Future Generation Computer Systems 137 (2022) 14–30

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Deep reinforcement learning for energy and time optimized scheduling of precedence-constrained tasks in edge–cloud computing environments

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

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

ARTICLE INFO

Article history:
Received 22 April 2021
Received in revised form 8 June 2022
Accepted 16 June 2022
Available online 21 June 2022

Keywords:
Workflow scheduling
Edge-computing
Deep reinforcement learning
Proximal policy optimization
Energy efficiency
Internet of things

ABSTRACT

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

1. Introduction

Recent advances in Internet of Things (IoT) 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 and privacy and security concerns associated with sending data through public cloud platforms. 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

* Corresponding author.
E-mail address: amjayanetti@student.unimelb.edu.au (A. Jayanetti).

https://doi.org/10.1016/j.future.2022.06.012
0167-739X/© 2022 Published by Elsevier B.V.

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 [1] while others have suggested that depending on the network infrastructure and application characteristics it could be less energy-efficient in certain scenarios [2]. 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 [3–5]. Only very few studies have considered more complex workloads, such as those with precedence relations [6,7]. In the remainder of this paper, we use the term workflow to refer to workloads with precedence-constrained tasks. Workflow is an application model that can be used to represent a wide variety of IoT applications (health care, stream processing, smart city applications).

In this paper we focus on addressing the conflicting objectives of time minimization and energy optimization in scheduling DAG (Directed Acyclic Graphs) based workflows across cloud and edge computing environments. Specifically, we use Deep Reinforcement Learning (DRL) techniques which have proven to be efficient at handling highly dynamic and complex environments [8]. 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 a Reinforcement Learning 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 Deep Reinforcement Learning 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 Deep Reinforcement Learning model comprising of multiple 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. The results of the experiments clearly demonstrate that the proposed multi-actor technique performs better than the single-actor technique.

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

2. Related work

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

2.1. Cloud computing environments

A number of studies [9,10] have used RL for addressing the problem of task scheduling in cloud computing environments. In [9] 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 [10] 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 [11–16] for scheduling dependent tasks of workflows in cloud computing environments. In [11] 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 [12]. In [13], a multi-agent reinforcement learning framework for multi-objective workflow scheduling in cloud infrastructures is proposed. The proposed approach uses separate Deep Q Learning agents for each objective (cost and makespan) and the scheduling problem is designed as

Table 1
Summary of literature review.

Ref	Application model		Edge–cloud	Algorithm	Heterogeneous	Objectives	Shortcomings
	Workflow	Bag of tasks					
[9]		✓		Q-Learning	✓	energy	Over simplification of state-space
[10]		✓		Q-Learning	✓	response time	Over simplification of state-space
[16]	✓			Deep Q-Learning	✓	energy	Lack of coordination among agents may lead to a local optima
[11]	✓			Q-Learning	✓	energy, cost	Loss of accuracy due to discretization of state space
[12]	✓			SARSA, Genetic Algorithm	✓	makespan, resource utilization	High time and space complexity associated with SARSA
[13]	✓			Deep Q-Learning	✓	cost, makespan	Complexity associated with multi-agent coordination
[14]	✓			Deep Q-Learning	✓	makespan	High time and space complexity associated with Q table updates
[15]	✓			Deep Q-Learning	✓	makespan, energy	High time and space complexity associated with Q-learning
[3]		✓	✓	Q-Learning	✓	delay, energy	Loss of information due to state space discretization
[4]		✓	✓	Asynchronous Advantage Actor–Critic	✓	delay, energy, cost	Trained model cannot distinguish between edge and cloud nodes
[5]		✓	✓	Deep Q-Learning	✓	delay, cost	Trained model cannot distinguish between edge and cloud nodes
[17]		✓	✓	Deep Q-Learning	✓	delay, resource utilization	High computational power required for training (LSTM)
[6]	✓		✓	Temporal Difference Learning		delay, energy	Only addresses the problem of scheduling tasks among edge nodes
[7]	✓		✓	Deep Q-Learning		system cost	High computational power required for training (LSTM)
Proposed	✓		✓	Proximal Policy Optimization	✓	energy, makespan	

a Markov game with a correlated equilibrium. Deep Q Learning was also used in [14] 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 [16] for scheduling tasks of DAG based jobs with the objectives of minimizing energy cost of cloud service providers. In the first stage, the server farm to which a task should be allocated is determined. Second stage determines the exact server to which the task is allocated for execution.

2.2. Edge–cloud environments

A number of studies [3–5,17] have used RL for task scheduling in edge–cloud environments. A number of studies have used the popular TD learning based Q learning algorithm for enhancing the performance of task scheduling in edge–cloud systems. Q learning was used in [3] 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 (Bandwidth, 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. [5] proposed a Double Deep Q Learning algorithm for task scheduling in fog computing environment with the objectives of minimizing delay and computation cost. In [4], A3C (Asynchronous Advantage Actor–Critic) technique is used together with R2N2 (Residual Recurrent Neural Networks for task scheduling and migration in edge–cloud environment. [17] used Deep Q Learning coupled with LSTM (Long Short Term Memory Networks) for task scheduling in cloud computing environments with the aims of optimizing resource utilization and minimizing execution delay.

Several studies [6,7] used RL for scheduling dependent tasks in edge–cloud environment. [6] 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 [7], 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.

2.3. A qualitative analysis

A table summarizing the primary characteristics and limitations of existing works are presented in Table 1. 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 [3,9–11]. The scheduling algorithm proposed in [12] 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 [3] 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 [13,14,16]. 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 [5] for task scheduling overcomes this overestimation bias by decoupling action selection from evaluation with the use of two function approximators [18].

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 [7,17]. 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 [4] 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 [3–5,17], 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 [19] 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.

3. System model

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

3.1. Application model

Directed Acyclic Graph (DAG) is a popular execution model that can be used to represent a wide variety of applications. A workflow can be modeled 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_i. 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.

3.2. Network model

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

3.3. Delay model

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

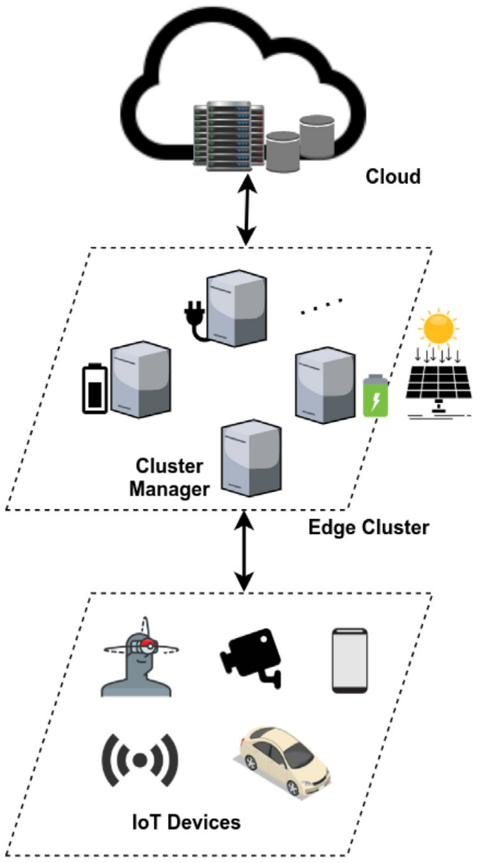


Fig. 1. System architecture.

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

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

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:

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

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

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

where $FT(t_i)$ is the finish 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:

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

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

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

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

3.4. Energy consumption model

Energy consumed during the execution of a workflow is the aggregate of the computation energy and communication energy incurred during the execution of workflow tasks. With CPU utilization based power consumption model [20] the energy consumed during the computation of task, t_j can be expressed as follows:

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

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:

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

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:

$$E = \sum_{t_j \in T} (ECOMP(t_j) + ECOMM(t_j)) \tag{8}$$

3.5. Deadline model

The primary goal of this work is to minimize energy consumption associated with workflow executions, and reduction in energy consumption is usually achieved at the expense of increased execution times. 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 is 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 paper, we transform the workflow scheduling problem to a task scheduling problem so that the scheduling decisions can be made in real time as tasks become ready for executions with the fulfillment 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, $r(t_j)$ of a task t_j [21] can be computed recursively with the following equation:

$$r(t_j) = \max_{t_k \in succ(t_j)} (TT(t_j, t_k) + r(t_k) + \overline{w(t_k)}) \tag{9}$$

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 Eq. (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 [22]:

$$d(t_j) = ST + r(t_A) + \overline{w(t_A)} - r(t_j) \tag{10}$$

where t_A is the source task of the workflow and $\overline{w(t_A)}$ is the average execution time of t_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 [22] addresses the aforementioned problem by setting the deadlines of tasks to the maximum values. Slack time (difference between earliest and latest finish times) is divided evenly among all tasks by setting the deadlines proportionally to their ranks. Accordingly, the deadline of a task is calculated as below:

$$d(t_j) = ST + (D - ST) * \frac{r(t_A) + \overline{w(t_A)} - r(t_j)}{r(t_A) + \overline{w(t_A)}} \tag{11}$$

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:

$$\begin{aligned}&\text{Minimize: } \sum_{n=1}^{N} \alpha M_n + (1-\alpha)E_n\\&\text{Subject to: } M_n \le D_n\end{aligned} \tag{12}$$

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

With the deadline decomposition technique introduced in previous subsection, an individual deadline, $d(t_j)$ is assigned to each sub task, t_j. Accordingly, for a system that schedules a set of workflows (W), constraint in Eq. (12) can be rewritten as:

$$\forall_{w \in W} \forall_{t_j \in T} \quad FT(t_j) \le d(t_j) \tag{13}$$

If task deadlines were set to minimize makespan as in Eq. (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 Eq. (13) is tighter than that in Eq. (12). However, since task deadlines are already set to maximum values by CP-P technique, the constraint imposed by the condition in Eq. (13) is equivalent to that in Eq. (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, the objective function for the system can be reformulated as:

$$\begin{aligned}&\text{Minimize: } \sum_{n=1}^{N} E_n\\&\text{Subject to: } \forall_{w \in W} \forall_{t_j \in T} \quad FT(t_j) \le d(t_j)\end{aligned} \tag{14}$$

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.

4.1. Background

Reinforcement learning is a branch of Machine Learning which operates by training an agent to learn a desired behavior in an interactive environment based on the experiences it 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 modeling 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, π. It is 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 π starting from state, s.

$$v_\pi(s) = E_\pi[G_t | s_t = s] \tag{15}$$

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

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{16}$$

γ 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, π is followed thereafter.

$$q_\pi(s, a) = E_\pi[G_t | s_t = s, a_t = a] \tag{17}$$

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 modeled as a parameterized function with respect to an adjustable parameter θ, as $\pi_\theta(a_t|s_t)$. In order to evaluate the performance of the policy, we define a performance objective $J(\theta)$, which can be defined as the expected return starting from the start state of the episode, s_0 and thereafter following the policy π_θ. This in fact is the value function of state, s_0 as expressed in Eq. (18).

$$J(\theta) = v_{\pi_\theta}(s_0) \tag{18}$$

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.

$$\theta_{t+1} = \theta_t + \alpha \nabla J(\theta_t) \tag{19}$$

Algorithm 1 Workflow Analyzing and Dispatching

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

7:  upon event Receipt of a task completion notification do
8:      if Completed task is a Sink task then
9:          Send results to user
10:     else
11:         Use the waiting-task map to identify successors
12:         for Each successor of completed task do
13:             if task is Ready then
14:                 Dispatch the task to task scheduler
15:             Update global waiting-task map
    return
```

where α is the learning rate. From the policy gradient theorem [23], the gradient of $J(\theta)$ can be expressed in the following manner:

$$\begin{aligned}\nabla J(\theta) &= E_\pi[\sum_a q_\pi(s_t, a)\nabla\pi(a|s_t, \theta)] \\ &= E_\pi[\sum_a \pi(a|s_t, \theta)q_\pi(s_t, a)\frac{\nabla\pi(a|s_t, \theta)}{\pi(a|s_t, \theta)}] \\ &= E_\pi[q_\pi(s_t, a_t)\frac{\nabla\pi(a_t|s_t, \theta)}{\pi(a_t|s_t, \theta)}] \\ &= E_\pi[G_t\frac{\nabla\pi(a_t|s_t, \theta)}{\pi(a_t|s_t, \theta)}]\end{aligned} \tag{20}$$

REINFORCE [24] is a popular policy gradient algorithm which uses the above derivation for updating policy parameters via gradient ascent as shown in Eq. (21). This however gives rise to slow convergence and high variance in gradient estimates due to the possibility of high deviations between trajectories. Therefore, we used Actor–Critic technique as the basis of our scheduling framework.

$$\begin{aligned}\theta_{t+1} &= \theta_t + \alpha G_t\frac{\nabla\pi(a_t|s_t, \theta_t)}{\pi(a_t|s_t, \theta_t)} \\ &= \theta_t + \alpha G_t \nabla \ln\pi(a_t|s_t, \theta)\end{aligned} \tag{21}$$

4.2. 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 [25], such approaches are less appropriate in a highly dynamic edge–cloud environment as the conditions may have significantly changed from the time scheduling decisions were made to the time the tasks (particularly the ones towards the bottom of the workflow) are actually scheduled for execution. Therefore, in this work we design the RL model in a manner such that all scheduling decisions are made in real time when workflow tasks are actually ready for execution.

For this we transform the complex workflow scheduling problem to a task scheduling problem, such that each scheduling decision corresponds to an allocation of a task whose precedence constraints are met, to an edge or cloud node for execution. Accordingly, the scheduler in gateway node maintains a map of pending tasks which await the completion of their predecessors for commencing execution. Whenever a task completes execution in an edge node or in the cloud, the gateway node is notified. With the receipt of this notification, the scheduler uses the proposed DRL based resource scheduling framework to determine where to execute any successors of the completed task which may now be in ready state. In the context of this problem, the DRL agent is the task scheduler which resides in the cluster manager. Algorithm 1 summarizes the sequence of events involved in the execution of aforementioned steps.

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

1. Total CPU and Memory requirements of the task, t_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 energy consumption of the system is minimized. Since the workflow scheduling problem is mapped to a task scheduling problem, from the perspective of the DRL agent, the goal would be 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 $(d_1, d_2, \dots, d_i)$ 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 [3,4] 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:

$$A = \{(a_1, a_2) | a_1 \in \{Cloud, Edge\} \ \& \ a_2 \in \{1, 2, \dots, N_{a_1}\}\} \tag{22}$$

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 [26] 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 [8].

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:

$$R_2 = \begin{cases} +1, & \text{if } FT(t_j) \le d(t_j) \\ -1, & \text{otherwise} \end{cases} \tag{23}$$

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

4.3. Actor–critic based scheduling framework with proximal policy optimization

Actor–critic is a branch of policy gradient algorithms that have proven to be efficient at overcoming the limitations of vanilla policy gradients by combining the advantages of value based methods and policy based methods. As the name implies, actor–critic algorithms consist of two main components; an actor and a critic. Actor is responsible for learning the policy, $\pi_\theta(a_t|s_t)$ which determines the action to be taken in each state for achieving the desired objective whereas critic is responsible for providing constructive criticism on the actions taken by the actor. In advantage actor–critic method [27], 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:

$$\theta_{t+1} = \theta_t + \alpha \hat{A}_t \nabla \ln \pi(a_t|s_t, \theta_t) \tag{24}$$

In this work, as the estimator, $\hat{A}_t$ of the advantage function we used the Generalized Advantage Estimator, $GAE(\gamma, \lambda)$ presented in [27], 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:

$$GAE(\gamma, 0) : \hat{A}_t := \delta_t = R_t + \gamma V(s_{t+1}) - V(s_t) \tag{25}$$

As expressed in Eq. (25), $\hat{A}_t$ is equal to δ_t in the special case where $\lambda = 0$. Note that we have used θ and ω 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 Eq. (26). And then updating the network parameters with TD error as shown in Eq. (27).

$$L(\omega) = \frac{1}{2}[v_\pi(s_t|\omega) - (R_t + v_\pi(s_{t+1}|\omega))]^2 \tag{26}$$

$$\begin{gathered} \omega \leftarrow \omega + \beta \delta_t \nabla v_\pi(s_t|\omega) \\ where\ \delta_t = R_t + v_\pi(s_{t+1}|\omega) - v_\pi(s_t|\omega) \end{gathered} \tag{27}$$

where β 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 Eq. (24), 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 (Eq. (19)) cannot be made too large since that could lead to large policy updates that collapses performance. Trust Region Policy Optimization (TRPO) [28] techniques address the aforementioned problem by maximizing a surrogate objective function subject to a constraint, σ as shown in Eq. (28). KL indicates the Kullback–Leibler divergence in Eq. (28). 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) [19] technique which is proven to provide the benefits of TRPO techniques, with the added advantages of less complexity, improved sample complexity and generalizability.

$$\begin{gathered} \text{Maximize: } \hat{E}_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}\hat{A}_t\right] \\ \text{Subject to: } \hat{E}_t[KL[\pi_{\theta_{\text{old}}}(.|s_t), \pi_\theta(.|s_t)]] \leqslant \sigma \end{gathered} \tag{28}$$

If the surrogate objective function of TRPO in Eq. (28) 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 Eq. (29). This is achieved through the clip function, clip($\mu_t(\theta)$, $1 - \epsilon$, 1

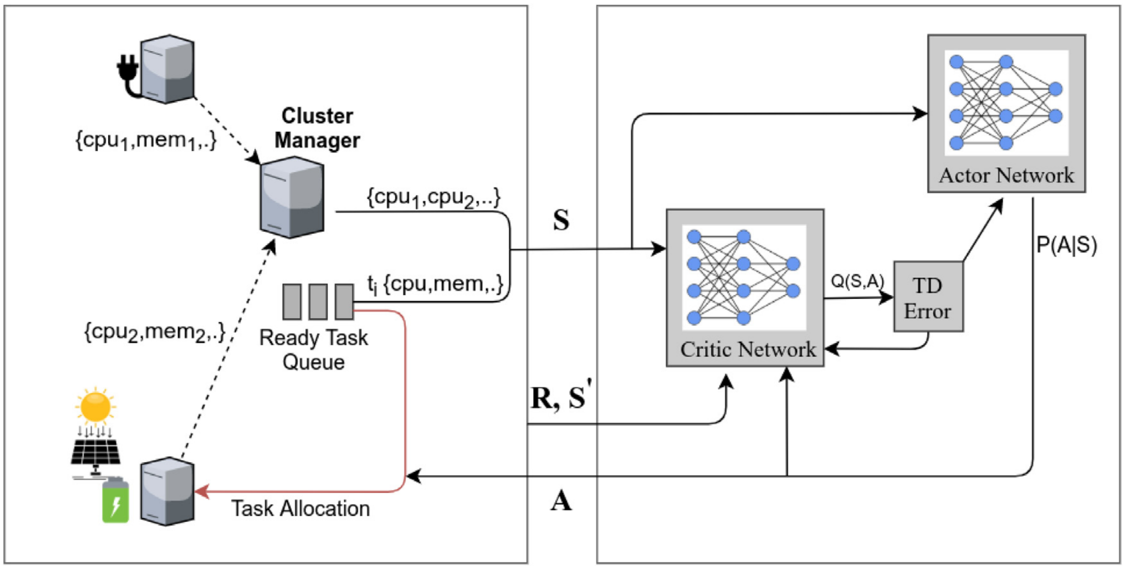


Fig. 2. Traditional actor–critic based scheduling.

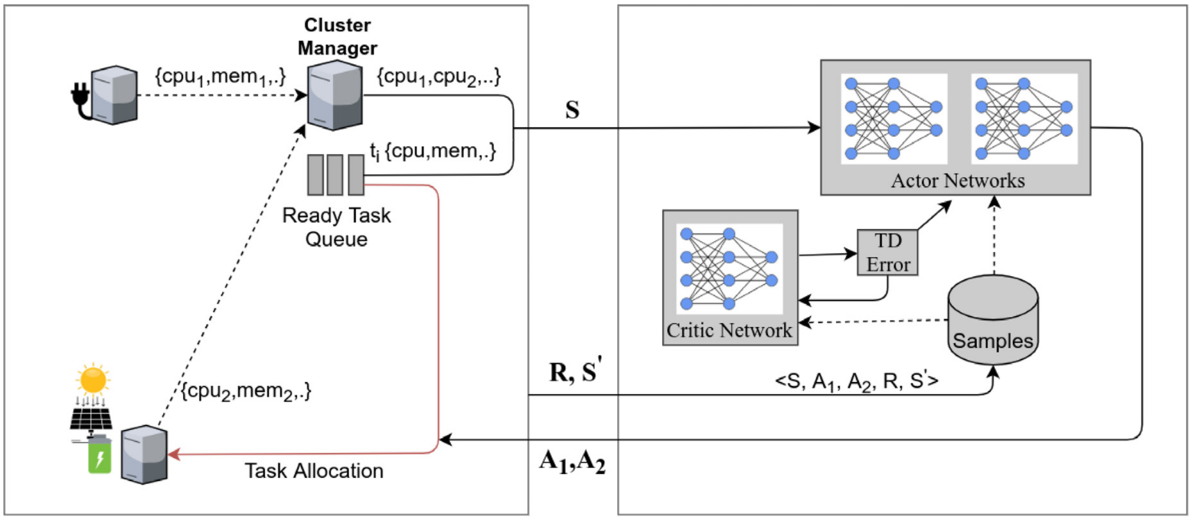


Fig. 3. Proposed scheduling framework.

$+\epsilon)\hat{A}_t$ which removes the desirability of large policy updates that changes the $r_t(\theta)$ ratio beyond the interval $[1-\epsilon, 1+\epsilon]$.

$$L^{CLIP}(\theta) = \hat{E}_t[\min(\mu_t(\theta))\hat{A}_t, \text{clip}(\mu_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t]$$
$$\text{where } \mu_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \tag{29}$$

As described in Section 4.2, we have proposed a novel hierarchical action space for the task scheduling problem in edge–cloud environment. We then adapted a hybrid actor–critic technique [26] designed for parameterized action spaces to the hierarchical action space in our problem.

Figs. 2 and 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 multiple actors and one critic. All actor networks use the same state space described in Section 4.2. Critic network also uses the same state space since it is trained to learn the state-value function and not the state–action value function which requires action input as well. The advantage given by the critic network is used to update the 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.

Pseudocode of the training process of the proposed multi-actor scheduling framework is presented in Algorithm 2. As indicated in lines 1–2 we first initialize the actor networks and critic network with random weights. Then the training parameters are also initialized. We train the DRL model for a total of N episodes (line 3), and at the beginning of each episode, the environment state is

Algorithm 2 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
```

Algorithm 3 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
```

reset. Since, the problem is modeled as an input driven MDP, each time-step of the episode actually corresponds to scheduling of a task from Ready Task Queue. As indicated in lines 6–8, at each time-step the current state of the environment is given as input to the actor networks, and the output of the first actor network provides the tier to which the task should be allocated, and the second actor network's output provides the node in the selected tier to which the task should be allocated. Upon the execution of combined action (allocation of task to the selected node), the agent receives a reward and the environment transitions to a new state (line 9). Details of the transition which includes state of the environment, combined action, reward and next state are stored in memory as indicated in line 10. For each transition, the advantage estimates are also calculated and stored. At the end of each episode, we train the networks K times with randomly sampled mini-batch samples of size S (line 12–18). 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 3 summarizes the steps involved in online task scheduling process. As training the DRL model is a resource intensive and time consuming process, the DRL model is pre-trained and used in real time for obtaining the scheduling decisions. Real-time network status of all nodes together with the resource requirements of the task to be scheduled (dequeued from ready task queue) are merged for formulating the current state of the environment. It is then provided as input to the actor networks. The task is then allocated to the tier and node given by the combined action output of the actor networks.

5. Performance evaluation

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

5.1. Experimental setup

For evaluating the performance of proposed workflow scheduling framework, we used an extension [30] 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 [31].

The simulated scenario comprises of a highly heterogeneous cluster with 16 edge nodes and 8 cloud nodes. We used the SPEC benchmark [29] for obtaining the resource configurations and power consumption rates of nodes as shown in Table 2. Following the simulation experiments of similar works [4] based on empirical studies, communication delay between edge–edge nodes and edge–cloud nodes was considered to be 1 ms and 10 ms respectively.

5.2. Dataset

Evaluation dataset was created based on synthetic workflow structures [32] provided by the popular Peagasus workflow framework. Task length in terms of number of instructions (as per CloudSim nomenclature) and the sizes of precedence constraints (in megabytes) were randomly selected from the ranges 0.5k to 1000k, and 0.1k to 10k respectively. A total of 1000 workflows comprising of 5292 tasks was used for the experiments. For simulating workflow arrival times, we used a Poisson 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 [21]. 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 [5] which is a constant value for each resulting critical path execution time, to derive deadlines that are realistically achievable in the simulated environment. For this we used a trial and error method where the target being the selection of a deadline base with which EFT (Earliest Finish Time) algorithm (described in Section 5.3) can meet approximately 90% of deadlines.

Table 2
Host configurations derived from SPEC benchmark [29] for experimental setup.

Layer	Server name	Processor	Cores	MIPS	RAM	Bandwidth	Power (Watts)	
					(GB)	(GB/s)	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 3
Hyper-parameters used for the DRL model.

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

5.3. Comparison algorithms

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

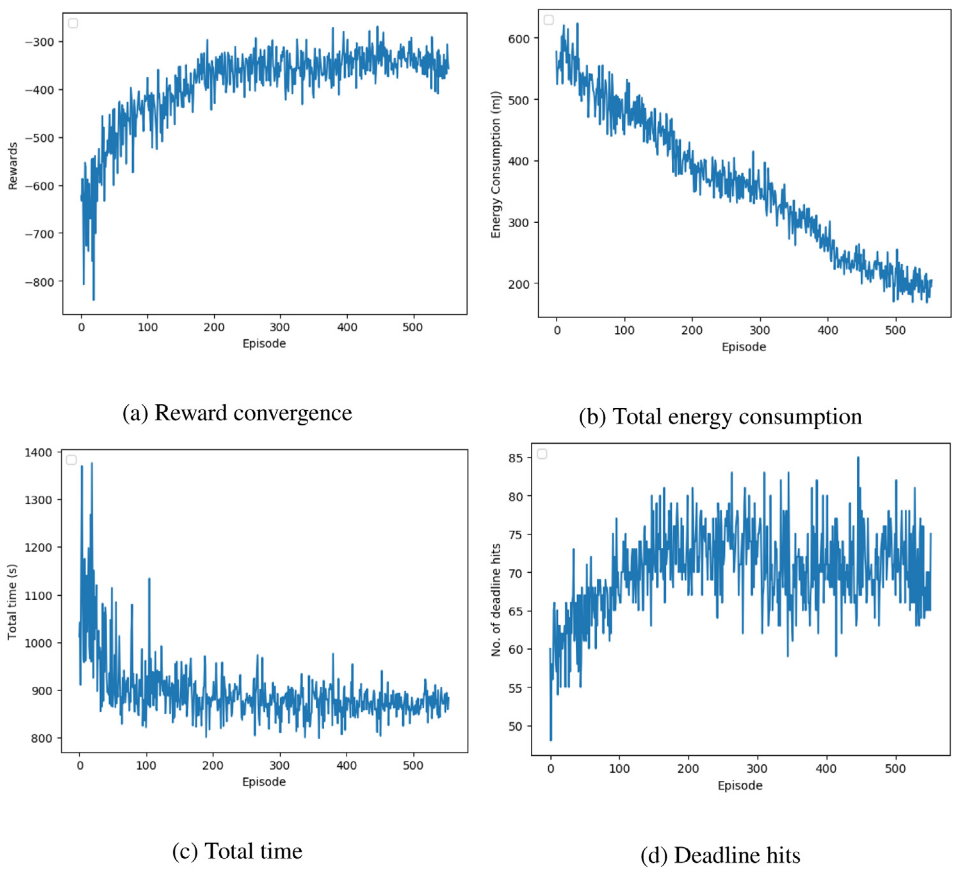
1. **Random:** This is a baseline algorithm which assigns tasks to a randomly selected node.
2. **EFT:** This algorithm is similar to the popular HEFT [21] algorithm except for the insertion based scheduling policy which is impractical in the considered edge–cloud scenario due to the lack of control 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 Eqs. (1)–(4) for computing the estimated finish time of the task to be scheduled in all of the nodes, and assigns the task to the node with the earliest finish time. Where there are multiple ready tasks, tasks are prioritzed based on their upward ranks.
3. **EDA** This is an energy and delay aware algorithm which operates by assigning tasks to nodes that can reduce both delay as well as energy associated with task execution. Accordingly, it uses Eqs. (4) and (6) to compute the product of estimated delay and energy for executing a task at each of the nodes and selects the highest ranked node based on a score calculated as follows:
$$SCORE = FT(t_i) \times ECOMP(t_i) \tag{30}$$
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 Eq. (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.

5.4. Hyper-parameters and network configurations

Table 3 lists the hyper-parameters used for training the agents. The critic network is set to learn at a faster rate than the actor networks, since the actor networks rely on the guidance of the critic network, a critic network with a relatively faster learning rate speeds up the learning process. Furthermore, a step learning curve is used for the second actor, and thereby the first actor is set to learn at a slower rate than the second actor for the first 100 episodes. This is because during early episodes of training the reward largely depends on the actions produced by the second actor, so regardless of how good first actor's action is in a given state, the reward could still be bad due to second actor's action. Therefore, large weight updates for the first actor at early stages of training more often leads to sub-optimal convergence. 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.

5.5. Analysis of convergence

Fig. 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 Fig. 4a, total rewards accumulated during an episode gradually increase and converges to a maximum around the 550th 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 Fig. 4b, and reaches a minimum around the 550th episode. As a part of the reward is designed to reward



(a) Reward convergence (b) Total energy consumption (c) Total time (d) Deadline hits

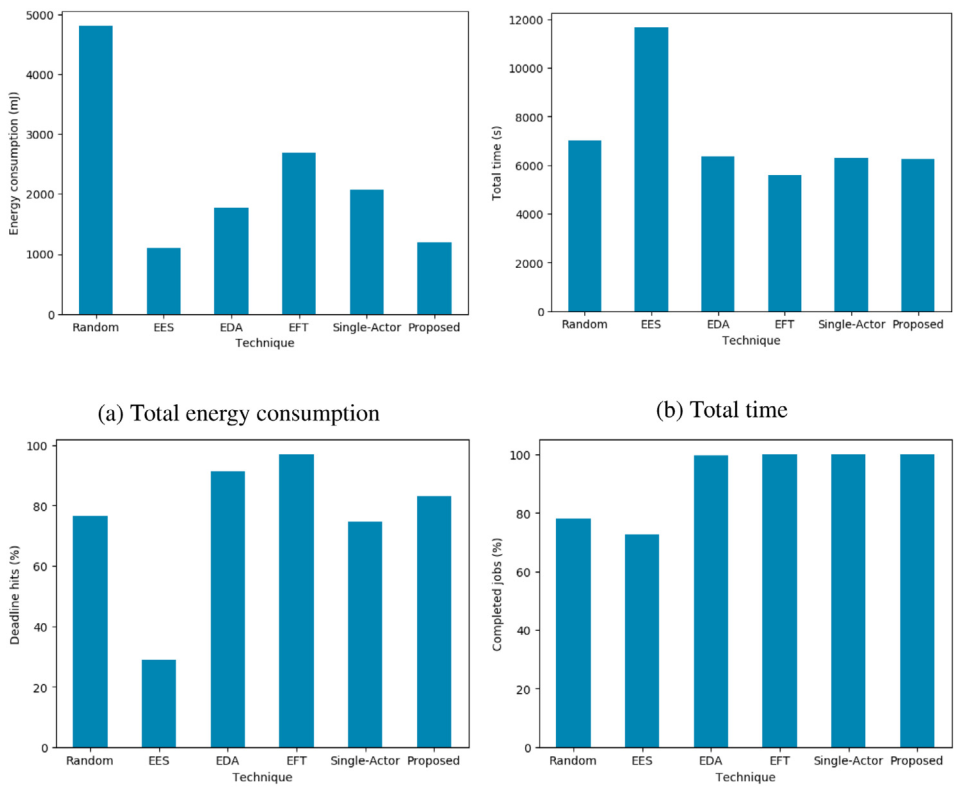
Fig. 4. Learning progress with training (convergence of DRL model).

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 Fig. 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 (Fig. 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.

5.6. Analysis of performance on experimental dataset

Fig. 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 Fig. 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.

Fig. 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 communication times leading to increased total execution time. An improvement of 11% and 47% over the execution times of Random and EES algorithms is achieved by the proposed DRL framework, respectively. EDA, 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



(a) Total energy consumption

(b) Total time

(c) Deadline hits

(d) Completed jobs

Fig. 5. Comparison of performance of scheduling algorithms on experimental dataset.

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 Figs. 5c and 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 finish time of workflow tasks. The next highest number of deadline hits (84%) is achieved by the proposed DRL technique. Note that in this work we have considered deadlines to be soft deadlines, which means meeting them is desirable but not mandatory. Accordingly, the DRL model was not trained to meet all deadlines as a hard constraint, rather the training objective was formulated to primarily minimize energy consumption while using deadlines to control the degree to which makespan of workflows is allowed to increase in exchange for higher energy savings. 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 scheduling decisions given by Random and EES algorithms result in timed out tasks which in turn lead to incomplete jobs as shown in Fig. 5d.

5.7. Analysis of performance at different workflow arrival rates

Fig. 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 Fig. 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 Fig. 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 Figs. 6c and 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 behavior since the DRL models were trained at a moderate arrival rate, so as the

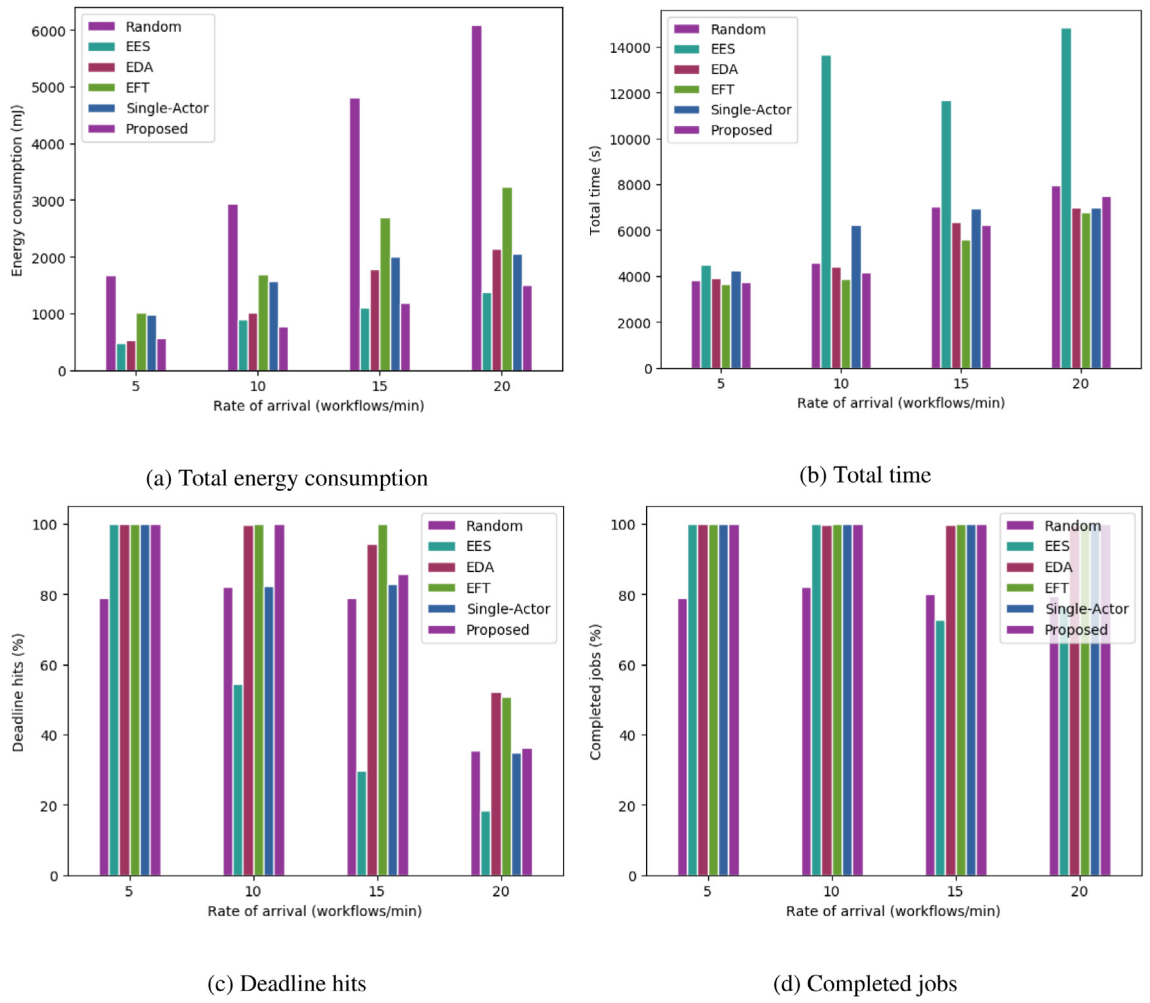


Fig. 6. Comparison of performance of scheduling algorithms at different workflow arrival rates.

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 [33]. 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 [33].

5.8. Analysis of performance at different computational workloads

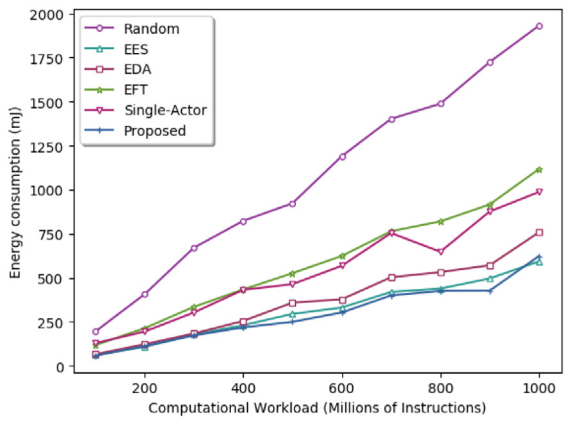
Fig. 7 demonstrates the performance of the algorithms as the computational workload varies. Computational workload in Fig. 7 represents the aggregate computational workload of all workflows scheduled by the system. In Fig. 7a, Random algorithm's performance with respect to energy consumption severely 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 Fig. 7b since it has completed less jobs compared to other algorithms as indicated in Fig. 7d.

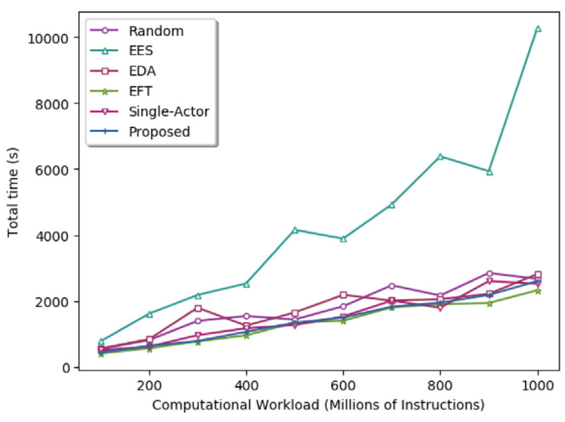
In Fig. 7c, EES algorithms ability to meet deadlines sharply drops with increasing workload. This is due to the fact that EES algorithm focuses solely on minimizing energy consumption, and therefore the allocation decisions it makes lead to increased execution delays. This in turn delays the completion 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 previously discussed, this is expected since the sole focus of EFT algorithm is to speed up the execution of tasks which in turn leads to higher deadline hits.

6. Overall analysis

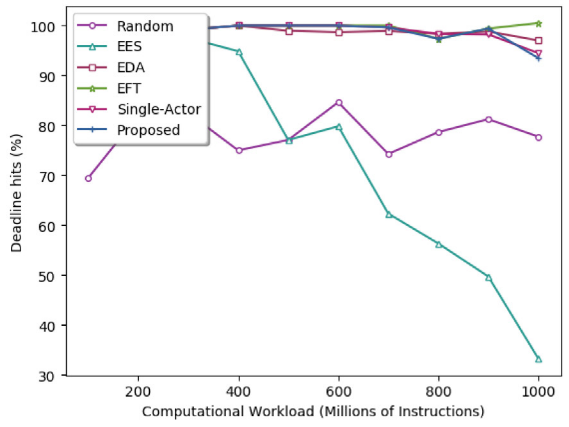
As evident through the results of experiments in Figs. 5a, 6a and 7a, EES algorithm is the most efficient at minimizing energy consumption of workflow executions. This is because its



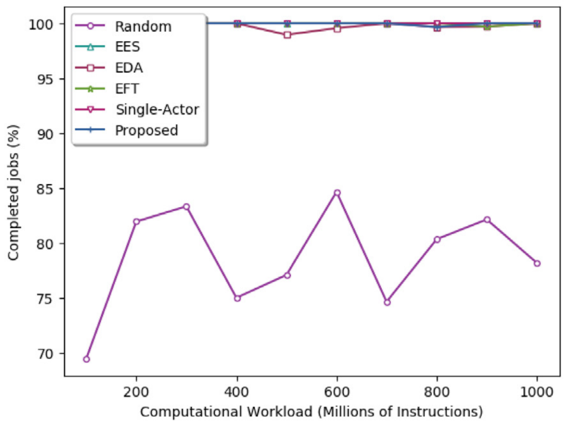
(a) Total energy consumption



(b) Total time



(c) Deadline hits



(d) Completed jobs

Fig. 7. Comparison of performance of scheduling algorithms at different computational workloads.

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 Figs. 5b, 6b and 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 4.2.

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 (Figs. 5c, 6c and 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.

7. Conclusions and future work

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

In this work we propose a novel hierarchical state space formulation coupled with a hybrid actor–critic technique for energy-efficient resource scheduling in edge–cloud environment. The resulting 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 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 scheduling in highly dynamic and complex edge computing environments while minimizing energy consumption as well as execution time. A limitation of the proposed reinforcement learning framework is that it is designed to operate in a centralized manner. As part of the future work, we will extend the proposed reinforcement learning framework to operate in a distributed manner. Furthermore, we will implement the proposed deep reinforcement learning framework based scheduling framework in a real edge-cloud environment. We will also enhance the proposed framework by leveraging long short term memory technique for enabling the deep reinforcement learning model to maintain an internal representation of history, which is proven to improve the ability of the deep reinforcement learning agents to adapt to varying conditions. Furthermore, we also intend to apply variance reduction techniques for stabilizing the performance of deep reinforcement learning agents in the presence of highly different job arrival rates.

CRediT authorship contribution statement

Amanda Jayanetti: Conceptualization, Data curation, Methodology, Software, Validation, Writing – original draft. **Saman Halgamuge:** Supervision, Methodology, Writing – review & editing. **Rajkumar Buyya:** Conceptualization, Supervision, Methodology, Visualization, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] S. Sarkar, S. Misra, Theoretical modelling of fog computing: a green computing paradigm to support IoT applications, Iet Netw. 5 (2) (2016) 23–29.

[2] F. Jalali, K. Hinton, R. Ayre, T. Alpcan, R.S. Tucker, Fog computing may help to save energy in cloud computing, IEEE J. Sel. Areas Commun. 34 (5) (2016) 1728–1739.

[3] T. Sen, 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.

[4] S. Tuli, S. Ilager, K. Ramamohanarao, R. Buyya, Dynamic scheduling for stochastic edge-cloud computing environments using a3c learning and residual recurrent neural networks, IEEE Trans. Mob. Comput. (2020).

[5] P. Gazori, D. Rahbari, M. Nickray, Saving time and cost on the scheduling of fog-based IoT applications using deep reinforcement learning approach, Future Gener. Comput. Syst. (2019).

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

[7] H. Lu, C. Gu, F. Luo, W. Ding, X. Liu, Optimization of lightweight task offloading strategy for mobile edge computing based on deep reinforcement learning, Future Gener. Comput. Syst. 102 (2020) 847–861.

[8] H. Mao, M. Alizadeh, I. Menache, S. Kandula, Resource management with deep reinforcement learning, in: Proceedings of the 15th ACM Workshop on Hot Topics in Networks, 2016, pp. 50–56.

[9] D. Ding, X. Fan, Y. Zhao, K. Kang, Q. Yin, J. Zeng, Q-learning based dynamic task scheduling for energy-efficient cloud computing, Future Gener. Comput. Syst. 108 (2020) 361–371.

[10] Z. Peng, D. Cui, J. Zuo, Q. Li, B. Xu, W. Lin, Random task scheduling scheme based on reinforcement learning in cloud computing, Cluster Comput. 18 (4) (2015) 1595–1607.

[11] A. Asghari, M.K. Sohrabi, F. Yaghmaee, A cloud resource management framework for multiple online scientific workflows using cooperative reinforcement learning agents, Comput. Netw. 179 (2020) 107340.

[12] A. Asghari, M.K. Sohrabi, F. Yaghmaee, Task scheduling, resource provisioning, and load balancing on scientific workflows using parallel SARSA reinforcement learning agents and genetic algorithm, J. Supercomput. 77 (3) (2021) 2800–2828.

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

[14] A. Kaur, P. Singh, R. Singh Batth, C. Peng Lim, Deep-Q learning-based heterogeneous earliest finish time scheduling algorithm for scientific workflows in cloud, Softw. - Pract. Exp. (2020).

[15] Y. Qin, H. Wang, S. Yi, X. Li, L. Zhai, An energy-aware scheduling algorithm for budget-constrained scientific workflows based on multi-objective reinforcement learning, J. Supercomput. 76 (1) (2020) 455–480.

[16] M. Cheng, J. Li, 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.

[17] G. Rjoub, J. Bentahar, O. Abdel Wahab, A. Saleh Bataineh, Deep and reinforcement learning for automated task scheduling in large-scale cloud computing systems, Concurr. Comput.: Pract. Exper. (2020) e5919.

[18] H. Van Hasselt, A. Guez, D. Silver, Deep reinforcement learning with double q-learning, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 30, (1) 2016.

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

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

[21] H. Topcuoglu, S. Hariri, M.-y. Wu, Performance-effective and low-complexity task scheduling for heterogeneous computing, IEEE Trans. Parallel Distrib. Syst. 13 (3) (2002) 260–274.

[22] M. Żotkiewicz, M. Guzek, D. Kliazovich, P. Bouvry, Minimum dependencies energy-efficient scheduling in data centers, IEEE Trans. Parallel Distrib. Syst. 27 (12) (2016) 3561–3574.

[23] R.S. Sutton, A.G. Barto, Reinforcement Learning: An Introduction, MIT Press, 2018.

[24] R.J. Williams, Simple statistical gradient-following algorithms for connectionist reinforcement learning, Mach. Learn. 8 (3–4) (1992) 229–256.

[25] M. Sharifi, S. Shahrivari, H. Salimi, PASTA: A power-aware solution to scheduling of precedence-constrained tasks on heterogeneous computing resources, Computing 95 (1) (2013) 67–88.

[26] Z. Fan, R. Su, W. Zhang, 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.

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

[28] J. Schulman, S. Levine, P. Abbeel, M. Jordan, P. Moritz, Trust region policy optimization, in: International Conference on Machine Learning, 2015 pp. 1889–1897.

[29] SPEC, SPEC, "Standard performance evaluation corporation", 2018, URL https://www.spec.org/benchmarks.html.

[30] J. Son, A.V. Dastjerdi, R.N. Calheiros, X. Ji, Y. Yoon, 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.

[31] F. Chollet, et al., Keras: The python deep learning library, Ascl (2018) ascl–1806.

[32] L. Ramakrishnan, D. Gannon, A survey of distributed workflow characteristics and resource requirements, Indiana Univ. (2008) 1–23.

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

Amanda Jayanetti is working towards a Ph.D. degree at the Cloud Computing and Distributed Systems (CLOUDS) Laboratory, Department of Computing and Information Systems, the University of Melbourne, Australia. Her research interests include Fog/Edge Computing, Internet of Things (IoT) and Distributed Systems, Artificial Intelligence. She is currently working on leveraging the power of Artificial Intelligence techniques for resource management in edge and cloud computing environments.

Prof. Saman Halgamuge is a Fellow of IEEE, a Professor in the School of Electrical, Mechanical and Infrastructure Engineering and a Distinguished Speaker/Lecturer on Computational Intelligence (2019–2021). He served as Director/Head, Research School of Engineering of the Australian National University (ANU) (2016–18), a member of Australian Research Council (ARC) College of Experts for Engineering, Information and Computing Sciences (2016–18), the founding Director of the Ph.D. training centre. Melbourne India Postgraduate Program (MIPP) of University of Melbourne and Associate Dean (2013–15) and Assistant Dean (2008–13) in International Engagement in the Melbourne School of Engineering.

Prof. Rajkumar Buyya is a Redmond Barry Distinguished Professor and Director of the Cloud Computing and Distributed Systems (CLOUDS) Laboratory at the University of Melbourne, Australia. He has authored over 625 publications and seven text books including "Mastering Cloud Computing" published by McGraw Hill, China Machine Press, and Morgan Kaufmann for Indian, Chinese and international markets respectively. He is one of the highly cited authors in computer science and software engineering worldwide (h-index=138, g-index=280, 103,400+ citations).