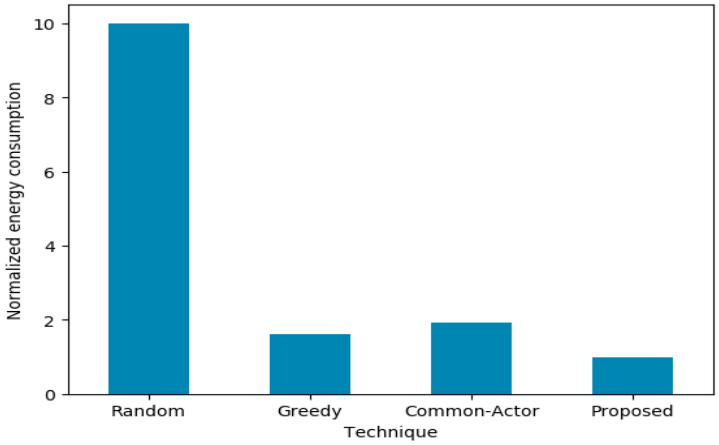
### 5.5.3 Comparison Algorithms

For evaluating the performance of the proposed multi-agent framework, three comparison algorithms were used. **Random** algorithm is a baseline that allocates workflow tasks to datacenters in a random manner. In the datacenters, the selection of hosts for task execution is also performed randomly without any consideration on the impact of such allocations on execution time or energy consumption. **Green-Opt** is an algorithm that operates with the objective of minimizing brown energy usage by allocating workflow tasks to the datacenters with the highest accumulated green energy. The algorithm is designed to favor task allocations to 'active' hosts with sufficient capacity for executing new tasks. The selection of active hosts rather than idle ones leads to higher energy savings which are reflected in the results of the experiments in the next section. The third comparison algorithm **Common-Actor** is a DRL-based multi-agent actor-critic algorithm. As the algorithm name implies, multiple actor networks are guided by a common critic.

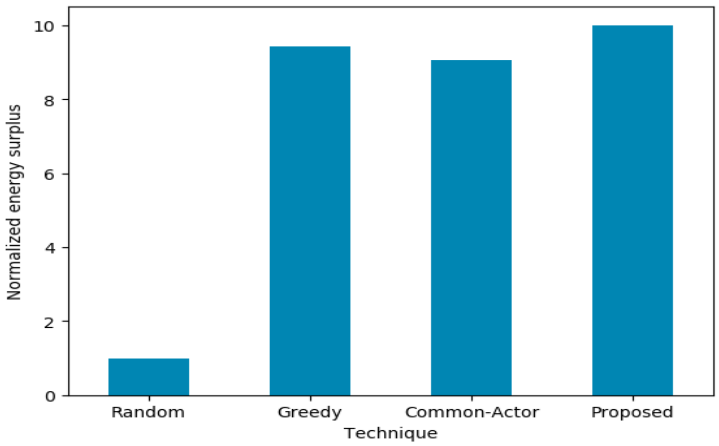
### 5.5.4 Experimental Results

The number of episodes required for a reinforcement learning algorithm to converge directly impacts the training time of the model. In order to ensure that the learned model remains up-to-date with the highly dynamic conditions in multi-cloud environments, it may be required to train and re-train the reinforcement learning agents incorporated in the resource schedulers. Therefore, convergence speed is an important factor that should be taken into account when designing reinforcement learning-oriented cloud resource schedulers. As evidenced through Figure 5.4, the local neighborhood-based multi-agent method proposed in this work is capable of achieving a significantly improved convergence speed in comparison to generic reinforcement algorithms. The generic multi-agent DRL technique has required 500 episodes of training for convergence, whereas the proposed model has converged in less than 100 episodes, this proves that the learning efficiency of the local neighborhood-based approach is more than five times better than that of the generic approach.

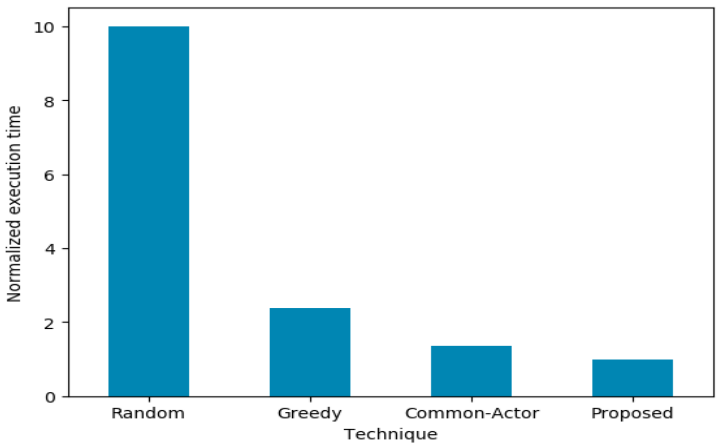
The performance of the algorithms was evaluated in three different experimental




**(a)** Total energy consumption

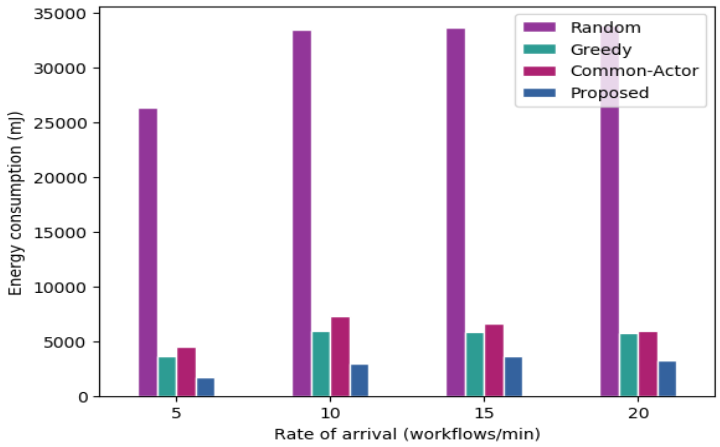



**(b)** Total energy surplus

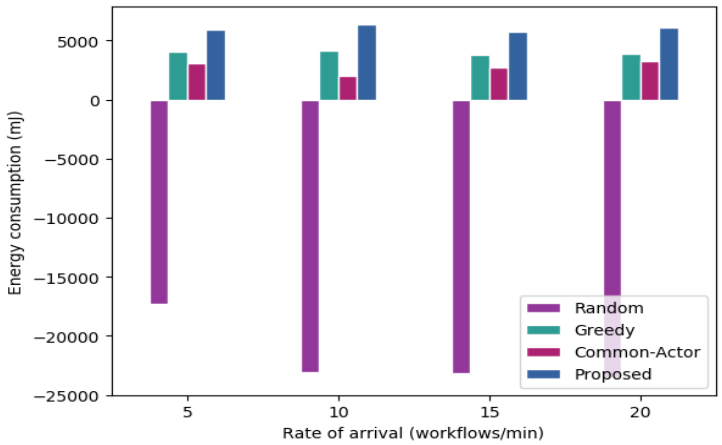



**(c)** Total time

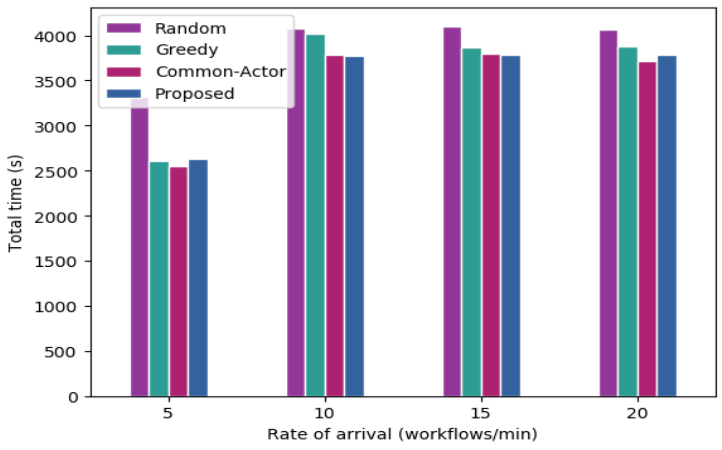
**Figure 5.5:** Comparison of performance of scheduling algorithms on an experimental dataset derived from the synthetic workflow structures provided by the popular Pegasus workflow framework [2]




**(a)** Total energy consumption




**(b)** Total energy surplus



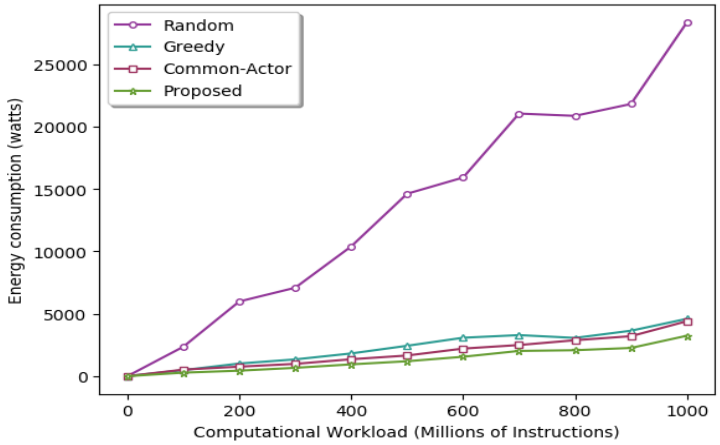

**(c)** Total time

**Figure 5.6:** Comparison of performance of scheduling algorithms as workflow arrival rate varies

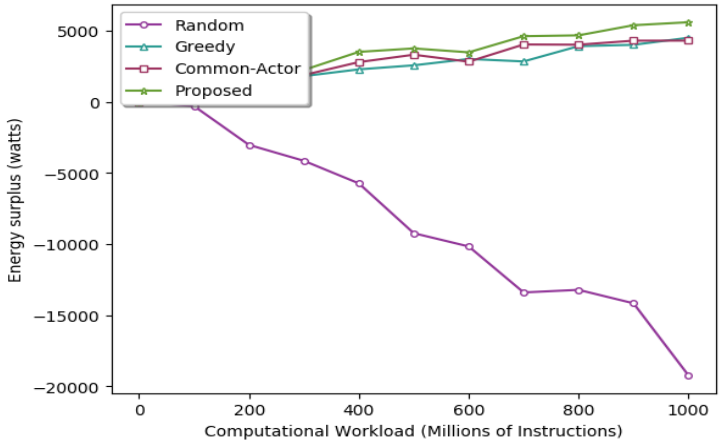
workload settings with respect to total energy consumption incurred during workflow executions, the total energy surplus of all the datacenters post to the execution of workflows and the total time taken for workflow executions, and the experimental results are presented in 5.5-5.7.

Graphs in Figure 5.5 depict the performance of the algorithms on the experimental dataset. The Random algorithm has consumed the highest amount of energy since it completely disregards the impact of allocations on energy-efficiency as well as time. Accordingly, as demonstrated in Figure 5.5b and 5.5c, workflows scheduled with Random algorithm results in the lowest energy surplus and consume the longest execution duration, respectively. In comparison to the Random algorithm, the Green-Opt algorithm has produced much better results in energy consumption and surplus. This is the expected behavior since it operates with the greedy objective of minimizing brown energy consumption. And also, the selection of active hosts over idle ones further contributes to improving energy-efficiency. Common Actor and Proposed Algorithms both use the same reward structures and as previously mentioned the difference between the two is that in the common actor method, one critic network is used to guide multiple actor networks. As evident through the results, the performance of the proposed algorithm is better than all comparison algorithms with respect to all three metrics. This is because the proposed multi-agent reinforcement algorithm is capable of finding the most efficient balance between energy consumption and execution time during the training process.

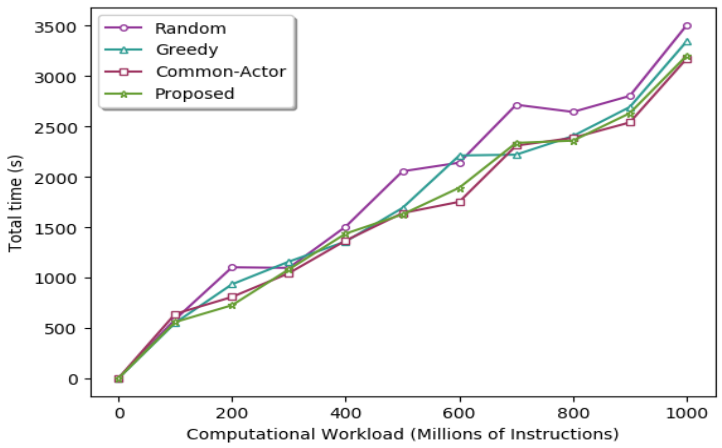
Graphs in Figure 5.6 demonstrate the performance of the algorithms as the workflow arrival rate varies. The Random algorithm has again failed to deliver favorable results signaling the importance of more fine-tuned algorithms for scheduling workflows in cloud computing environments. It is clearly evident that the proposed algorithm significantly outperforms all the other algorithms at all arrival rates with respect to energy-efficiency while also maintaining comparable performance with respect to total execution time. This indicates the fact that the learned model is capable of adapting the scheduling decisions to perform equally well under highly dynamic conditions. The difference in performance between the common actor and the proposed method clearly highlights the fact that the multi-agent coordination achieved through the proposed lo-




**(a)** Total energy consumption




**(b)** Total energy surplus




**(c)** Total time

**Figure 5.7:** Comparison of performance of scheduling algorithms as the size of computational workload varies

cal neighborhood-based technique leads to better learning which in turn leads to better performance.

Figure 5.7 depicts the performance of the algorithms as the computational workload varies. In the Random algorithm, very significant degradation in energy-efficiency is observable at high workloads. All the other algorithms are better capable of maintaining energy consumption levels at considerably moderate levels despite the increasing workload. The proposed algorithm has yet again managed to outperform all the other algorithms with respect to energy consumption as well as energy surplus, while performing equally well with respect to time total execution time.

## 5.6 Summary

We proposed a hierarchical multi-agent scheduling framework for scheduling workflows across geo-distributed cloud datacenters with the objectives of minimizing brown energy usage, while also keeping execution times in par with comparison algorithms. The agent environment is modeled as a POMDP, and the paradigm of centralized training and distributed execution is adopted by allowing the agents to share extra information during training and operating solely based on local information during execution. Furthermore, a novel approach for limiting observation sharing to a local neighborhood is presented for overcoming the curse of dimensionality and thereby improving training efficiency. As evidenced through the empirical results, the incorporation of domain-specific characteristics for designing the multi-agent coordination in a local neighborhood-oriented manner reduced training time by 5 times compared to a generic multi-agent technique. The results also clearly demonstrated that the proposed algorithm outperformed the generic DRL algorithm with respect to minimizing total energy consumption by 47%, while outperforming the baseline algorithms with even larger margins. In the next chapter, we design a DRL framework for cost optimized workflow scheduling in cloud environments and integrate it with an open source container-native workflow engine.

# Chapter 6

# Cost Optimized Workflow Scheduling in Cloud Computing Environments

*Cost optimization is a common goal of workflow schedulers operating in cloud computing environments. The use of spot instances is a potential means of achieving this goal, as they are offered by cloud providers at discounted prices compared to their on-demand counterparts in exchange for reduced reliability. This is due to the fact that spot instances are subjected to interruptions when spare computing capacity used for provisioning them is needed back owing to demand variations. Also, the prices of spot instances are not fixed as pricing is dependent on long term supply and demand. The possibility of interruptions and pricing variations associated with spot instances adds a layer of uncertainty to the general problem of workflow scheduling across cloud computing environments. These challenges need to be efficiently addressed for enjoying the cost savings achievable with the use of spot instances without compromising the underlying business requirements. To this end, in this chapter we use Deep Reinforcement Learning for developing an autonomous agent capable of scheduling workflows in a cost efficient manner by using an intelligent mix of spot and on-demand instances. The proposed solution is implemented in the open source container native Argo workflow engine that is widely used for executing industrial workflows. The results of the experiments demonstrate that the proposed scheduling method is capable of outperforming the current benchmarks.*

## 6.1 Introduction

Cloud computing leverages virtualization techniques for providing users with convenient access to a pool of scalable resources. As opposed to maintaining their own computing infrastructures, the pay-as-you-go model of cloud computing paradigm enables users to acquire a diverse range of virtual machines with varying flavors (CPU, Memory etc.) for meeting business needs in a more cost effective manner. The flavor of vir-

tualized instances used for executing tasks determines the total execution times of the workflows as well as the associated monetary costs. In order to maximize the achievable cost savings achievable while also ensuring the performance is maintained to a satisfactory level, it is imperative that cost optimized scheduling strategies are designed and implemented.

In particular, the intelligent use of a mix of on-demand and spot instances for workflow executions is a potential means of achieving high cost efficiencies without adversely affecting performance expectations. Spot instances are offered by cloud providers at steep discounts compared to their on-demand counterparts in exchange for reduced reliability. This is because the cloud providers utilize spare computing capacities available for provisioning spot instances, and therefore when the capacity is needed back, the instances are interrupted. Furthermore,as opposed to on-demand instances with fixed prices, the prices of spot instances are not guaranteed to be fixed, as the pricing is dependent on long term supply and demand. The possibility of interruptions and pricing variations adds a layer of complexity that needs to be efficiently handled for enjoying the cost savings without compromising the underlying business requirements. Therefore, it is imperative to establish the right balance between the use of on-demand and spot instances for workflow executions in cloud computing environments.

The ability of Reinforcement Learning (RL) agents to operate in stochastic environments, and learn through experience to act in an optimal manner amid highly dynamic conditions and uncertainties makes it an ideal candidate for overcoming the aforementioned challenges. While many heuristics and meta-heuristics have been proposed for cost optimized workflow scheduling, only very few works have explored the potential of RL in this area. In particular, Deep Reinforcement Learning (DRL) has emerged as an efficient means of solving highly complex problems as evidenced by the recent successes achieved by DRL agents in complex control tasks in fields such as robotics, autonomous driving, healthcare and so on. In this work, we leverage the advanced capabilities of DRL for designing a cost optimized workflow scheduling framework.

The design of action space is a fundamental characteristic of a DRL based formulation of a problem. The action spaces of a vast majority of scheduling problems that are modeled as DRL problems, include a flat set of actions. The action space may be discrete

or continuous, and the agent selects an action from the action space. In this work, we propose a novel hierarchical way of designing the action space of the DRL model such that there is a clear distinction between on-demand and spot instances in action selection. A DRL framework comprising multiple actor networks guided by a common critic network is then designed to select a combination of actions from the hierarchical action space, to optimize cost of workflow executions.

Container orchestration engines such as Kubernetes can seamlessly operate atop highly distributed and heterogeneous infrastructures and abstract away the complex coordination details from users. This in turn has enabled users to conveniently deploy workloads across a variety of cloud deployments ranging from private and public clouds to hybrid combinations of these. Complementary frameworks such as Argo workflow engine have emerged to extend the functionalities of Kubernetes to facilitate the management of more complex workloads such as Workflows. The schedulers of these frameworks are pre-configured to follow basic scheduling policies such as bin-packing. These simple policies are not capable of satisfying the complex cost optimization requirements of users. In order to achieve complex user-defined goals it is imperative to incorporate more advanced scheduling policies in the aforementioned workflow management engines. These policies should be capable of adapting to highly stochastic conditions that are inherent in clusters deployed in cloud computing environments. In this regard, we present an end-to-end means of training and deploying the DRL agent proposed in this work in the Argo workflow engine.

More specifically, the following summarizes the main contributions of this work:

- A DRL model for cost optimized workflow scheduling in a cloud computing environment with the use of a balanced mix of on-demand and spot instances
- A logical organization of the cluster in a hierarchical manner, along with a novel representation of the action selection process as a tree structure
- A RL framework with multiple actors guided by a single critic network trained with Proximal Policy Optimization (PPO) algorithm for learning to schedule workflows in the cluster

- An end-to-end means of training and deploying the proposed DRL agent in a workflow engine. To the best of our knowledge, this is the first attempt at embedding an intelligent agent in an open source container-native workflow engine

## 6.2 Problem Formulation

The objective of the scheduling framework is minimizing the monetary cost of workflow executions, while also minimizing the execution times. The resource requirements in terms of CPU and memory and the dependencies of workflow tasks are included in the submitted workflow specifications. In the workflow specification submitted by users, a workflow is represented by a DAG, $G = (V, E)$ where the nodes, $V = \{v_0, v_1..v_n\}$ of the DAG represent workflow tasks, and the precedence constraints between tasks are represented by the edges, $E = \{(v_i, v_j)|v_i, v_j \in V\}$. The computation time of a task, $t_j$ can be represented as:

$$CT(t_j) = \frac{L(t_j)}{F} \tag{6.1}$$

where $L(t_j)$ is the size of task, $t_j$ and $F$ is the processing rate of the node to which is it assigned. All the precedence constraints of task, $t_j$ must be satisfied before its execution commences. Accordingly, the execution of all the predecessors must be completed, and the output data required for the execution of $t_j$ must be transmitted to the node in which it is scheduled. If $t_i$ is an immediate predecessor of $t_j$ and the size of data to be transferred from $t_i$ to $t_j$ is $D(t_i, t_j)$, then the total transmission time ($TT$) can be denoted as follows:

$$TT(t_i, t_j) = \frac{D(t_i, t_j)}{B} \tag{6.2}$$

where $B$ is the bandwidth between the execution nodes of $t_i$ and $t_j$. Task execution delay, $TD(t_j)$ primarily depends on the computation time, $CT(t_j)$ of the task, and the maximum data transfer time from predecessor nodes, $\max_{t_i \in pred(t_j)} TT(t_i, t_j)$. The waiting time, $WT(t_j)$ before a task gets scheduled also contributes to total execution delay. Accordingly, $TD(t_j)$ can be represented as:

$$TD(t_j) = CT(t_j) + WT(t_j) + \max_{t_i \in pred(t_j)} TT(t_i, t_j) \tag{6.3}$$

The finish time, $FT(t_j)$ of task, $t_j$ that started execution at time, $ST(t_j)$ can then be expressed as:

$$FT(t_j) = ST(t_j) + TD(t_j) \tag{6.4}$$

The completion time, $MT$ of a workflow is equivalent to the time at which that last task of the workflow completes execution. It can be denoted as:

$$MT = \max_{t_j \in T}(FT(t_j)) \tag{6.5}$$

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

The computation cost of $t_j$ that executes in a Node with unit cost per second, $UC$ can be represented as:

$$CC(t_j) = CT(t_j) * UC \tag{6.6}$$

The cost of execution, $MC$ of a workflow is equivalent to the sum of execution costs of all tasks, and it can be denoted as follows:

$$MC = \sum_{t_j \in T} CC(t_j) \tag{6.7}$$

The objective of the scheduling problem is to minimize the cost of workflow executions, and it can be denoted as follows:

$$\text{Minimize: } \sum_{i=1}^{N} MC_i \tag{6.8}$$

where $N$ is the total number of workflows submitted to the system.

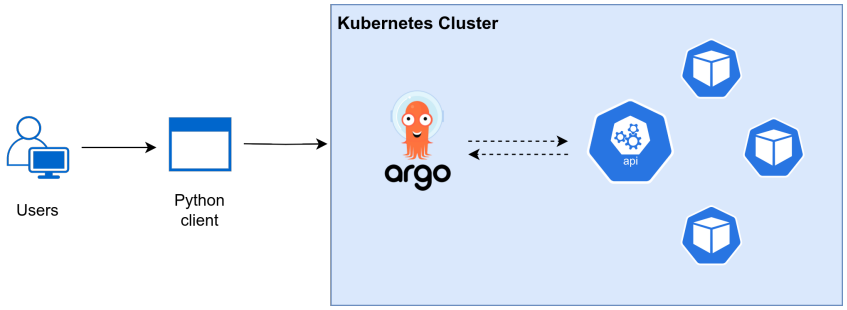



**Figure 6.1:** System Architecture

## 6.3 Proposed approach

In this section, we present a background of the popular container orchestration engine Kubernetes and the open-source Argo workflow engine along with details on how the proposed DRL framework is implemented in the Argo Workflow engine that runs atop the Kubernetes cluster. Worker nodes of the Kubernetes cluster are Virtual Machines with different flavors (compute, memory, and storage capacity of VM instances). Argo workflow engine is deployed in the Kubernetes cluster for the management of workflows submitted by users. The scheduler is responsible for selecting the VMs in which the Pods corresponding to each task of the workflow will be scheduled. A high level architecture of the system is shown in Figure 6.1.

### 6.3.1 Kubernetes

Kubernetes is a popular open-source container orchestration engine that facilitates containerized applications to be deployed, scaled, and managed in an automated manner. With Kubernetes, containerized workloads can be conveniently deployed and managed in any infrastructure including public clouds and on-site deployments, as well as hybrid combinations of these as required. Workloads can be seamlessly deployed across multi-cloud environments thus enabling the selection of the most appropriate infrastructure for the execution of different parts of the workload. Furthermore, it facilitates the up-scaling and down-scaling of clusters to suit demand variations of applications, which in turn helps reduce costs due to reduced resource wastage. The need for manual intervention is minimized since Kubernetes monitors the health of the deployment and

redeploys new containers in the event of a failure to restore operations, and this helps reduce application downtime. Owing to the multitude of benefits offered by Kubernetes, it has become the defacto platform for the deployment and management of containerized workloads. In this work, we extend the capabilities of the default Kubernetes scheduler by incorporating intelligence into it with the use of RL techniques.

A Kubernetes cluster consists of a set of virtual or physical machines which are referred to as Nodes. The smallest unit deployable in Kubernetes is referred to as a Pod. Pods are hosted by Nodes. A Pod may comprise one or more tightly coupled containers that share storage and network resources, it also contains a specification of how the containers are to be run. The contents of a Pod run in a shared context, and are always located and scheduled together. Pods and Nodes of a Kubernetes cluster are managed by the control plane. It comprises multiple components that work together for managing the cluster. Kube-api server exposes the Kubernetes API that serves as the front end of the Kubernetes control plane. Cluster data are stored in a key-value store termed etcd. The kube-controller-manager runs several controller processes that monitor and regulate the cluster state. Cloud-controller-manager handles cloud-specific control logic. Kube-scheduler is responsible for scheduling unassigned Pods to Nodes for execution.

### 6.3.2 Argo Workflow Engine

Argo workflow engine is an open-source container-native workflow engine that facilitates the orchestration of workflows on Kubernetes. Argo workflows are implemented as a Custom Resource Definition (CRD) in Kubernetes. This enables Argo workflows to be managed using kubectl and they integrate natively with Kubernetes services including secrets, volumes and Role Based Access Control (RBAC).

The workflow engine comprises two main components: the Argo server and the workflow controller. The Argo API is exposed by Argo server and the controller performs workflow reconciliation. In the reconciliation process, the workflows that are queued based on additions and updates to workflows and workflow pods, are processed by a set of worker goroutines. The controller processes one workflow at a time. Both Argo server and controller run in the Argo Namespace.

Each task of workflow results in the generation of a Pod. Each pod includes three containers. The main container runs the image that the user has configured for the task. The init container is an init container that fetches artifacts and parameters and makes them available to the main container. Wait container performs tasks related to clean up including the saving of artifacts and parameters.

Argo provides multiple templates for defining workflow specifications and dependencies. For example, a workflow can be defined as a sequence of steps. Alternatively, DAGs can be used for defining a workflow and its dependencies. As this facilitates the representation of complex workflows and parallelism, in this work we have used DAGs for modeling workflows.

A workflow specification comprises a set of Argo templates, each with an optional input section, an optional output section, and either a list of steps where another template is invoked by each step or a container invocation (leaf template). The options accepted by the container section of the workflow specification are the same options as the container section of a Pod specification.

### 6.3.3 Reinforcement Learning

In the RL paradigm, an agent learns in a trial-and-error manner by interacting with the environment. The agent receives a *reward*, $r_t$ when it performs an *action*, $a_t$ in a particular *state* $s_t$, and then the *environment* transitions to the next state, $s_{t+1}$. The process repeats until the agent encounters the *terminal state* at which point the *episode* terminates. Markov Decision Process (MDP) can be used for mathematically modeling RL problems. According to the Markov property, it is considered the next state of the environment and the reward received depends solely on the current state and the agent's action in the current state. The cumulative discounted rewards, $G_t$ at any given timestep, $t$ is expressed as:

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{6.9}$$

where $\gamma$ is a discount factor and $\gamma \in (0, 1)$. The RL agent operates with the goal of maximizing the expected return, $E[G_t]$ from each state, $s_t$. A *policy*, $\pi(a_t|s_t)$ is a map-

ping from the current observation of the environment to a probability distribution of the actions that can be taken from the current state. During the training process, a traditional RL agent is required to visit all the states of the problem and store experiences in space-consuming tabular formats. This is a limitation that makes it infeasible to apply the traditional RL paradigm to problems with high dimensional states and action spaces. The integration of Deep Learning with the RL paradigm gave rise to an efficient means of overcoming the aforementioned limitation through the use of neural networks as function approximators for enabling the agent to estimate the value of a state or an action when it encounters a similar circumstance. In the resulting Deep Reinforcement Learning (DRL) paradigm, the policy, $\pi(a_t|s_t)$ is modeled as a parameterized function, $\pi_\theta(a_t|s_t)$ where $\theta$ is an adjustable parameter derived with an RL algorithm.

In *value based* RL methods, the RL agent attempts to learn a state-value function, $v_{\pi_\theta}(s)$, or a state-action value function, $Q_{\pi_\theta}(s,a)$. As the name implies, the state-value function estimates the value of a state, and it can be expressed in terms of expected return when following a policy $\pi_\theta$ starting from the state, $s$ as shown in Equation 6.10. Equation 6.11 indicates the state-action value function which is the expected return when action, $a_t$ is taken at state, $s_t$, and policy, $\pi_\theta$ is followed afterward.

$$v_{\pi_\theta}(s) = E_{\pi_\theta}[G_t|s_t = s] \tag{6.10}$$

$$Q_{\pi_\theta}(s,a) = E_{\pi_\theta}[G_t|s_t = s, a_t = a] \tag{6.11}$$

In *policy gradient* RL methods, the agent directly learns the policy, $\pi_\theta(a_t|s_t)$. Typically gradient-based techniques on the expectation of returns are used for learning the policy. Equation 6.12 indicates the form of the most commonly used gradient estimator.

$$\hat{g} = \hat{E}_t[\nabla_\theta \ln \pi_\theta(a_t|s_t)\hat{A}_t] \tag{6.12}$$

where, $\hat{A}_t$ is an estimator of the advantage function at timestep, $t$ and $\pi_\theta$ is a stochastic policy. In an RL algorithm that alternately performs sampling and optimization, the expectation $\hat{E}_t[..]$ indicates the empirical average computed over a batch of samples. For evaluating the performance of the policy, a performance objective the gradient of which

is the policy gradient estimator, $\hat{g}$ is defined. Accordingly, $\hat{g}$ is obtained by differentiating the objective:

$$L^{PG}(\theta) = \hat{E}_t[\ln \pi_\theta(a_t|s_t)\hat{A}_t] \tag{6.13}$$

Although multiple rounds of optimizations can be performed on the loss, $L^{PG}(\theta)$ defined in Equation 6.13 using a single trajectory of experience samples, it is not desirable since that could lead to adverse consequences such as policy updates that are destructively large. In order to overcome the aforementioned issue, in Proximal Policy Optimization [58] method, a clipped surrogate objective is used. More specifically, the degree to which new policy, $\pi_\theta(a_t|s_t)$ is allowed to change from old policy, $\pi_{\theta_{old}}(a_t|s_t)$ is restricted by the use of a clip function as indicated in Equation 6.14. The clip function, $\text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t$ removes the desirability of large policy updates that changes the $r_t(\theta)$ ratio beyond the interval $[1-\epsilon, 1+\epsilon]$.

$$\begin{aligned} &L^{CLIP}(\theta) = \hat{E}_t[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t] \\ &\text{where } r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \end{aligned} \tag{6.14}$$

Actor-critic is a branch of RL algorithms that combines the advantages of value-based methods and policy gradient RL methods. The actor is the policy that outputs a probability distribution over the actions that can be taken in the current state, and the critic is the value function approximator that evaluates the actions taken by the actor as per the policy.

### 6.3.4 Proposed RL Framework

As previously discussed, the default kube-scheduler takes multiple factors into account in formulating scheduling decisions including resource requirements and constraints, specifications of affinity and anti-affinity, deadlines, and interference caused by co-located workloads. These policies need to be pre-defined and may suffer from the general limitations of heuristic scheduling techniques. In this work, we override the default behavior and incorporate intelligence into the scheduler by training a DRL agent to select

**Algorithm 8** Actor-Critic based Scheduling Framework with PPO

Initialize actor networks and critic network with random weights
Initialize the training parameters: $\alpha, \beta, \gamma$
**for** episode = 1 to $N$ **do**
    Reset the environment
    **for** step = 1 to $T$ **do**
        Input the state of the environment to actor networks
        Select action $a_1$ from $\pi_\theta$
        Select action $a_2$ from $\pi_{\omega_i}$
        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** p = 1 to $S$ **do**
            Update critic network:
            $\sigma \leftarrow \sigma + \beta\delta_t \nabla v_\pi(s_t|\sigma)$
            Update first actor network:
            $\theta \leftarrow \theta + \alpha \hat{A}_p \nabla \ln \pi(a_1|s, \theta)$
            Update second actor network:
            $\omega \leftarrow \omega + \gamma \hat{A}_p \nabla \ln \pi(a_2|s, \omega)$
    Clear memory $D$
**return**

appropriate scheduling decisions with the objective of achieving a desired goal.

#### Agent Environment

The problem of scheduling workflows in a cloud cluster can be simplified by formulating it as a dependent task-scheduling problem. In the Argo workflow engine, pods corresponding to independent tasks are scheduled directly in the cluster for execution, while the tasks with dependencies are not scheduled until the parent tasks have completed execution. Whenever the workflow scheduler (RL agent), discovers a pod that is not assigned to a node, it takes the current state of the environment as input and outputs the most desirable node for task execution based on the trained policy. The environment then transitions to the next state. Accordingly, the timesteps of the proposed RL model are discrete and event-driven. The state, action, and reward of the RL model

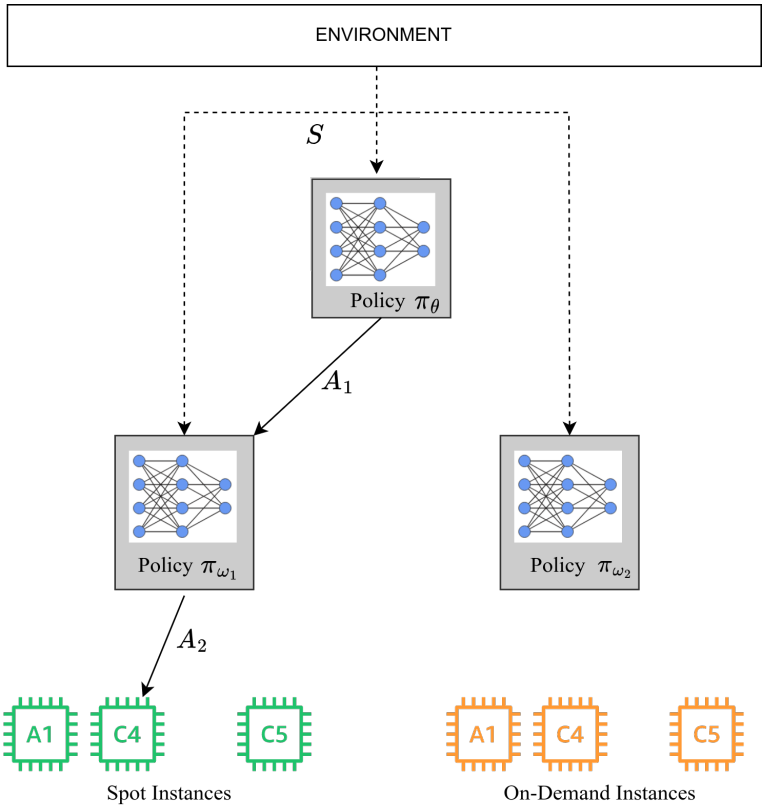



**Figure 6.2:** Proposed hierarchical action space and multi-actor DRL model

are designed as follows:

*State Space:* State of the environment comprises of total CPU and Memory requirements of the task, and nodes together with the estimated waiting time at each node based on the number of pods executing in each node.

*Action Space:* Compared to the problem of scheduling tasks in a cluster comprising nodes from the same cloud data center, scheduling tasks in a multi-cloud cluster is more challenging since resource capacities and cost are not the only factors that differentiate nodes. In such scenarios, the intercloud communication delay is an important factor that needs to be factored into the formulation of scheduling decisions. This requirement is further heightened in workflow scheduling due to the presence of data dependencies among tasks that may result in costly data transfers if communication costs among nodes from different clouds are ignored.

In the most straightforward design of the action space, the action of selecting any

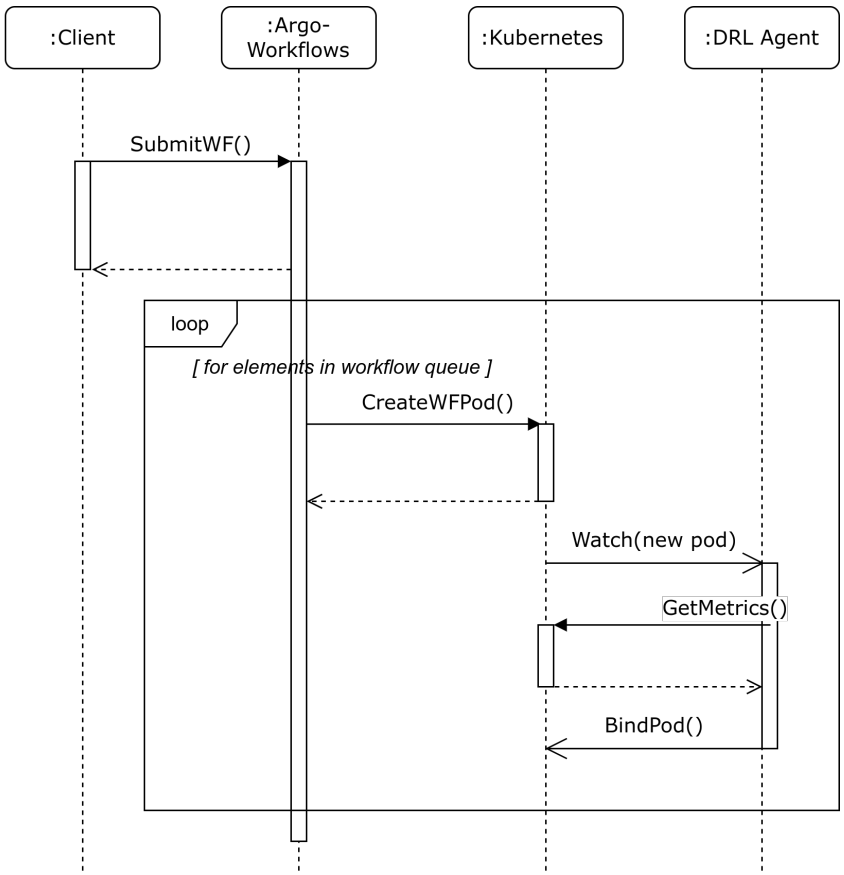



**Figure 6.3:** Sequence diagram of DRL based scheduling framework

one of the nodes in the cluster can be represented together in a flat action space. In this approach, the burden of distinguishing nodes from different pricing categories lies with the DRL agent. Although the agent may eventually manage to learn the presence of nodes from different categories based on rewards and thereby develop an internal representation of the composition of the cluster, it will inevitably reduce the training efficiency of the agent. Furthermore, as the size of the cluster grows, flat action spaces are more prone to the problem of the 'curse of dimensionality'.

In order to efficiently overcome the aforementioned challenges, we have designed the action space considering a logical organization of cluster. In the logical organization, nodes from different pricing categories are grouped together as shown in Figure 6.2. Accordingly, we define a hierarchical action space for the problem as follows:

$$A = \{(a_1, a_2) | a_1 \in \{\pi_{\omega_1}, \pi_{\omega_2}\} \ \ \& \ \ a_2 \in \{1, 2, ..., N_{a_1}\}\} \tag{6.15}$$

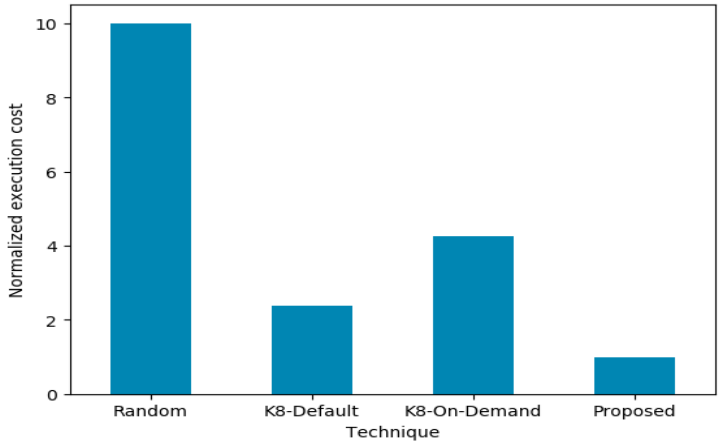
where $N_{a_1}$ is the total number of nodes in the cluster that belong to the pricing category given by action $a_1$. The action, $a_1$ corresponds to the selection of a pricing category, and the action, $a_2$ corresponds to the selection of a node from the selected category. An action at each timestep then corresponds to the joint action $(a_1, a_2)$.

*Reward:* Reward is the estimated cost of execution at the allocated node computed with Equation 6.6.

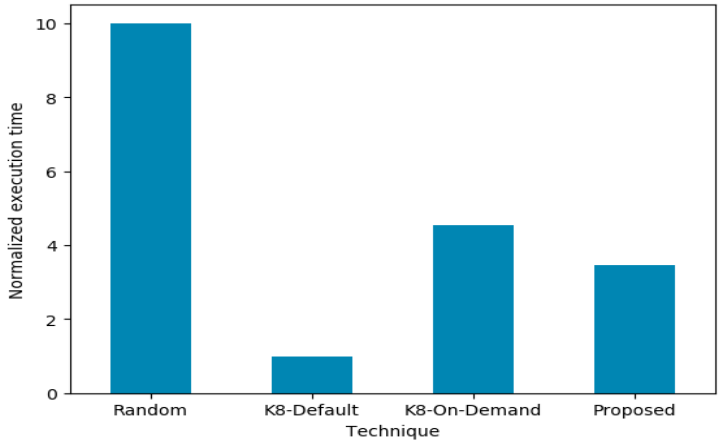
**Multi-Actor RL Algorithm**

The hierarchical action space described above can be represented as the tree structure in Figure 6.2. Each level of the tree corresponds to an action selection sub-problem. The first level of the tree represents the sub-problem of selecting a node group and the second level represents the sub-problem of selecting a node. We then adopt the hybrid actor-critic technique presented in [129] for selecting joint actions from the hierarchical action space. Different from a traditional actor-critic algorithm which contains a single actor-network and a single critic network, in the proposed architecture multiple parallel actor networks are guided by a common critic network.

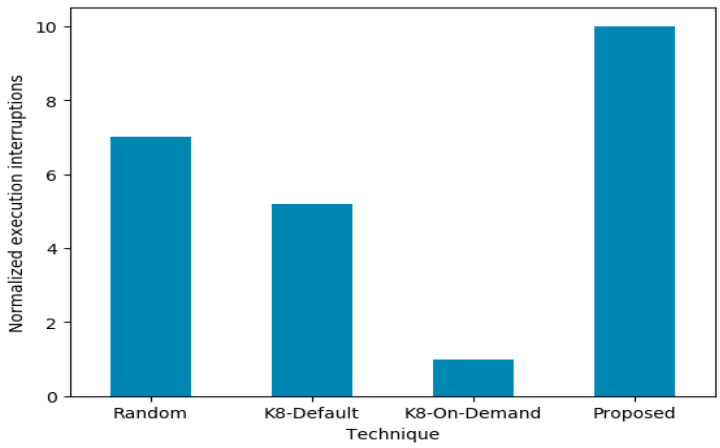
As shown in Figure 6.2 each action-selection sub-problem is handled by a separate actor network. Accordingly, one actor network learns a stochastic policy for selecting a node group. For each of the node groups, a separate actor network learns a stochastic policy for selecting a node from the respective node group. The critic network estimates the state value function, $V(s)$. The advantage function provided by the critic network is used for updating the stochastic policies. Actor networks are separately updated at each timestep by their respective update rules. We used the Proximal Policy Optimization method for updating the networks. Algorithm 8 summarizes the steps included in the training process of the DRL agent.




**(a)** Execution cost




**(b)** Execution time




**(c)** Execution interruptions

**Figure 6.4:** Comparison of performance of scheduling algorithms on an experimental dataset

| Instance Type | CPU Cores | Memory(GB) | Quantity | | Price | |
|---|---|---|---|---|---|---|
| | | | Spot | On-demand | Spot | On-demand |
| t4g.large | 2 | 8 | 2 | 2 | $0.033/h | $0.0672/h |
| t4g.xlarge | 4 | 16 | 3 | 2 | $0.0857/h | $0.1344/h |
| t4g.2xlarge | 8 | 32 | 1 | 1 | $0.1589/h | $0.2688/h |

**Table 6.1:** Resource configurations of Kubernetes cluster

## 6.4 Performance Evaluation

In this section we present the details of the experimental testbed used for evaluating the proposed DRL framework along with the results of the performance evaluation.

### 6.4.1 Experimental testbed

The experimental testbed was set up in Nectar Cloud. The resource configurations and composition of the Kubernetes cluster are shown in Table 6.1. Argo workflow engine is installed in the cluster in a separate namespace. A Python client that communicates with Argo api server was developed for submitting workflows and querying about the execution statistics of workflows.

### 6.4.2 Experimental dataset

The experimental dataset comprises of a set of Map-Reduce workflows. Each map task performs a CPU intensive parallelizable computation that involves finding the sum of the square-roots of numbers in a given input range. Workflow arrival rates for the experiment were drawn from a uniform distribution. The experiment was repeated multiple times to calculate results at a confidence interval of 95%.

### 6.4.3 DRL Scheduler Implementation

The Argo workflow engine uses the default Kubernetes scheduler for allocating tasks (i.e. pods) to nodes. We have overridden it with a DRL agent trained according to the proposed DRL framework. The configurations of all test workflows were updated such

that they are scheduled with the custom DRL scheduler instead of the default scheduler. Keras library [12] was used for developing the DRL framework.

Kubernetes metrics server collects resource metrics of the underlying nodes from Kubelets and shares it with the Kubernetes api server via the Metrics api. Therefore, by querying the Kubernetes api server we were able to retrieve near real-time CPU and Memory usages of the nodes, which were required to formulate the state space composition that needs to be provided as the state of the environment to the agent at each timestep of the episode. At the end of each episode, the python client queries the Argo api server for retrieving the execution statistics of workflows including resource times, start and end times of workflows and success rates which are then used for computing the resource usages and associated costs. The client also queries Kubernetes api server for retrieving node metrics that is required for computing the up times of nodes.

### 6.4.4 Comparison Algorithms

The performance of the proposed DRL algorithm was compared against three scheduling policies. **Random** policy allocates tasks to nodes in a Random manner, and is completely agnostic to pricing as well as other resource utilization levels of the cluster. **K8-Default** refers to the default scheduling policy of Kubernetes cluster. **On-Demand** is a policy that uses Kubernetes default scheduler but the selection is limited to the on-demand instances.

### 6.4.5 Experimental Results

Figure 6.4a shows the performance of the algorithms on the experimental dataset with respect to monetary cost of workflow executions. Random algorithm has incurred the highest cost owing to the fact that it distributes tasks across multiple instances without trying to optimize resource utilization or cost. In comparison the Kubernetes scheduler exhibits much better cost savings. By default, it is designed to select the most appropriate node through a node filtering and scoring process. In the filtering phase, nodes that are feasible for executing the pod are selected, and then they are ranked according to a scoring process. Based on the outcome of the filtering and scoring process the most ap-

propriate node for pod execution is selected. Clearly, this process has resulted in much better resource efficiency and thereby cost savings in comparison to random allocation. As expected, K8-On-Demand method has incurred a higher cost than the default policy since it is only allowed to make a selection from amongst the on-demand instances which have a higher unit cost. The proposed method has resulted in the highest cost savings. The significant reduction is cost is due to the intelligent cost aware allocation of pods among the instances in the cluster.

Figure 6.4b shows a comparison of the execution times of workflows scheduled with different algorithms. Again, the highest amount of time is taken by Random algorithm. K8-On-Demand has resulted in higher execution times compared to k8-Default due to the limited selection of instances available for scheduling. K8-Default has resulted in the least execution time since it distributes pods amongst multiple high scoring nodes, without considering the respective unit cost differences. Proposed algorithm has incurred a higher time in comparison to default since it's favoring nodes that are of low cost which leads to more pods being assigned to the same nodes, hence resulting in increased execution times. This is expected since instances with more vCPUS are more expensive, which results in a trade-off between execution time and cost. Furthermore, the re-execution of workflows that got interrupted due to spot instance interruptions contributes to increasing the total time.

Figure 6.4c shows the no of execution interruptions. Execution interruptions in the experimental context are solely due to the interruption of spot instanced which leads to workflows timing out and thereby failing to complete. As expected the proposed algorithm results is the highest interruptions since it is favoring spot instances for task executions, and the spot instances are subjected to interruptions. This is a known trade-off associated with the use of spot instances, therefore it is important to restrict the use of spot instances for failure tolerant workflows.

## 6.5 Summary

In this work, we designed a DRL technique for cost-optimized workflow scheduling in Cloud environments by the intelligent use of spot and on-demand instances. We

then designed and implemented an end-to-end system for integrating and training the DRL agent in the container-native Argo workflow engine that runs atop Kubernetes. As evidenced by the results of the experiments, higher cost savings can be achieved by overriding the default schedulers with intelligent cost-optimized scheduling policies.

# Chapter 7
# Conclusions and Future Directions

*This chapter concludes the thesis and provides a summary of works and key contributions. Next, it identifies and discusses several future research directions for further improving the scheduling of workflows across distributed cloud computing environments with the use of DRL techniques*

## 7.1 Summary of Contributions

Traditionally, centralized cloud data centers were used for the execution of workflow applications regardless of the diverse requirements of different types of applications. With the emergence of multi-clouds and edge computing, a hybrid combination of these computing infrastructures can be used for the execution of workflows, depending on specific needs of the applications [5]. For instance, a vast majority of IoT workflows have stringent real-time processing requirements which makes edge-cloud environments ideal for their execution, but certain other workflows such as scientific workflows are typically not delay-sensitive to the same extent and they could be executed in cloud and multi-cloud environments where there are less resource restrictions. Along with the added benefits offered by these novel computing paradigms, a set of new challenges are also introduced to the general problem of workflow scheduling in centralized cloud computing environments. For example, as opposed to cloud computing environments, edge nodes are resource constrained and maybe powered through batteries with limited capacities. Multi-cloud environments offered through geo-distributed datacenters may be powered with regional renewable energy sources which are inherently intermittent in nature. In order to leverage the true potential of these computing infrastructures, it is necessary to design workflow scheduling strategies that are capable of self-adapting to

the dynamic conditions and uncertainties associated with these environments. As evidenced by the success of Deep Reinforcement Learning in complex decision-making problems in multiple domains including robotics, gaming and healthcare, it is a viable candidate for efficiently handling the aforementioned challenges. However, the application of DRL techniques to complex cloud and edge computing environments is not a trivial task, and multiple challenges including curse of dimensionality, multi-objectivity, multi-agent coordination and decentralized execution need to be addressed for designing efficient DRL algorithms and architectures. In this thesis, we focused on devising novel DRL algorithms and architectures for efficiently solving these problems.

Chapter 1 presented the basic concepts of workflow scheduling in distributed cloud computing environments along with an introduction of RL basics. Next, the important challenges of the problem that motivates the work undertaken in this thesis are discussed. Finally, the specific research questions addressed are presented.

Chapter 2 analyzed existing RL based workflow scheduling techniques in cloud computing environments from a number of different perspectives and proposed a taxonomy based on these. The perspectives used for reviewing the techniques include the agent action for which RL is used, RL algorithm, number of objectives, agent architecture, training and execution architecture of the RL method and specific optimization objective of the problem.

Chapter 3 presents a baseline study performed for optimizing the energy-efficiency of workflow executions in a centralized cloud environment. In this work, a heuristic technique was proposed for jointly optimizing host and network energy consumption. The proposed heuristic considers workflow characteristics including precedence constraints and communication requirements among task instances together with the current operating status of execution nodes in the formulation of topology-aware scheduling decisions.

Chapter 4 In this chapter we studied the problem of workflow scheduling across edge-cloud environments with the objectives of optimizing energy-consumption and execution-time. The two objectives are inherently contradictory in nature as reduction in energy consumption is typically achieved at the expense of increased execution times. We utilized workflow deadlines to impose a soft upper bound on the degree to which

execution time is allowed to increase in exchange for higher energy savings. In order to incorporate multiple objectives to the DRL model, we formulated a single scalar additive reward function. One component of the scalar reward function reflects the desirability of agents' action with respect to the goal of energy minimization and the other component either rewards or penalizes the agent for meeting or exceeding deadlines respectively. In order to promote a clear distinction between edge and cloud nodes, we have proposed a novel hierarchical action space. A DRL algorithm in which multiple actor networks are guided by a single critic network is then proposed for efficiently scheduling workflows across the highly dynamic edge-cloud environments.

Chapter 5 presents a novel multi-agent DRL architecture for handling the problem of scheduling workflows across geo-distributed cloud datacenters powered through a combination of green and brown energy sources. The proposed architecture comprises a hierarchical design in which a global DRL agent allocates tasks to local DRL agents each of which are responsible for selecting execution nodes from a local datacenter. For incorporating the realistic problem of partial observability that is inherent in multi-cloud environments to the agent environment, a novel formulation of the problem as a Partially Observable Markov Decision Process (POMDP) is presented. An efficient centralized training and decentralized execution technique is utilized for enabling the agents to share extra information during training and then only use locally available information during execution. Along with this a novel shared reward model which unilaterally motivates agents to operate towards a common goal is proposed. In order to tackle the problem of curse of dimensionality which is particularly prominent in multi-agent environments, a novel approach to limiting the observations shared by agents to a local neighborhood is proposed.

Chapter 6 presents a cost optimized workflow scheduling technique in cloud computing environments with the use of DRL. Proposed approach uses a combination of on-demand and spot instances for reducing the execution cost of workflows. The DRL agent is then integrated to the open source Argo workflow engine.

The chapters described above presented multiple algorithms and architectures for optimizing workflow executions in distributed cloud computing environments. The proposed DRL algorithms are indeed a timely contribution with a great potential for

advancing the state-of-the-art.

## 7.2 Future Research Directions

### 7.2.1 Supporting multiple objectives with multi-policy RL algorithms

Due to the diverse requirements of stakeholders real world scenarios typically require the workflow schedulers to optimize more than one objective. The existing RL based scheduling algorithms convert multiple objectives into a single objective with a single scalar reward function. The function may include multiple weighted components each relating to a particular objective. There are multiple problems with this approach [61]. Firstly, the decision of which weights are to be assigned for which objectives is a manual process that requires significant domain expertise. Even an educated guess made by a domain expert could deviate from the optimal solution that could've been achieved if all potentially optimal policies were evaluated to find the policy that results in the best trade-off between multiple objectives. Furthermore, a change in objective preferences of the underlying system would make the current policy obsolete hence requiring the single objective agent to be retrained. Finally, evaluating multiple reward functions to identify the right match tends to be a costly process in terms of computation time and sample complexity. Therefore, it would be beneficial to explore how multiple objectives can be explicitly incorporated into RL frameworks designed for scheduling workflows across distributed cloud computing environments. Multi-agent architectures in which each agent is solely responsible for optimizing a particular agent is one potential means of achieving this [59].

### 7.2.2 Designing Multi-agent RL solutions for complex scheduling problems

A majority of existing workflow scheduling methods have proposed single-agent RL algorithms regardless of the dynamicity and decentralized nature of underlying infrastructures. Although single agent systems are less complex to design, they may not be efficient to capture the dynamics of more complex real world scheduling scenarios such

as when workloads are to be scheduled across federated clouds and emerging fog computing environments. Accordingly, Multi-Agent Systems (MAS) in which interactions between multiple agents are leveraged, is proven to be a better fit for problem solving in highly distributed and stochastic environments compared to its single agent counterpart [9]. Therefore, it is worthwhile to investigate how multiple agents can be leveraged for designing more efficient RL solutions complex workflow execution environments. For instance, competitive multiple agents settings can be designed so multiple agents work concurrently on sub-problems of a larger problem which in turn enhances the scalability of the system as well as fault tolerance.

Furthermore, as opposed to centralized cloud environments, in highly distributed multi-cloud and edge computing scenarios, it is likely that information about the status of all nodes are not centrally available in real time. It will be beneficial to use the centralized training and distributed execution paradigm in such scenarios so that the agents can be trained to learn decentralized policies with additional information that is only available at training time. During execution the agents can operate solely based on local observations and partial information about the intentions of other agents. This enables more realistic modeling of the actual environment and also helps simplify communications between agents [67].

### 7.2.3 Estimating task execution times accurately

Accurate estimations of task runtimes are important since some scheduling algorithms rely on such estimates for formulating scheduling plans. For RL based algorithms, such estimates are particularly useful since they can be used during the training process of agents to reward efficient allocations. The more accurate the estimations are, the better the learning of the agent will be. However, accurately estimating task runtimes is a non-trivial task particularly due to the performance variability that is inherent in cloud and edge computing environments. Factors such as interference due to co-located workloads from multiple tenants, geographical distribution of resources makes it difficult to produce accurate estimations. Few studies have used machine learning techniques for estimating run times [148]. Techniques such as online incremental machine learning

could be explored for achieving accurate task estimates.

### 7.2.4 Using asynchronous RL methods for improving training efficiency

The existing RL based workflow scheduling algorithms are mainly based on synchronous learning/training. However, in highly dynamic cloud environments, it is likely that the changing environments as well as stakeholder preferences give rise to the need for training and re-training RL models frequently. Therefore, it is beneficial to use asynchronous reinforcement learning techniques [70] for speeding up the training process without requiring the use of specialized hardware such as GPUs. In particular, the asynchronous advantage actor critic (A3C) in which multiple actor critic agents are asynchronously executed in parallel on separate instances of the environment, is highly effective in reducing training time. Importance Weighted Actor-Learner Architecture (IMPALA ) is a distributed agent architecture that is proven to achieve even better performance than A3C methods [71]. These methods can be leveraged by researchers for designing scheduling agents that are more data efficient and stable.

### 7.2.5 Handling large action spaces more efficiently

Existing works have modeled the action spaces of workflow scheduling problems with simple flat designs. This in turn results in large discrete action spaces, particularly if workflows are to be scheduled across large infrastructures such as multiple cloud datacenters. Such designs are more prone to Curse of dimensionality problem that inhibits the learning progress of RL agents. This issue is exacerbated in multi-agent settings due to exponential growth in joint state and action spaces with the number of agents [7]. Therefore it is worthwhile to explore techniques that result in more efficient action space designs. For example, in hierarchical reinforcement learning methods, the problem is sub-divided into a number of sub-problems [57], and these methods can be used for designing action hierarchies. For instance, in multi cloud scenarios, action spaces could be subdivided at different levels including availability zone, subnet, rack and node levels. Action branching architecture is another potential direction that could be investigated in this regard [56]. Some studies have also used clustering techniques such as nearest

neighbors [149], [47], and it is worth exploring how such approaches can be incorporated in workflow schedulers designed to operate across large-scale infrastructures.

## 7.3 Final Remarks

The problem of scheduling workflows on cloud and edge computing environments includes multiple levels of complexities. Firstly, the general case of workflow scheduling in distributed systems in NP-hard. Scheduling workflows across highly dynamic cloud and edge computing environments is even more complex due to inherent challenges associated with these environments including the need to satisfy diverse contradictory objectives, coordinating executions across highly distributed infrastructures and dynamicity of the operating conditions. Deep Reinforcement Learning (DRL) has emerged as a promising paradigm for dealing with highly dynamic and complex problems due to the ability of DRL agents to learn to operate in stochastic environments. Despite the benefits of DRL, there are multiple challenges associated with the application of DRL techniques in distributed computing environments including partial observability, multi-agent coordination, decentralized execution, multi-objectivity and curse of dimensionality. In this thesis, we investigated novel DRL algorithms and architectures to overcome the aforementioned challenges. We then incoporated these DRL techniques in the formulation of proposed workflow scheduling algorithms. As evidenced by the outcomes of this research, DRL based solutions have the potential to efficiently solve complex challenges associated with the problem of workflow scheduling on cloud and edge computing environments.

# Bibliography

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

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

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

[4] R. S. Sutton and A. G. Barto, Reinforcement learning: An introduction. MIT press, 2018.

[5] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," Machine learning, vol. 8, no. 3-4, pp. 229–256, 1992.

[6] J. D. Ullman, "Np-complete scheduling problems," Journal of Computer and System sciences, vol. 10, no. 3, pp. 384–393, 1975.

[7] L. Canese, G. C. Cardarilli, L. Di Nunzio, R. Fazzolari, D. Giardino, M. Re, and S. Spanò, "Multi-agent reinforcement learning: A review of challenges and applications," Applied Sciences, vol. 11, no. 11, p. 4948, 2021.

[8] M. Goudarzi, M. S. Palaniswami, and R. Buyya, "A distributed deep reinforcement learning technique for application placement in edge and fog computing environments," IEEE Transactions on Mobile Computing, 2021.

[9] L. Buşoniu, R. Babuška, and B. De Schutter, "Multi-agent reinforcement learning: An overview," Innovations in multi-agent systems and applications-1, pp. 183–221, 2010.

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

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

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

[13] D. Abts, M. R. Marty, P. M. Wells, P. Klausler, and H. Liu, "Energy proportional datacenter networks," in ACM SIGARCH Computer Architecture News, vol. 38, no. 3. ACM, 2010, pp. 338–347.

[14] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in Proceedings of the 15th ACM workshop on hot topics in networks, 2016, pp. 50–56.

[15] W. Deng, F. Liu, H. Jin, B. Li, and D. Li, "Harnessing renewable energy in cloud datacenters: opportunities and challenges," iEEE Network, vol. 28, no. 1, pp. 48–55, 2014.

[16] H. Topcuoglu, S. Hariri, and M.-y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," IEEE Transactions on Parallel and Distributed Systems, vol. 13, no. 3, pp. 260–274, 2002.

[17] W. Zheng and S. Huang, "An adaptive deadline constrained energy-efficient scheduling heuristic for workflows in clouds," Concurrency and Computation: Practice and Experience, vol. 27, no. 18, pp. 5590–5605, 2015.

[18] M. Dayarathna, Y. Wen, and R. Fan, "Data center energy consumption modeling: A survey," IEEE Communications surveys & tutorials, vol. 18, no. 1, pp. 732–794, 2015.

[19] G. L. Stavrinides and H. D. Karatza, "An energy-efficient, qos-aware and cost-effective scheduling approach for real-time workflow applications in cloud computing systems utilizing dvfs and approximate computations," Future Generation Computer Systems, vol. 96, pp. 216–226, 2019.

[20] A. Taghinezhad-Niar, S. Pashazadeh, and J. Taheri, "Energy-efficient workflow scheduling with budget-deadline constraints for cloud," Computing, vol. 104, no. 3, pp. 601–625, 2022.

[21] X. Xu, W. Dou, X. Zhang, and J. Chen, "Enreal: An energy-aware resource allocation method for scientific workflow executions in cloud environment," IEEE transactions on cloud computing, vol. 4, no. 2, pp. 166–179, 2015.

[22] Y. Wen, Z. Wang, Y. Zhang, J. Liu, B. Cao, and J. Chen, "Energy and cost aware scheduling with batch processing for instance-intensive iot workflows in clouds," Future Generation Computer Systems, vol. 101, pp. 39–50, 2019.

[23] Z. Li, J. Ge, H. Hu, W. Song, H. Hu, and B. Luo, "Cost and energy aware scheduling algorithm for scientific workflows with deadline constraint in clouds," IEEE Transactions on Services Computing, vol. 11, no. 4, pp. 713–726, 2015.

[24] S. Ijaz, E. U. Munir, S. G. Ahmad, M. M. Rafique, and O. F. Rana, "Energy-makespan optimization of workflow scheduling in fog–cloud computing," Computing, vol. 103, no. 9, pp. 2033–2059, 2021.

[25] M. Kaur, S. Kadam, and N. Hannoon, "Multi-level parallel scheduling of dependent-tasks using graph-partitioning and hybrid approaches over edge-cloud," Soft Computing, vol. 26, no. 11, pp. 5347–5362, 2022.

[26] M. A. Rodriguez and R. Buyya, "Deadline based resource provisioningand scheduling algorithm for scientific workflows on clouds," IEEE transactions on cloud computing, vol. 2, no. 2, pp. 222–235, 2014.

[27] Z. Li, J. Ge, H. Yang, L. Huang, H. Hu, H. Hu, and B. Luo, "A security and cost aware scheduling algorithm for heterogeneous tasks of scientific workflow in clouds," Future Generation Computer Systems, vol. 65, pp. 140–152, 2016.

[28] G. Ismayilov and H. R. Topcuoglu, "Neural network based multi-objective evolutionary algorithm for dynamic workflow scheduling in cloud computing," Future Generation computer systems, vol. 102, pp. 307–322, 2020.

[29] P. Hosseinioun, M. Kheirabadi, S. R. K. Tabbakh, and R. Ghaemi, "A new energy-aware tasks scheduling approach in fog computing using hybrid meta-heuristic algorithm," Journal of Parallel and Distributed Computing, vol. 143, pp. 88–96, 2020.

[30] M. Mokni, S. Yassa, J. E. Hajlaoui, M. N. Omri, and R. Chelouah, "Multi-objective fuzzy approach to scheduling and offloading workflow tasks in fog–cloud computing," Simulation Modelling Practice and Theory, vol. 123, p. 102687, 2023.

[31] D. Ajwani, A. Cosgaya-Lozano, and N. Zeh, "A topological sorting algorithm for large graphs," Journal of Experimental Algorithmics (JEA), vol. 17, pp. 3–1, 2012.

[32] C.-g. Wu, W. Li, L. Wang, and A. Y. Zomaya, "Hybrid evolutionary scheduling for energy-efficient fog-enhanced internet of things," IEEE Transactions on Cloud Computing, vol. 9, no. 2, pp. 641–653, 2018.

[33] Y. Xie, Y. Zhu, Y. Wang, Y. Cheng, R. Xu, A. S. Sani, D. Yuan, and Y. Yang, "A novel directional and non-local-convergent particle swarm optimization based workflow scheduling in cloud–edge environment," Future Generation Computer Systems, vol. 97, pp. 361–378, 2019.

[34] N. Bacanin, M. Zivkovic, T. Bezdan, K. Venkatachalam, and M. Abouhawwash, "Modified firefly algorithm for workflow scheduling in cloud-edge environment," Neural Computing and Applications, vol. 34, no. 11, pp. 9043–9068, 2022.

[35] A. Choudhary, I. Gupta, V. Singh, and P. K. Jana, "A gsa based hybrid algorithm for bi-objective workflow scheduling in cloud computing," Future Generation Computer Systems, vol. 83, pp. 14–26, 2018.

[36] O. H. Ahmed, J. Lu, Q. Xu, A. M. Ahmed, A. M. Rahmani, and M. Hosseinzadeh, "Using differential evolution and moth–flame optimization for scientific workflow scheduling in fog computing," Applied Soft Computing, vol. 112, p. 107744, 2021.

[37] E. Barrett, E. Howley, and J. Duggan, "Applying reinforcement learning towards automating resource allocation and application scalability in the cloud,"

Concurrency and computation: practice and experience, vol. 25, no. 12, pp. 1656–1674, 2013.

[38] L. Lin, L. Pan, and S. Liu, "Learning to make auto-scaling decisions with heterogeneous spot and on-demand instances via reinforcement learning," Information Sciences, vol. 614, pp. 480–496, 2022.

[39] L. Schuler, S. Jamil, and N. Kühl, "Ai-based resource allocation: Reinforcement learning for adaptive auto-scaling in serverless environments," in 2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid). IEEE, 2021, pp. 804–811.

[40] N. Liu, Z. Li, J. Xu, Z. Xu, S. Lin, Q. Qiu, J. Tang, and Y. Wang, "A hierarchical framework of cloud resource allocation and power management using deep reinforcement learning," in 2017 IEEE 37th international conference on distributed computing systems (ICDCS). IEEE, 2017, pp. 372–382.

[41] L. Benini, A. Bogliolo, and G. De Micheli, "A survey of design techniques for system-level dynamic power management," IEEE transactions on very large scale integration (VLSI) systems, vol. 8, no. 3, pp. 299–316, 2000.

[42] B. Wang, F. Liu, and W. Lin, "Energy-efficient vm scheduling based on deep reinforcement learning," Future Generation Computer Systems, vol. 125, pp. 616–628, 2021.

[43] F. M. Talaat, M. S. Saraya, A. I. Saleh, H. A. Ali, and S. H. Ali, "A load balancing and optimization strategy (lbos) using reinforcement learning in fog computing environment," Journal of Ambient Intelligence and Humanized Computing, vol. 11, no. 11, pp. 4951–4966, 2020.

[44] J.-y. Baek, G. Kaddoum, S. Garg, K. Kaur, and V. Gravel, "Managing fog networks using reinforcement learning based load balancing algorithm," in 2019 IEEE Wireless Communications and Networking Conference (WCNC). IEEE, 2019, pp. 1–7.

[45] Z. Gao, Q. Jiao, K. Xiao, Q. Wang, Z. Mo, and Y. Yang, "Deep reinforcement learning based service migration strategy for edge computing," in 2019 IEEE international conference on service-oriented system engineering (SOSE). IEEE, 2019, pp. 116–1165.

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

[47] G. Dulac-Arnold, R. Evans, H. van Hasselt, P. Sunehag, T. Lillicrap, J. Hunt, T. Mann, T. Weber, T. Degris, and B. Coppin, "Deep reinforcement learning in large discrete action spaces," arXiv preprint arXiv:1512.07679, 2015.

[48] A. Jayanetti, S. Halgamuge, and R. Buyya, "Deep reinforcement learning for energy and time optimized scheduling of precedence-constrained tasks in edge–cloud computing environments," Future Generation Computer Systems, vol. 137, pp. 14–30, 2022.

[49] Y. Hu, C. de Laat, and Z. Zhao, "Learning workflow scheduling on multi-resource clusters," in 2019 IEEE International Conference on Networking, Architecture and Storage (NAS). IEEE, 2019, pp. 1–8.

[50] A. Asghari, M. K. Sohrabi, and F. Yaghmaee, "Task scheduling, resource provisioning, and load balancing on scientific workflows using parallel sarsa reinforcement learning agents and genetic algorithm," The Journal of Supercomputing, vol. 77, no. 3, pp. 2800–2828, 2021.

[51] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Task scheduling algorithms for heterogeneous processors," in Proceedings. Eighth Heterogeneous Computing Workshop (HCW'99). IEEE, 1999, pp. 3–14.

[52] Z. Tong, X. Deng, H. Chen, J. Mei, and H. Liu, "Ql-heft: a novel machine learning scheduling scheme base on cloud computing environment," Neural Computing and Applications, vol. 32, no. 10, pp. 5553–5570, 2020.

[53] A. Kaur, P. Singh, R. Singh Batth, and C. Peng Lim, "Deep-q learning-based heterogeneous earliest finish time scheduling algorithm for scientific workflows in cloud," Software: Practice and Experience, vol. 52, no. 3, pp. 689–709, 2022.

[54] A. Asghari, M. K. Sohrabi, and F. Yaghmaee, "Online scheduling of dependent tasks of cloud's workflows to enhance resource utilization and reduce the makespan using multiple reinforcement learning-based agents," Soft Computing, vol. 24, no. 21, pp. 16 177–16 199, 2020.

[55] ——, "A cloud resource management framework for multiple online scientific workflows using cooperative reinforcement learning agents," Computer Networks, vol. 179, p. 107340, 2020.

[56] A. Tavakoli, F. Pardo, and P. Kormushev, "Action branching architectures for deep reinforcement learning," in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 32, no. 1, 2018.

[57] A. G. Barto and S. Mahadevan, "Recent advances in hierarchical reinforcement learning," Discrete event dynamic systems, vol. 13, no. 1, pp. 41–77, 2003.

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

[59] R. Rădulescu, P. Mannion, D. M. Roijers, and A. Nowé, "Multi-objective multi-agent decision making: a utility-based analysis and survey," Autonomous Agents and Multi-Agent Systems, vol. 34, no. 1, p. 10, 2020.

[60] D. M. Roijers, P. Vamplew, S. Whiteson, and R. Dazeley, "A survey of multi-objective sequential decision-making," Journal of Artificial Intelligence Research, vol. 48, pp. 67–113, 2013.

[61] C. F. Hayes, R. Rădulescu, E. Bargiacchi, J. Källström, M. Macfarlane, M. Reymond, T. Verstraeten, L. M. Zintgraf, R. Dazeley, F. Heintz et al., "A practical guide to multi-objective reinforcement learning and planning," Autonomous Agents and Multi-Agent Systems, vol. 36, no. 1, p. 26, 2022.

[62] D. L. Mammen and V. R. Lesser, "Problem structure and subproblem sharing in multi-agent systems," in Proceedings International Conference on Multi Agent Systems (Cat. No. 98EX160). IEEE, 1998, pp. 174–181.

[63] K. Zhang, Z. Yang, and T. Başar, "Multi-agent reinforcement learning: A selective overview of theories and algorithms," Handbook of reinforcement learning and control, pp. 321–384, 2021.

[64] M. L. Littman, "Markov games as a framework for multi-agent reinforcement learning," in Machine learning proceedings 1994. Elsevier, 1994, pp. 157–163.

[65] P. P. Santos, D. S. Carvalho, M. Vasco, A. Sardinha, P. A. Santos, A. Paiva, and F. S. Melo, "Centralized training with hybrid execution in multi-agent reinforcement learning," arXiv preprint arXiv:2210.06274, 2022.

[66] J. K. Gupta, M. Egorov, and M. Kochenderfer, "Cooperative multi-agent control using deep reinforcement learning," in International conference on autonomous agents and multiagent systems. Springer, 2017, pp. 66–83.

[67] J. Foerster, I. A. Assael, N. De Freitas, and S. Whiteson, "Learning to communicate with deep multi-agent reinforcement learning," Advances in neural information processing systems, vol. 29, 2016.

[68] Y. Zhang, R. Li, Y. Zhao, R. Li, Y. Wang, and Z. Zhou, "Multi-agent deep reinforcement learning for online request scheduling in edge cooperation networks," Future Generation Computer Systems, vol. 141, pp. 258–268, 2023.

[69] P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls et al., "Value-decomposition networks for cooperative multi-agent learning," arXiv preprint arXiv:1706.05296, 2017.

[70] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in International conference on machine learning. PMLR, 2016, pp. 1928–1937.

[71] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning et al., "Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures," in International conference on machine learning. PMLR, 2018, pp. 1407–1416.

[72] A. Nair, P. Srinivasan, S. Blackwell, C. Alcicek, R. Fearon, A. De Maria, V. Panneershelvam, M. Suleyman, C. Beattie, S. Petersen et al., "Massively parallel methods for deep reinforcement learning," arXiv preprint arXiv:1507.04296, 2015.

[73] M. H. Hilman, M. A. Rodriguez, and R. Buyya, "Multiple workflows scheduling in multi-tenant distributed systems: A taxonomy and future directions," ACM Computing Surveys (CSUR), vol. 53, no. 1, pp. 1–39, 2020.

[74] M. Avgerinou, P. Bertoldi, and L. Castellazzi, "Trends in data centre energy consumption under the european code of conduct for data centre energy efficiency," Energies, vol. 10, no. 10, p. 1470, 2017.

[75] C. Jiang, T. Fan, H. Gao, W. Shi, L. Liu, C. Cérin, and J. Wan, "Energy aware edge computing: A survey," Computer Communications, vol. 151, pp. 556–580, 2020.

[76] S. Ilager, K. Ramamohanarao, and R. Buyya, "Thermal prediction for efficient energy management of clouds using machine learning," IEEE Transactions on Parallel and Distributed Systems, vol. 32, no. 5, pp. 1044–1056, 2020.

[77] R. Medara and R. S. Singh, "A review on energy-aware scheduling techniques for workflows in iaas clouds," Wireless Personal Communications, pp. 1–40, 2022.

[78] Y. Qin, H. Wang, S. Yi, X. Li, and L. Zhai, "An energy-aware scheduling algorithm for budget-constrained scientific workflows based on multi-objective reinforcement learning," The Journal of Supercomputing, vol. 76, no. 1, pp. 455–480, 2020.

[79] Y. Wang, H. Liu, W. Zheng, Y. Xia, Y. Li, P. Chen, K. Guo, and H. Xie, "Multi-objective workflow scheduling with deep-q-network-based multi-agent reinforcement learning," IEEE access, vol. 7, pp. 39 974–39 982, 2019.

[80] A. Nascimento, V. Olimpio, V. Silva, A. Paes, and D. de Oliveira, "A reinforcement learning scheduling strategy for parallel cloud-based workflows," in 2019

IEEE international parallel and distributed processing symposium workshops (IPDPSW). IEEE, 2019, pp. 817–824.

[81] A. M. Kintsakis, F. E. Psomopoulos, and P. A. Mitkas, "Reinforcement learning based scheduling in a workflow management system," Engineering Applications of Artificial Intelligence, vol. 81, pp. 94–106, 2019.

[82] T. Dong, F. Xue, C. Xiao, and J. Li, "Task scheduling based on deep reinforcement learning in a cloud manufacturing environment," Concurrency and Computation: Practice and Experience, vol. 32, no. 11, p. e5654, 2020.

[83] Z. Peng, J. Lin, D. Cui, Q. Li, and J. He, "A multi-objective trade-off framework for cloud resource scheduling based on the deep q-network algorithm," Cluster Computing, vol. 23, no. 4, pp. 2753–2767, 2020.

[84] A. I. Orhean, F. Pop, and I. Raicu, "New scheduling approach using reinforcement learning for heterogeneous distributed systems," Journal of Parallel and Distributed Computing, vol. 117, pp. 292–302, 2018.

[85] Q. Wu, Z. Wu, Y. Zhuang, and Y. Cheng, "Adaptive dag tasks scheduling with deep reinforcement learning," in International Conference on Algorithms and Architectures for Parallel Processing. Springer, 2018, pp. 477–490.

[86] H. Li, J. Huang, B. Wang, and Y. Fan, "Weighted double deep q-network based reinforcement learning for bi-objective multi-workflow scheduling in the cloud," Cluster Computing, vol. 25, no. 2, pp. 751–768, 2022.

[87] F. Xue, Q. Hai, T. Dong, Z. Cui, and Y. Gong, "A deep reinforcement learning based hybrid algorithm for efficient resource scheduling in edge computing environment," Information Sciences, vol. 608, pp. 362–374, 2022.

[88] Y. Zhang, Z. Zhou, Z. Shi, L. Meng, and Z. Zhang, "Online scheduling optimization for dag-based requests through reinforcement learning in collaboration edge networks," IEEE Access, vol. 8, pp. 72 985–72 996, 2020.

[89] Z. Tong, H. Chen, X. Deng, K. Li, and K. Li, "A scheduling scheme in the cloud computing environment using deep q-learning," Information Sciences, vol. 512, pp. 1170–1191, 2020.

[90] D. Kliazovich, J. E. Pecero, A. Tchernykh, P. Bouvry, S. U. Khan, and A. Y. Zomaya, "Ca-dag: Modeling communication-aware applications for scheduling in cloud computing," Journal of Grid Computing, vol. 14, no. 1, pp. 23–39, 2016.

[91] H. Chen, X. Zhu, D. Qiu, H. Guo, L. T. Yang, and P. Lu, "Eons: minimizing energy consumption for executing real-time workflows in virtualized cloud data centers," in Proceedings of the 45th International Conference on Parallel Processing Workshops (ICPPW). IEEE, 2016, pp. 385–392.

[92] M. A. Rodriguez and R. Buyya, "Scheduling dynamic workloads in multi-tenant scientific workflow as a service platforms," Future Generation Computer Systems, vol. 79, pp. 739–750, 2018.

[93] W. Gerlach, W. Tang, K. Keegan, T. Harrison, A. Wilke, J. Bischof, M. D'Souza, S. Devoid, D. Murphy-Olson, N. Desai et al., "Skyport: container-based execution environment management for multi-cloud scientific workflows," in Proceedings of the 5th International Workshop on Data-Intensive Computing in the Clouds. IEEE Press, 2014, pp. 25–32.

[94] "Firecracker," 2018. [Online]. Available: https://firecracker-microvm.github.io/

[95] M. A. Rodriguez and R. Buyya, "A taxonomy and survey on scheduling algorithms for scientific workflows in iaas cloud computing environments," Concurrency and Computation: Practice and Experience, vol. 29, no. 8, p. e4041, 2017.

[96] K. Bousselmi, Z. Brahmi, and M. M. Gammoudi, "Energy efficient partitioning and scheduling approach for scientific workflows in the cloud," in 2016 IEEE International Conference on Services Computing (SCC). IEEE, 2016, pp. 146–154.

[97] M. Mezmaz, N. Melab, Y. Kessaci, Y. C. Lee, E.-G. Talbi, A. Y. Zomaya, and D. Tuyttens, "A parallel bi-objective hybrid metaheuristic for energy-aware scheduling

for cloud computing systems," Journal of Parallel and Distributed Computing, vol. 71, no. 11, pp. 1497–1508, 2011.

[98] A. Verma and S. Kaushal, "A hybrid multi-objective particle swarm optimization for scientific workflow scheduling," Parallel Computing, vol. 62, pp. 1–19, 2017.

[99] G. Yao, Y. Ding, Y. Jin, and K. Hao, "Endocrine-based coevolutionary multi-swarm for multi-objective workflow scheduling in a cloud system," Soft Computing, vol. 21, no. 15, pp. 4309–4322, 2017.

[100] S. Yassa, R. Chelouah, H. Kadima, and B. Granado, "Multi-objective approach for energy-aware workflow scheduling in cloud computing environments," The Scientific World Journal, vol. 2013, 2013.

[101] M. Khaleel and M. M. Zhu, "Energy-efficient task scheduling and consolidation algorithm for workflow jobs in cloud," International Journal of Computational Science and Engineering, vol. 13, no. 3, pp. 268–284, 2016.

[102] X. Qu, P. Xiao, and L. Huang, "Improving the energy efficiency and performance of data-intensive workflows in virtualized clouds," The Journal of Supercomputing, vol. 74, no. 7, pp. 2935–2955, 2018.

[103] M. Sharifi, S. Shahrivari, and H. Salimi, "Pasta: a power-aware solution to scheduling of precedence-constrained tasks on heterogeneous computing resources," Computing, vol. 95, no. 1, pp. 67–88, 2013.

[104] Z. Tang, L. Qi, Z. Cheng, K. Li, S. U. Khan, and K. Li, "An energy-efficient task scheduling algorithm in dvfs-enabled cloud environment," Journal of Grid Computing, vol. 14, no. 1, pp. 55–74, 2016.

[105] M. Żotkiewicz, M. Guzek, D. Kliazovich, and P. Bouvry, "Minimum dependencies energy-efficient scheduling in data centers," IEEE Transactions on Parallel and Distributed Systems, vol. 27, no. 12, pp. 3561–3574, 2016.

[106] D. Kliazovich, P. Bouvry, and S. U. Khan, "Dens: data center energy-efficient network-aware scheduling," Cluster Computing, vol. 16, no. 1, pp. 65–75, 2013.

[107] S. Vakilinia, "Energy efficient temporal load aware resource allocation in cloud computing datacenters," Journal of Cloud Computing, vol. 7, no. 1, p. 2, 2018.

[108] B. Cao, X. Gao, G. Chen, and Y. Jin, "Nice: network-aware vm consolidation scheme for energy conservation in data centers," in Proceedings of the 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS). IEEE, 2014, pp. 166–173.

[109] W. Fang, X. Liang, S. Li, L. Chiaraviglio, and N. Xiong, "Vmplanner: Optimizing virtual machine placement and traffic flow routing to reduce network power costs in cloud data centers," Computer Networks, vol. 57, no. 1, pp. 179–196, 2013.

[110] H. Jin, T. Cheocherngngarn, D. Levy, A. Smith, D. Pan, J. Liu, and N. Pissinou, "Joint host-network optimization for energy-efficient data center networking," in Proceedings of the 27th IEEE International Symposium on Parallel and Distributed Processing. IEEE, 2013, pp. 623–634.

[111] S. Pelley, D. Meisner, T. F. Wenisch, and J. W. VanGilder, "Understanding and abstracting total data center power," in Workshop on Energy-Efficient Design, vol. 11, 2009.

[112] X. Wang, Y. Yao, X. Wang, K. Lu, and Q. Cao, "Carpo: Correlation-aware power optimization in data center networks," in Proceedings of IEEE INFOCOM. IEEE, 2012, pp. 1125–1133.

[113] O. Sinnen, Task scheduling for parallel systems. John Wiley & Sons, 2007, vol. 60.

[114] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in ACM SIGCOMM Computer Communication Review, vol. 38, no. 4. ACM, 2008, pp. 63–74.

[115] M. Guzek, D. Kliazovich, and P. Bouvry, "Heros: Energy-efficient load balancing for heterogeneous data centers," in Proceedings of the 8th IEEE International Conference on Cloud Computing. IEEE, 2015, pp. 742–749.

[116] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," Software: Practice and experience, vol. 41, no. 1, pp. 23–50, 2011.

[117] "Alibaba cluster data," 2018. [Online]. Available: https://github.com/alibaba/clusterdata

[118] S. Sarkar and S. Misra, "Theoretical modelling of fog computing: a green computing paradigm to support iot applications," Iet Networks, vol. 5, no. 2, pp. 23–29, 2016.

[119] F. Jalali, K. Hinton, R. Ayre, T. Alpcan, and R. S. Tucker, "Fog computing may help to save energy in cloud computing," IEEE Journal on Selected Areas in Communications, vol. 34, no. 5, pp. 1728–1739, 2016.

[120] T. Sen and H. Shen, "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, 2019, pp. 1–10.

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

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

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

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

[125] Z. Peng, D. Cui, J. Zuo, Q. Li, B. Xu, and W. Lin, "Random task scheduling scheme based on reinforcement learning in cloud computing," Cluster computing, vol. 18, no. 4, pp. 1595–1607, 2015.

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

[127] M. Cheng, J. Li, and S. Nazarian, "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, 2018, pp. 129–134.

[128] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in Proceedings of the AAAI conference on artificial intelligence, vol. 30, no. 1, 2016.

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

[130] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," arXiv preprint arXiv:1506.02438, 2015.

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

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

[133] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," arXiv preprint arXiv:1312.5602, 2013.

[134] W. Guo, W. Tian, Y. Ye, L. Xu, and K. Wu, "Cloud resource scheduling with deep reinforcement learning and imitation learning," IEEE Internet of Things Journal, vol. 8, no. 5, pp. 3576–3586, 2020.

[135] C. Xu, K. Wang, P. Li, R. Xia, S. Guo, and M. Guo, "Renewable energy-aware big data analytics in geo-distributed data centers with reinforcement learning," IEEE Transactions on Network Science and Engineering, vol. 7, no. 1, pp. 205–215, 2018.

[136] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," in NIPS'17: Proceedings of the 31st International Conference on Neural Information Processing Systems, 2017, pp. 6382—-6393.

[137] X. Li, W. Yu, R. Ruiz, and J. Zhu, "Energy-aware cloud workflow applications scheduling with geo-distributed data," IEEE Transactions on Services Computing, 2020.

[138] Z. Wen, J. Cała, P. Watson, and A. Romanovsky, "Cost effective, reliable and secure workflow deployment over federated clouds," IEEE Transactions on Services Computing, vol. 10, no. 6, pp. 929–941, 2016.

[139] Z. Wen, R. Qasha, Z. Li, R. Ranjan, P. Watson, and A. Romanovsky, "Dynamically partitioning workflow over federated clouds for optimising the monetary cost and handling run-time failures," IEEE Transactions on Cloud Computing, 2016.

[140] S. Iturriaga, S. Nesmachnow, A. Tchernykh, and B. Dorronsoro, "Multiobjective workflow scheduling in a federation of heterogeneous green-powered data centers," in 2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid). IEEE, 2016, pp. 596–599.

[141] Z. Wen, S. Garg, G. S. S. Aujla, K. Alwasel, D. Puthal, S. Dustdar, A. Y. Zomaya, and R. Rajan, "Running industrial workflow applications in a software-defined multi-cloud environment using green energy aware scheduling algorithm," IEEE Transactions on Industrial Informatics, 2020.

[142] C. Gu, C. Liu, J. Zhang, H. Huang, and X. Jia, "Green scheduling for cloud data centers using renewable resources," in 2015 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS). IEEE, 2015, pp. 354–359.

[143] H. Yuan, M. Zhou, Q. Liu, and A. Abusorrah, "Fine-grained resource provisioning and task scheduling for heterogeneous applications in distributed green clouds," IEEE/CAA Journal of Automatica Sinica, vol. 7, no. 5, pp. 1380–1393, 2020.

[144] D. Cheng, X. Zhou, Z. Ding, Y. Wang, and M. Ji, "Heterogeneity aware workload management in distributed sustainable datacenters," IEEE Transactions on Parallel and Distributed Systems, vol. 30, no. 2, pp. 375–387, 2018.

[145] D. Cui, Z. Peng, J. Xiong, B. Xu, and W. Lin, "A reinforcement learning-based mixed job scheduler scheme for grid or iaas cloud," IEEE Transactions on Cloud Computing, vol. 8, no. 4, pp. 1030–1039, 2017.

[146] J. Zhao, M. A. Rodríguez, and R. Buyya, "A deep reinforcement learning approach to resource management in hybrid clouds harnessing renewable energy and task scheduling," in 2021 IEEE 14th International Conference on Cloud Computing (CLOUD). IEEE, 2021, pp. 240–249.

[147] Y. Wang, H. Yao, and S. Zhao, "Auto-encoder based dimensionality reduction," Neurocomputing, vol. 184, pp. 232–242, 2016.

[148] T.-P. Pham, J. J. Durillo, and T. Fahringer, "Predicting workflow task execution time in the cloud using a two-stage machine learning approach," IEEE Transactions on Cloud Computing, vol. 8, no. 1, pp. 256–268, 2017.

[149] J. de Lope, D. Maravall et al., "Robust high performance reinforcement learning through weighted k-nearest neighbors," Neurocomputing, vol. 74, no. 8, pp. 1251–1259, 2011.