Deep Reinforcement Learning for Dynamic Task Scheduling in Edge-Cloud Environments

Original Scientific Paper

D. Mamatha Rani*
TGSWRAFPDC(W), Bhongir
Department of Computer Science, Bhongir, Telangana- 508126, India
mamatha3004@gmail.com

Supreethi K P
Jawaharlal Nehru Technological University
Department of Computer Science and Engineering, Hyderabad, Telangana- 500085, India
supreethi.pujari@jntuh.ac.in

Bipin Bihari Jayasingh
CVR College of Engineering/IT Department
Hyderabad, Telangana- 501510, India
bipinbjayasingh@cvr.ac.in

*Corresponding author

Abstract *– With The advent of the Internet of Things (IoT) and its use cases there is a necessity for improved latency which has led to edgecomputing technologies. IoT applications need a cloud environment and appropriate scheduling based on the underlying requirements of a given workload. Due to the mobility nature of IoT devices and resource constraints and resource heterogeneity, IoT application tasks need more efficient scheduling which is a challenging problem. The existing conventional and deep learning scheduling techniques have limitations such as lack of adaptability, issues with synchronous nature and inability to deal with temporal patterns in the workloads. To address these issues, we proposed a learning-based framework known as the Deep Reinforcement Learning Framework (DRLF). This is designed in such a way that it exploits Deep Reinforcement Learning (DRL) with underlying mechanisms and enhanced deep network architecture based on Recurrent Neural Network (RNN). We also proposed an algorithm named Reinforcement Learning Dynamic Scheduling (RLbDS) which exploits different hyperparameters and DRL-based decision-making for efficient scheduling. Real-time traces of edge-cloud infrastructure are used for empirical study. We implemented our framework by defining new classes for CloudSim and iFogSim simulation frameworks. Our empirical study has revealed that RLbDS out performs many existing scheduling methods.*

Keywords: *Task Scheduling, Edge-Cloud Environment, Recurrent Neural Network, Edge Computing, Cloud Computing, Deep Reinforcement Learning*

Received: November 17, 2023; Received in revised form: June 12, 2024; Accepted: June 17, 2024

1. INTRODUCTION

Unprecedented growth of cloud-assisted use cases has led to compelling Cloud Service Providers (CSPs) to optimize resource usage in the presence of Service Level Agreements (SLAs). Ubiquitous adoption of technological innovation such as the Internet of Things (IoT) has led to the emergence of fog and edge computing phenomena which leverage latency. In the presence of IoT applications, the scheduling of tasks is challenging for many reasons such as network hierarchy, heterogeneity of resources, mobility of devices, resource-constrained devices and stochastic behaviour of nodes [1]. Traditional cloud scheduling algorithms are not sufficient to harness the power of the dynamic computing environment made up of cloud, fog and edge resources. To overcome this problem, different scheduling algorithms came into existence. Reinforcement learning is one such technique used with the machine learning approach [2]. Many learning-based task scheduling approaches came into existence. Their merits and demerits are summarized in Table 1 and Table 2 in Section 2. The advantages of the research in [1] include consideration of dynamic environments and heterogeneous cores. However, it does not consider adaptive QoS, edge cloud, decentralized environment and presence of stochastic workloads.

The work in [3] considered edge cloud and also heterogeneous cores for their task scheduling research.

However, it does not support adaptive QoS, dynamic and decentralized environments, edge cloud and stochastic workloads. The merits of [4] include the consideration of dynamic environment, stochastic workloads and heterogeneous cores. But it lacks adaptive QoS, support for edge cloud and decentralized environments. The research in [5] and [6] has similar findings. Their method has provision for considering dynamic environments, heterogeneous cores, adaptive QoS and stochastic workloads. But is not designed for edge cloud and decentralized environments. In [7], there is consideration of dynamic environment, stochastic workloads, adaptive QoS and heterogeneous cores but does not support decentralized and edge-cloud environments. The work in [8] supports dynamic environments and stochastic workloads. However, it has limitations to deal with heterogeneous cores, adaptive QoS, edge cloud and decentralized environments. There is a similarity in the task scheduling methods proposed in [9] and [10].

Their methods are dynamic supporting adaptive QoS and stochastic workloads besides dealing with heterogeneous cores. However, they do not support decentralized and edge cloud environments. The scheduling research in [11] supports dynamic environments along with stochastic workloads. They also deal with heterogeneous cores and adaptive QoS. However, the drawback is that those methods do not consider decentralized and edge-cloud environments.

Concerning optimization parameters, Table 2 provides research gaps in existing solutions. Research in [1] is based on a heuristics approach and considers energy and SLA violation parameters. Their research lacks in the study of response time and cost of scheduling which are crucial for task scheduling. The work in [3] is also based on the heuristics method but considers cost and energy parameters. It does not throw light on response time and SLA violations. In [4], their method is based on Gaussian process regression and considers two parameters such as energy and SLAs. It has no support for optimization of cost and response time.

The task scheduling research in [5] and [6] is based on the Deep Queue Learning Network (DQN) method and supports cost and energy parameters for optimization. However, they have no optimization of SLAs and response time. In [7] Q-learning-based phenomenon is used considering energy and cost dynamics for optimization. However, it lacks optimization of response time and SLAs. Deep Neural Network (DNN) is the scheduling method used in [8] and it has support for optimization of cost and SLA parameters. It lacks support for energy and response time optimizations. The work in [9] and [10] is based on the Double DQN (DDQN) method and it supports only energy parameters for optimization. It lacks support for response time, cost and SLA optimizations. In [11] DRL method is used for task scheduling by considering response time for optimization. However, it does not support the optimization of SLAs, cost and energy. From the literature, it is observed that there is a need for a more comprehensive methodology in edge-cloud environments for task scheduling. Our contributions to this paper are as follows.

1. We proposed a learning-based framework known as the Deep Reinforcement Learning Framework (DRLF). This is designed in such a way that it exploits Deep Reinforcement Learning (DRL) with underlying mechanisms and enhanced deep network architecture based on Recurrent Neural Network (RNN).
2. We proposed an algorithm named Reinforcement Learning Dynamic Scheduling (RLbDS) which exploits different hyperparameters and DRL-based decision-making for efficient scheduling.
3. Our simulation study has revealed that the proposed RLbDS outperforms many existing scheduling methods.

The remainder of the paper is structured as follows. Section 2 reviews prior works on existing task scheduling methods for cloud and edge-cloud environments. Section 3 presents details of the proposed system including the system model, DRL mechanisms and the underlying algorithm. Section 4 presents the results of the empirical study while Section 5 concludes our work and provides directions for the future scope of the research.

2. RELATED WORK

This section reviews prior works on existing task scheduling methods for cloud and edge-cloud environments. VM plays a vital role in cloud infrastructure for resource provisioning. Beloglazov and Bu proposed a method for improving resource utilization in the cloud through VM migration and consolidation. They found that VM live migration has the potential to exploit idle nodes in cloud data centres to optimize resource utilization and reduce energy consumption. They considered the dynamic environment and presence of heterogeneous cores for their task scheduling study. Their method is based on a heuristics approach. It considers SLA negotiations and algorithms designed to support optimizations such as energy efficiency and SLAs. Their algorithm monitors VMs and their resource usage. By considering VM consolidation and VM live migration, their method is aimed at reducing energy consumption and adherence to SLAs. This method lacks adaptive QoS and support for dynamic workloads. Pham and Huh [3] proposed a task scheduling method based on a heuristics approach for such an environment.

It is designed to work for heterogeneous cores in fog-cloud. They considered optimizations such as energy efficiency and cost reduction by scheduling tasks in an edge-cloud environment. Their algorithm is based on heuristics towards reducing cost and energy consump-

tion. It is based on graph representation. Towards this, their method exploits the task graph and processor graph. Given the two graphs representing tasks and resources, their method finds appropriate resource allocation for given tasks. It has a provision for determining task priority and then choosing the most suitable node for the execution of the task.

Bui et al. [4] proposed an optimization framework for the cloud with a predictive approach. They could predict the dynamics of resource utilization for scheduling by employing a method named Gaussian process regression. The prediction result helped them to minimize the number of servers to be used to process the requests leading to a reduction of energy usage. Their method is, however, based on heuristics and is not suitable for dynamic workloads and edge-cloud environments. Cheng et al. [2] explored DRL based approach towards task scheduling and resource provisioning in the cloud. They further optimized the Q-learning method to reduce the task rejection rate and improve energy efficiency. Huang et al. [5] and Mao et al. [6] followed the DRL approach for improving task scheduling performance in a cloud computing environment.

In [5] DRL based online offloading method is proposed based on deep neural networks. It is a scalable solution since it is a learning-based approach. In [6] DeepRM is the framework proposed for task scheduling considering efficient resource management. Both methods are based on the DQN approach rather than heuristics. Both methods considered optimization parameters such as energy and cost. In other words, they are designed to reduce energy consumption and also the cost incurred for task execution in cloud environments. They support stochastic workloads and adaptive QoS. However, they do not support edge-cloud environments and do not optimize SLA and response time parameters.

Basu et al. [7] focused on the problem of live migration of VMs based on the RL-based Q-learning process. Their methodology improves live migration and heuristics-based existing approaches. Towards this end, their method exploits the Megh and RL-based model to have continuous adaptation to the runtime situations towards leveraging energy efficiency. Xu et al. [8] defined a DNN approach named LASER to support deadline-critical jobs with replication and speculative execution. Their implementation of the framework is designed for the Hadoop framework. Zhang et al. [9] defined a DDQN method towards energy efficiency in edge computing. It is based on the Q-learning process and also the dynamic voltage frequency scaling (DVFS) method that has the potential to reduce energy usage. As Q-learning is not able to recognize continuous system states, they extended it to have double-deep Q-learning. Table 1 shows provides a summary of findings among existing scheduling methods.

Table 1. Merits and demerits of existing scheduling methods compared with the proposed method

Reference	Dynamic	Stochastic Workload	Decentralized	Edge Cloud	Adaptive QoS	Heterogeneous
[1]	Yes	No	No	No	No	Yes
[3]	No	No	No	Yes	No	Yes
[4]	Yes	Yes	No	No	No	Yes
[5], [6]	Yes	Yes	No	No	Yes	Yes
[7]	Yes	Yes	No	No	Yes	Yes
[8]	Yes	Yes	No	No	No	No
[9], [10]	Yes	Yes	No	No	Yes	Yes
[11]	Yes	Yes	No	No	Yes	Yes
[18]	Yes	No	No	No	Yes	Yes
[19]	Yes	No	No	Yes	Yes	No
[20]	Yes	No	No	No	Yes	Yes
[21]	Yes	No	No	No	Yes	Yes
[22]	Yes	No	No	No	Yes	Yes
[23]	Yes	Yes	No	No	Yes	Yes
[25]	Yes	No	No	No	No	No
[26]	Yes	No	No	No	Yes	Yes
[27]	Yes	No	Yes	No	Yes	Yes
Proposed (RLbDS)	Yes	Yes	Yes	Yes	Yes	Yes

Similar to the work of [2], Mao et al. [6] employed DDQN for efficient resource management. This kind of work is also found in Li et al. [10]. Both have employed the DRL technique towards job scheduling over diversified resources. However, these learning-based methods are not able to withstand stochastic environments. Mao et al. [6] and Rjoubet al. [11] investigated DRL based approach for task scheduling in edge-cloud. However, they considered only response time in their research. Its drawback is that they could not exploit asynchronous methods for optimization of their methods towards robustness and adaptability. There is a need to improve it by considering the dynamic optimization of parameters in the presence of stochastic workloads. Skarlat et al. [12] explored IoT service placement dynamics in fog computing resources while Pham et al. [13] focused on cost and performance towards proposing a novel method for task scheduling. Brogi and Forti [14] investigated on deployment of QoS-aware IoT

tasks in fog infrastructure. Task prioritization [15], DRL for resource provisioning [4, 7], energy-efficient scheduling using Q-learning [16] and DRL usage in 5G networks [17] are other important contributions.

As presented in Table 1, we summarize our findings leading to important research gaps. The summary is made in terms of different parameters such as dynamic environment, presence of stochastic workload, decentralized environment, usage of edge cloud, consideration for adaptive QoS and presence of heterogeneous cores for task scheduling. Table 1 also provides the proposed method and its merits over existing methods.

Almutairi and Aldossary [18] proposed a novel method for IoT tasks to offload in the edge-cloud ecosystem. It is designed to serve latency-sensitive applications in a better way. It has a fuzzy logic-based approach for inferring knowledge towards decision-making in the presence of resource utilization and dynamic resource utilization. Ding et al. [19] considered an edge-cloud environment to investigate stateful data stream applications. They proposed a method to judge state migration overhead and make partitioning decisions based on the dynamically changing network bandwidth availability. Murad et al.

[20] proposed an improved version of the min-min task scheduling method to deal with scientific workflows in cloud computing. It could reduce the minimum completion time besides optimizing resource utilization. Bulej et al. [21] did their research on the management of latency in the edge-cloud ecosystem towards better performance in task scheduling in the presence of dynamic workloads. It is designed to explore the upper bound of response time and optimize the performance further. Almutairi and Aldossary [22] proposed an edge-cloud system architecture to investigate modelling methodology on task offloading. It has offloading latency models along with various offloading schemes. Their simulations are made using Edge CloudSim. They intend to improve it in future with fuzzy logic.

Zhang and Shi [23] explored workflow scheduling in an edge-cloud environment. They analyzed different possibilities in workflow scheduling in such an ecosystem. They opined that workflow applications need novel approaches in the scheduling process. Zhao et al. [24] focused on task scheduling along with security to prevent intrusions in edge computing environments. They considered low-rate intrusions and focused on preventing them along with task scheduling. It is a Q-learning-based approach designed to meet runtime requirements based on the learning process. Zhang et al. [25] proposed a time-sensitive algorithm that dynamically caters to the needs of deadline-aware tasks in edge-cloud environments. It considers job size and server capability in a given dynamic and hierarchical scenario. It is a multi-objective task considering execution time, cost and reduction of SLAs. Lakhan et al. [26] proposed a task scheduling approach for IoT tasks considering a hybrid mechanism consisting of task scheduling and task offloading. Singh and Bhushan [27] proposed a method for task scheduling based on Cuckoo Search Optimization (CSO). It has an integrated local search strategy. From these recent works, it is found that they targeted IoT kind of workflows in edge-cloud environments. There is Q-Learning used in one of the papers. However, deep reinforcement learning is not found in the latest works. Service placement in edge resources using DRL [28], dynamic scheduling [29] and task offloading [30] are other important contributions. Table 2 provides a summary of findings among existing scheduling methods in terms of optimization parameters. Magotra [41] focused on energy-efficient approaches in cloud infrastructures by developing adaptive solutions that could help the system towards proper VM consolidation, leading to better performance.

Table 2. Optimization parameters considered by existing scheduling methods

Reference	Method	Optimization Parameters			
		SLA Violations	Cost	Response Time	Energy
[1]	Heuristics	Yes	No	No	Yes
[3]	Heuristics	No	Yes	No	Yes
[4]	Gaussian Process Regression	Yes	No	No	Yes
[5], [6]	DQN	No	Yes	No	Yes
[7]	Q Learning	No	Yes	No	Yes
[8]	DNN	Yes	Yes	No	No
[9], [10]	DDQN	No	No	No	Yes
[11]	DRL (REINFORCE)	No	No	Yes	No
[18]	SJF	No	No	Yes	Yes
[19]	Cloud Computing	No	No	Yes	No
[21]	Cloud computing	No	Yes	Yes	Yes
[23]	CSA	No	Yes	No	No
[24]	Cloud computing	No	Yes	Yes	Yes
[25]	Cloud computing	No	Yes	No	No
[27]	CSP	No	Yes	Yes	No

As presented in Table 2, we summarized the existing methods in terms of optimization parameters and the approach considered in the task scheduling research. The optimization parameters considered for the comparative study of existing methods are SLA violations, cost, response time and energy.

Table 2 also provides the proposed method and its merits over existing methods. Table 1 and Table 2 provide very useful insights reflecting gaps in the research. Our work in this paper is based on such research gaps as those tables reveal the merits of the proposed system.

3. PROPOSED SYSTEM

We proposed a DRL-based framework for dynamic task scheduling in an edge-cloud environment. This section presents the framework and proposed algorithm besides DRL mechanisms.

3.1. PROBLEM DEFINITION

Considering an edge-cloud environment, let H be a collection of hosts denoted as $\{H_1, H_2, H_3, \ldots, H_n\}$ where n indicates a maximum number of hosts. A task T can be assigned to host H. Scheduling is considered as the assignment of T to H. However, in terms of RL, the system state is mapped to an action. Here action does mean allocation of T to H. T may be an active task that could be migrated to a new H or a newly arrived task. At the beginning of an interval, denoted as SI_i, The system state initially is denoted as state$_i$ which reflects the hosts and their parameters, tasks yet to be allocated in the prior interval, denoted as $(a_{i-1} \setminus l_i)$ beside newly arrived tasks denoted as n_i. For each task, denoted as a_i $(=a_{i-1} \cup n_i \setminus l_i)$, the scheduler needs to take an action, denoted as $Action_i$, for the system interval SI_i in terms of either allocating it to a host or migrating to a new host. A task is satisfying Let $m_i \subseteq a_{i-1} \setminus l_i$ is considered a migratable task. A scheduler can be understood as a model which reflects a decision-making function $State_i \rightarrow Action_i$. Here loss function associated with the model for a given interval denoted as $Loss_i$, is computed based on task allocations. Therefore, the problem of realizing an optimal model is expressed in Eq. 1.

$$\begin{aligned} &Minimize_{Model} \sum_i Loss_i \\ &\text{Subject to } \forall \text{ i}, Action_i = \text{Model } (State_i) \\ &\forall \text{ i } \forall \text{ T} \in m_i \cup n_i, \{T\} \leftarrow Action_i(T) \end{aligned} \quad (1)$$

Different notations used in our work are presented in Table 3.

3.2. OUR SYSTEM MODEL

We considered infrastructure or resources for scheduling in an edge-cloud environment. The resources are heterogeneous. Edge resources are nearby while cloud resources reside in a remote data centre. Therefore, each host in the infrastructure is different in response time and computational power. Edge resources are closer and exhibit low response times but they do have limited resources and computational power. Cloud resources take more response time but they do have high computational power. Our system model is presented in Fig. 1.

The edge and cloud nodes are part of computing resources. These resources are managed by the resource management module. This module has several components or sub-modules to deal with resource management either directly or indirectly. The scheduler module is responsible for either scheduling a task T to a host H or migrating a task from one host to another host based on runtime dynamics. The dynamic workload is generated by IoT devices being used by different users. The workload contains several tasks with varied requirements. Resource management module takes the workload and follows DRL based (learning-based) approach in task allocation or task migration. These decisions are based on the ideal objective functions and the requirements associated with tasks. The requirements may include deadline, bandwidth, RAM and CPU.

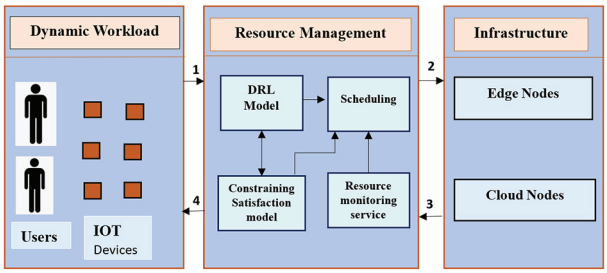

Fig. 1. Our system model

The workload is generated automatically to evaluate the functionality of the proposed system. Our system has a DRL model which influences the scheduler module in decision-making. There are multiple schedulers to be used at runtime to serve dynamically generated workloads. In the process, there is the distribution of workload among hosts leading to faster convergence. Each resource in edge-cloud accumulates local gradients associated with corresponding schedulers besides synchronizing them to update models. The DRL module follows asynchronous updates. The constraint satisfaction module takes suggestions as input from DRL and finds whether it is valid. Here valid does mean a task is in migration or the host's capacity is optimally being used.

3.3. WORKLOAD GENERATION

We generate workload programmatically to evaluate the proposed system. Since IoT devices and user's demands are dynamic, there is a change in the bandwidth and computational requirements of tasks. The whole execution time in our system is divided into several scheduling intervals. Each interval is assumed to have the same duration. SI_i denotes the i^{th} scheduling interval. This interval has a start time and end time denoted as t_i and t_{i+1} respectively. Each interval has active tasks associated with it. They are the tasks being executed and denoted as a_i. The tasks that have been completed at the beginning of the interval are denoted as l_i while

newly arrived tasks that are dynamically generated by the workload generator are denoted as n_i.

3.4. OUR LEARNING-BASED APPROACH FOR SCHEDULING

We proposed a framework known as the Deep Reinforcement Learning Framework (DRLF), as shown in Fig. 2, which exploits a learning-based approach using the DRL model for dynamic task scheduling in an edge-cloud environment. The framework supports several scheduling intervals. The framework has a workload generator which generates tasks (n_i) and gives them to the scheduling and migration module. The tasks given to the scheduler are in turn given to the resource monitoring module which schedules new tasks and migrates existing tasks if required to ensure optimal resource utilization, load balancing and latency in task completion. The scheduler activity changes the state of the edge-cloud environment.

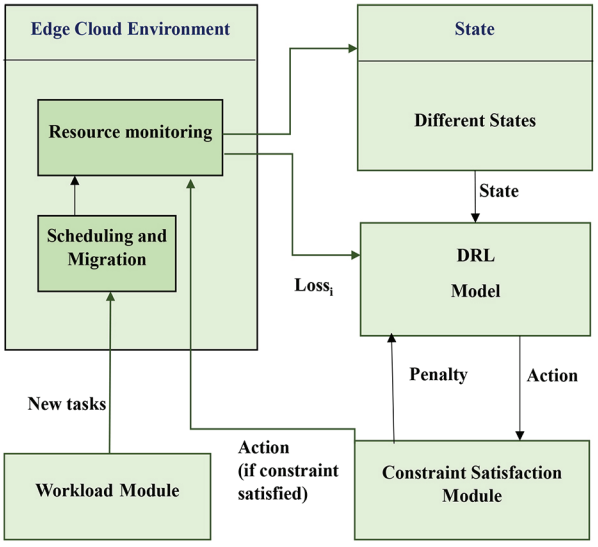

Fig. 2. Proposed Deep Reinforcement Learning Framework (DRLF) for task scheduling in edge-cloud environment

Every time $State_i$ is updated by the resource monitoring module it is given to the DRL model. The state information consists of hosts' feature vectors, new tasks n_i and the rest of the tasks associated with the previous interval and denoted by $(a_{i-1} \backslash l_i)$. The resource monitoring module also gives $Loss_i$ data to the DRL model. The DRL model suggests an action, denoted as $Action_{i-1}^{PG}$, based on the state information to the constraint satisfaction module and updates parameters as expressed in Eq. 2. This module then determines $Penality_i$ to the DRL model.

$$Loss_i^{PG} = Loss_i + Penality_i \quad (2)$$

This process continues iteratively. Once the constraint is satisfied, the constraint satisfaction module gives the suggested action (Action)by the DRL module to the resource management module. It then computes $Penality_{i+1}$ about SI_{i+1} the next scheduling interval.

Table 3. Notations used in our work

Notation	Description
a_i	Indicates a set of active tasks linked to SI_i
H_i	Indicates i^{th} host in a given set of hosts
I_i	Indicates the initial set of tasks of SI_i
m_i	Indicates a decision for task migration
n_i	It indicates a task allocation decision
$Action_i^{PG}$	Scheduling actions at the beginning of SI_i
$Loss_i^{PG}$	Loss function at the beginning of SI_i
SI_i	Denotes i^{th} scheduling interval
T_i^S	It indicates i^{th} in a given set of tasks
$\{T\}$	Indicates the host to which task T has been assigned
AEC	Average Energy Consumption
AMT	Average Migration Time
ART	Average Response Time
Hosts	Indicates a collection of hosts in the edge-cloud environment
N	Indicates the maximum number of hosts
T	Denotes a task to be executed

Based on the action received from the constraint satisfaction module, the resource management module either allocates a new task to a specific host or migrates tasks, denoted as $(a_{i-1} \backslash l_i)$, of the preceding interval. This will result in an update from a_{i-1} to a_i. Then the tasks associated with a_i are execute for SI_i and the cycle continues for SI_{i+1}.

3.5. DEEP LEARNING ARCHITECTURE

The DRL model is built based on an enhanced Recurrent Neural Network (RNN) architecture. It has the functionality to achieve reinforcement learning. In the process, it approximates $State_i$ towards $Action_i^{PG}$ which is an action bestowed from the DRL model to the constraint satisfaction module for a given scheduling interval. The enhanced RNN can ascertain temporal relationships between input space and output space. This deep learning architecture is shown in Fig. 3. After each interval, cumulative loss and policy are predicted by a single network

The network has two fully connected layers, denoted as $fc1$ and $fc2$, configured. These are followed by three recurrent layers, denoted as $r1$, $r2$ and $r3$, with skip connections. The given 2D input is flattened and sent to dense layers. The output of $r3$ is given to two fully connected layers denoted as $fc3$ and $fc4$. The $fc4$ outputs a 2D vector of 100x100. It does mean that the model can deal with 100 tasks allocated to 100 hosts in cloud infrastructure. Eventually, a softmax function is employed to the second dimension to have values [0,1] and the resultant value in a row becomes 1. For interpretation O_{jk}, denoting a probability map, indicates that there is a probability of a task T_j^{ai} being assigned to H_k. At the $fc4$, a cumulative loss function $Loss_{i+1}^{PG}$ is computed. The layers in the network are made up of a Gated Recurrent Unit that have the capacity to model the temporal dimension of a given task and also the

characteristics of the host comprising of bandwidth, RAM and CPU. The Gated Recurrent Unit (GRU) layers tend to have increased network parameters leading to complexity. This problem is addressed by exploiting skip connections towards gradient propagation faster.

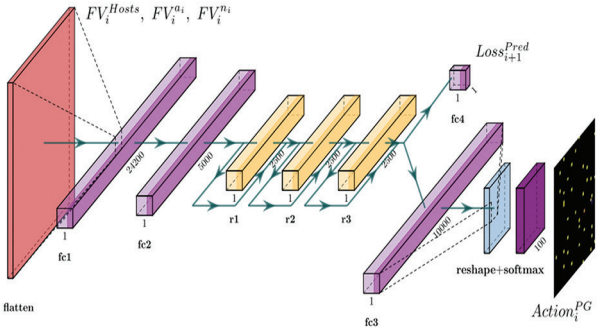

Fig. 3. Architecture of an RNN variant used to realize the DRL model

This model takes $State_i$ as input which is represented in the form of a 2D vector. This vector contains a continuous element FV_i^{Hosta}, and another continuous element $FV_i^{n_i}$ and $FV_i^{a_{i-1}\backslash l_i}$ has categorical host indices. Therefore, pre-processing is required to transform host indices into one hot vector with a maximum size of n. Then there is a need for the concatenation of all feature vectors. Afterwards, each element in the resultant vector is subjected to normalization based on a range of values [0, 1]. Each element has a feature denoted as f_e while min_{fe} and max_{fe} denote their minimum and maximum values respectively. These values are computed relying on the dataset with the help of two heuristics namely local regression and maximum migration time. Afterwards, standardization is carried out feature-wise using the expression in Eq. 3.

$$e=\begin{cases} 0 \; if \; max_{f_e} = min_{f_e} \\ \min\left(1, \max\left(0, \frac{e - min_{f_e}}{max_{f_e} - min_{\;e}}\right)\right) \text{ otherwise} \end{cases} \quad (3)$$

Once pre-processing of the given input is carried out, it is fed to the network (Fig. 3) where it first flattens the pre-processed input before sending it through dense layers. The output of these layers is transformed into $Action_i^{PG}$. We employed a backpropagation algorithm to ascertain the biases and weights of the network. The learning rate is kept adaptive from 10 to 2 and later on, 1/10th value based on reward change associated with the preceding 10 iterations is not greater than 0.1. Automatic differentiation is exploited to modify the parameters of the network using $Loss_i^{PG}$ as a reward. Gradients of local networks are accumulated across the edge nodes periodically in an asynchronous fashion towards the update of global network parameters. Towards this end, a gradient accumulation rule expressed in Eq. 4 is followed.

$$d\theta \leftarrow d\theta - \alpha\nabla_\theta, \log\left[\pi(State_i; \theta')\left(Loss_i^{PG} + CLoss_{i+1}^{pred}\right)\right] + \alpha\nabla_\theta, (Loss_i^{PG} + CLoss_{i+1}^{pred} - CLoss_{i+1}^{pred})^2 \quad (4)$$

Where local and global network parameters are denoted as θ' and θ respectively, it has a log term to indicate a change direction in parameters and the ($Loss_i^{PG}$+$CLoss_{i+1}^{pred}$) term denotes cumulative loss predicted in a given episode that begins with *State s*. Mean Square Error (MSE) is a gradient associated with the cumulative loss predicted. Finally, there is the transformation of output from $Action_i^{PG}$ to $Action_i$ by the constraint satisfaction module and the same is given to the resource management module.

3.6. Algorithm Design

We proposed an algorithm to realize the optimal scheduling of given tasks in the edge-cloud ecosystem. It is presented in Algorithm 1.

Algorithm: Reinforcement Learning based Dynamic Scheduling (RLbDS)

Inputs:

Size of batch **B**

Maximum intervals for scheduling N

1. Begin
2. For each interval n in N
3. IF n%B==0 and n>1 Then
4. Compute loss function
5. $Loss_i^{PG}$=$Loss_i$+$Penality_i$
6. Use $Loss_i^{PG}$ in the network (Fig. 3) for backpropagation
7. End If
8. $State_i$←$PreProcess$($State_i$)
9. Feed $State_i$ to the network (Fig. 3)
10. pMap←Output of RL model (network as in Fig. 3)
11. ($Action$, $Penality_{i+1}$)←ConSatMod(map)
12. Resource monitoring module takes *action*
13. DRL model takes $Penality_{i+1}$
14. ResourceMonitoring($Action_i$) migrates active task
15. Execution of all tasks in interval n in edge-cloud
16. End For
17. End

Algorithm 1. Reinforcement Learning based Dynamic Scheduling (RLbDS)

The algorithm takes the size of batch B and maximum intervals for scheduling N and performs optimal scheduling of given tasks of every interval in edge-cloud resources. The algorithm exploits the enhanced RNN network (Fig. 3) to update the model from time to time towards making DRL-based decisions for scheduling. At each interval of scheduling, there is an iterative process for taking care of pre-processing and feeding the state to the DRL model. Based on the action suggested by DRL, the constraint satisfaction module specifies a penalty when there is an ideal scheduling decision, that is notified to resource monitoring which schedules new tasks and also performance migration of active tasks based on the decisions rendered.

3.7. LOSS FUNCTION COMPUTATION

In the proposed learning model we want to optimize, in each interval, with minimal $Loss_i$. The model is also designed to adapt to the state that dynamically changes while mapping $State_i$ to $Action_i$. Towards this end, $Loss_i$ is a metric defined to update model parameters. Besides different metrics that result in normalized value 0 or 1 are defined. Average energy consumption is a metric defined as the edge cloud resources have different sources of energy as discussed in [32]. The consumed energy by host $h \in Hosts$ is multiplied by a factor $\alpha_h \in [0, 1]$ that is associated edge-cloud deployment strategy. The normalized AEC is computed as in Eq. 5.

$$AEC_i^{Hosts} = \frac{\sum_{h \in Hosts} \alpha_h \int_{t=t_i}^{t_{i+1}} p_h(t)dt}{\sum_{h \in Hosts} \alpha_h p_h^{max}(t_{i+1}-t_i)}, \quad (5)$$

Where the power function of host h is denoted by $p_h(t)$ linked to time and its maximum possible power is denoted as p_h^{max}.

Average response time is another metric defined to be used for interval SI_i. ART for all tasks is normalized by maximum response time. ART is computed as in Eq. 6.

$$ART_i = \frac{\sum_{t \in l_{i+1}} \text{Response Time(t)}}{|l_{i+1}| max_i \, max_{t \in l_i} \text{Response Time(t)}} \quad (6)$$

The average migration time metric is defined for a given SI_i. It reflects all tasks' average migration time in the interval normalized by maximum migration time. AMT is computed as in Eq. 7.

$$AMT_i = \frac{\sum_{t \in a_i} \text{Migration Time(t)}}{|a_i| max_i \, max_{t \in l_i} \text{Response Time(t)}}. \quad (7)$$

Cost (C) is yet another metric defined for SI_i. It indicates the total incurred cost in the interval and is computed as in Eq. 8.

$$Cost_i = \frac{\sum_{h \in Hosts} \int_{t=t_i}^{t_{i+1}} C_h t(t)dt}{\sum_{h \in Hosts} C_h^{max}(t_{i+1}-t_i)} \quad (8)$$

Average SLA violation is another metric for SI_i. It reflects SLA violation dynamics as expressed in Eq. 9.

$$SLVA_i = \frac{\sum_{t \in l_{i+1}} SLA(t)}{|l_{i+1}|} \quad (9)$$

To minimize the resultant value for all the aforementioned metrics, as used in [16] and [33], the $Loss_i$ metric is defined as expressed in Eq. 10.

$$Loss_i = \alpha . AEC_{i-1} + \beta . ART_{i-1} + \gamma . AMT_{i-1} + \delta . Cost_{i-1} + \epsilon . SLVA_{i-1} \quad (10)$$

such that $\alpha, \beta, \gamma, \delta, \in \geq 0 \wedge \alpha + \beta + \gamma + \delta + \in = 1$.

Different users can have varied QoS needs and hyper-parameters (α, β, γ, δ, $\in$) need to be set with different values. As discussed in [33], [34] and [35] it is important to optimize energy consumption in cloud infrastructure. Therefore, it is essential to optimize loss. Even when other metrics are compromised, it is possible to optimize loss. In such a case, the loss can have $\alpha = 1$ while the other metrics can have 0. As discussed in [36] traffic management and healthcare monitoring are sensitive to response time. In such cases, loss can have $\beta = 1$ while other measures can have 0. In the same fashion, setting hyper-parameters is application-specific.

$$Loss_i^{PG} = Loss_i + Penality_i \quad (11)$$

As specified in the works such as [37] and [38], the penalty is to be included in neural network modes. With the penalty, the model can update parameters towards minimizing $Loss_i$ and ensure constrained satisfaction. Therefore, for neural network loss function is defined as in Eq. 11.

4. RESULTS AND DISCUSSION

This section presents our simulation environment, the dataset used and the results of experiments.

4.1. SIMULATION SETUP

We built a simulation application using Java language. The IDE used for development is the IntelliJ Idea 2022 version. CloudSim [39] and iFogSim [40] libraries are used to have a simulation environment. Scheduling intervals are considered equal to be compatible with other existing works [4, 7, 41]. Cloudlets or tasks are generated programmatically from the Bitbrain dataset collected from [42].

The two simulation tools such as iFogSim and CloudSim are extended with required classes to facilitate the usage of cost, response time and power parameters associated with edge nodes. New modules are created to incorporate simulation of IoT devices with mobility with delayed task execution, variations in bandwidth and communication with deep learning model. Additional classes are defined to have constraint satisfaction modules and also take care of input formats, output formats and pre-processing. Based on the provision in CloudSim, a loss function is implemented. The dataset collected from [43] has traces of real workload run on Bitbrain infrastructure. This dataset contains logs of workloads of more than 1000 VMs associated with host machines. The workload information contains time-stamp, RAM usage, CPU usage, CPU cores requested, disk, network and bandwidth details. This dataset is available at [44] to reproduce our experiments. The dataset is divided into 75% and 25% VM workloads for training and testing respectively. Training deep learning model is done with the former while the latter is used to test the network and analyse results.

4.2. ANALYSIS OF RESULTS

We evaluated the performance of the proposed algorithm named RLbDS by comparing it with state-of-the-art methods such as Local Regression and Minimum Migration Time (LR-MMT) [41], Median Absolute Deviation and Maximum Correlation Policy (MAD-MC) [41], DDQN [44] and REINFORCE [9]. LL-MMT works for dynamic workloads

considering minimum migration time and local regression. It has heuristics to have task selection and overhead detection. MAD-MC is also a dynamic scheduler which is based on maximum correlation and median absolute deviation heuristics. DDQN is a deep learning-based approach that exploits RL to schedule tasks. DRL method is also based on RL which is based on policy gradient. The results reveal the sensitivity dynamics hyperparameters, such as (α, β, γ, δ, ε), of the proposed RLbDS about model learning and its impact on different performance metrics. Model training is given with 10 days of simulations while testing is carried out with 1-day simulation time.

4.2.1. Impact of Hyperparameters on RLbDS

The performance of the proposed algorithm named RLbDS is analysed with loss function associated with many hyperparameters such as (α, β, γ, δ, ε). Experiments are made with value 1 set to each of the hyperparameters. The rationale behind this is that when the value is set to 1, it could provide optimal performance.

Table 4. Performance of RLbDS with different hyper parameters

Model Parameters	Total Energy (Watts)	Time (milliseconds)	Fraction of SLA Violations	Total Cost (USD)	Time (seconds)	Number of completed tasks
α=1	1.37	8.5	0.17	6305.5	4.45	815
β=1	1.43	8.18	0.17	6306.5	4.3	830
γ=1	1.51	8.8	0.148	6307.5	3.65	845
δ=1	1.38	8.78	0.178	6304.5	4.15	810
ε=1	1.44	8.22	0.134	6307.8	3.75	850

As presented in Table 5, the performance of RLbDS is provided in terms of the number of performance metrics.

Table 5. Performance of RLbDS compared against existing algorithms

Models	Total Energy (Watts)	Time (milliseconds)	Fraction of SLA Violations	Total Cost (US Dollar)	Time (seconds)	Number of completed tasks
LR-MMT	0.959	8.58	0.06	6325	4.5	700
MAD-MC	0.95	8.4	0.13	6325	4.3	800
DDQN	0.85	8.8	0.07	6325	4	850
REINFORCE	0.82	8.35	0.06	6300	3.8	850
RLbDS	0.73	7.7	0.04	6000	3.3	1000

Loss function with different hyperparameters has its influence on the performance of the RLbDS algorithm as presented in Fig. 4. The network learning process differs with changes in hyperparameters. Energy consumption differed when the loss function used different hyperparameters. With α=1 RLbDS consumed 1.37 watts, with β=1 it needed 1.43 watts, with γ=1 the algorithm consumed 1.51 watts, with δ=1 it required 1.38 watts and with ε=1 RLbDS consumed 1.44 watts. The least energy is consumed when α=1 (all energy consumption values are given in 1*108 format). The average response time of the algorithm RLbDS is influenced by each hyperparameter. With α=1 RLbDS required 8.5 milliseconds, with β=1 it needed 8.18 milliseconds, with γ=1 the algorithm needed 8.8 milliseconds, with δ=1 it required 8.78 milliseconds and with ε=1 RLbDS required 8.22 milliseconds. The least response time is recorded when β=1.

SLA violations are also studied with these hyperparameters. It is observed that they influence a fraction of SLA violations. With α=1 the fraction of SLA violations caused by RLbDS is 0.17, with β=1 also it is 0.17, with γ=1 the algorithm showing 0.148, with δ=1 it is 0.178, and with ε=1 RLbDS caused by 0.134. The last fraction of SLA violations is recorded when ε=1. The total cost is also analysed in terms of USD (as per the pricing calculator of Microsoft Azure [45]).

It was observed earlier that hyperparameters have an impact on energy consumption. Since energy consumption attracts the cost of execution in the cloud, obviously these parameters have an impact on the cost incurred. With α=1 the total cost exhibited by RLbDS is 6305.5, with β=1 it is 6306.5, γ=1 the algorithm showed 6307.5, with δ=1 it is 6304.5, and with ε=1 RLbDS caused 6307.8. The least cost is recorded when δ=1.

Average task completion time is also analysed with different hyperparameters. With α=1 the average task completion time exhibited by RLbDS is 4.45 seconds, with β=1 it is 4.3, with γ=1 the algorithm showed 3.65, with δ=1 it is 4.15, and with ε=1 RLbDS caused 3.75. The least average task completion time is recorded when γ=1 (all average task completion values are given in 1*106 format). The total number of tasks completed with scheduling done by RLbDS is also influenced by hyperparameters. With α=1 the number of completed tasks achieved by RLbDS is 815, β=1 it is 830, γ=1 the algorithm showed 845, with δ=1 it is 810, and with ε=1 RLbDS showed 850 tasks to be completed. The least number of completed tasks is recorded when δ=1.

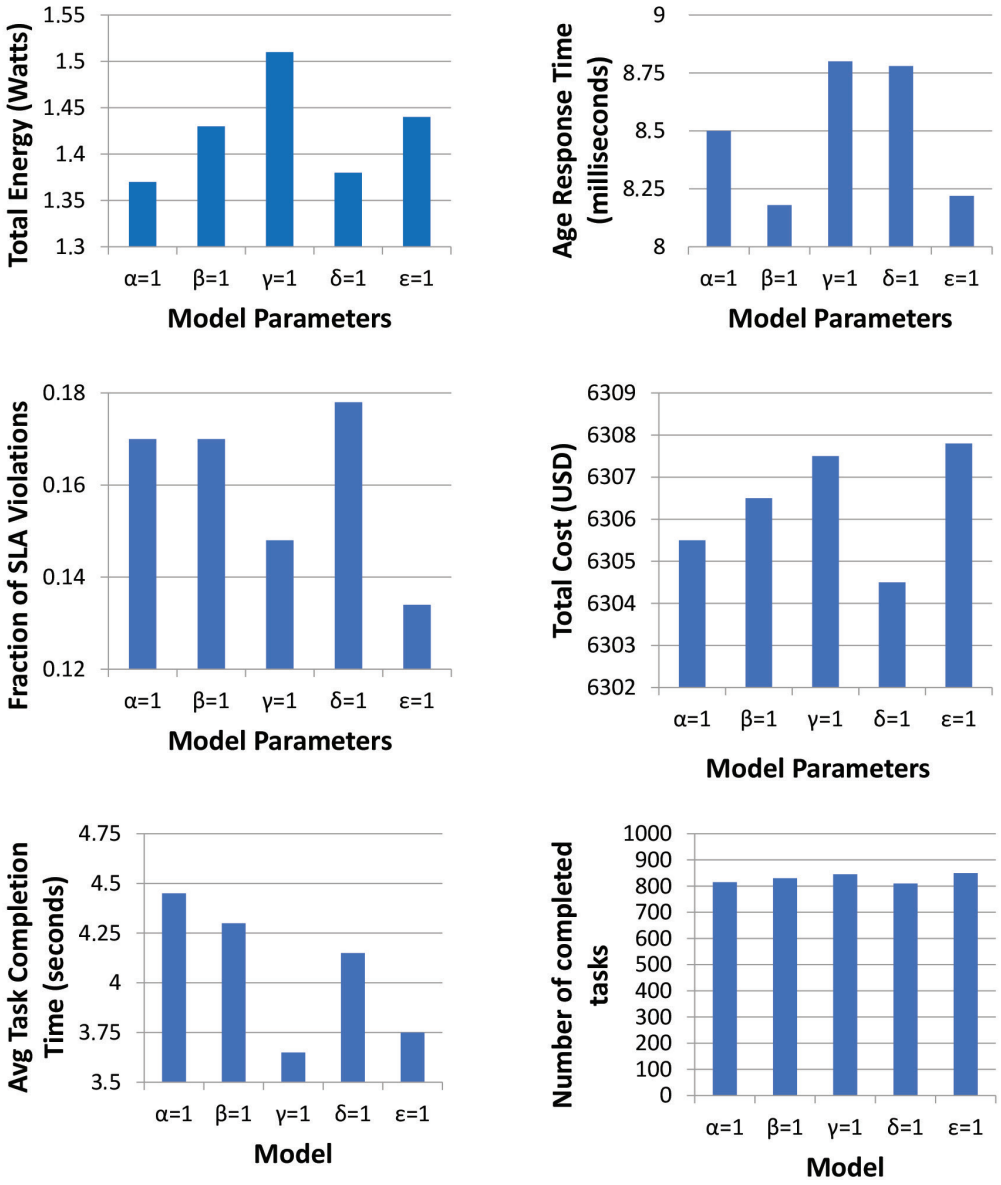

Fig. 4. Performance dynamics of proposed RLbDS algorithm with different model parameters associated with loss function

4.2.2. Performance Comparison with State of the Art

Our algorithm RLbDS is compared against several existing algorithms as presented in Fig. 5. Total energy consumption values are provided in 1*108 watts format. LR-MMT algorithm consumed 0.959, MAD-MC 0.95, DDQN 0.85, REINFORCE 0.82 and the proposed RLbDS consumed 0.73. The energy consumption of RLbDS is found to be the least among the scheduling algorithms. Average response time is another metric used for comparison. LR-MMT algorithm exhibited an average response time of 8.58 milliseconds, MAD-MC 8.4, DDQN 8.8, REINFORCE 8.35 and the proposed RLbDS required 7.7 milliseconds. The average response time of RLbDS is found to be the least among the scheduling algorithms. SLA violations are another important metric used for comparison. LR-MMT algorithm exhibited a fraction of SLA violations as 0.06, MAD-MC 0.13, DDQN 0.07, REINFORCE 0.06 and the proposed RLbDS exhibited 0.04. The fraction of SLA violations of RLbDS is found least among the scheduling algorithms.

Algorithm compared with the state-of-the-art

Total cost in terms of USD is another metric used for comparison. This metric is influenced by energy consumption. LR-MMT algorithm needs 6325 USD, MAD-MC 6325, DDQN 6325, REINFORCE 6300 and the proposed RLbDS needed 6000 USD. The total cost of RLbDS is found least among the scheduling algorithms. Concerning average task completion time, the LR-MMT algorithm needs 4.5 seconds, MAD-MC 4.3, DDQN 4, REINFORCE 3.8 and the proposed RLbDS requires 3.3 seconds. The average task completion time of RLbDS is found to be the least among the scheduling algorithms (average task completion time is given in 1*106 seconds format). The number of completed tasks is another observation made in our empirical study.

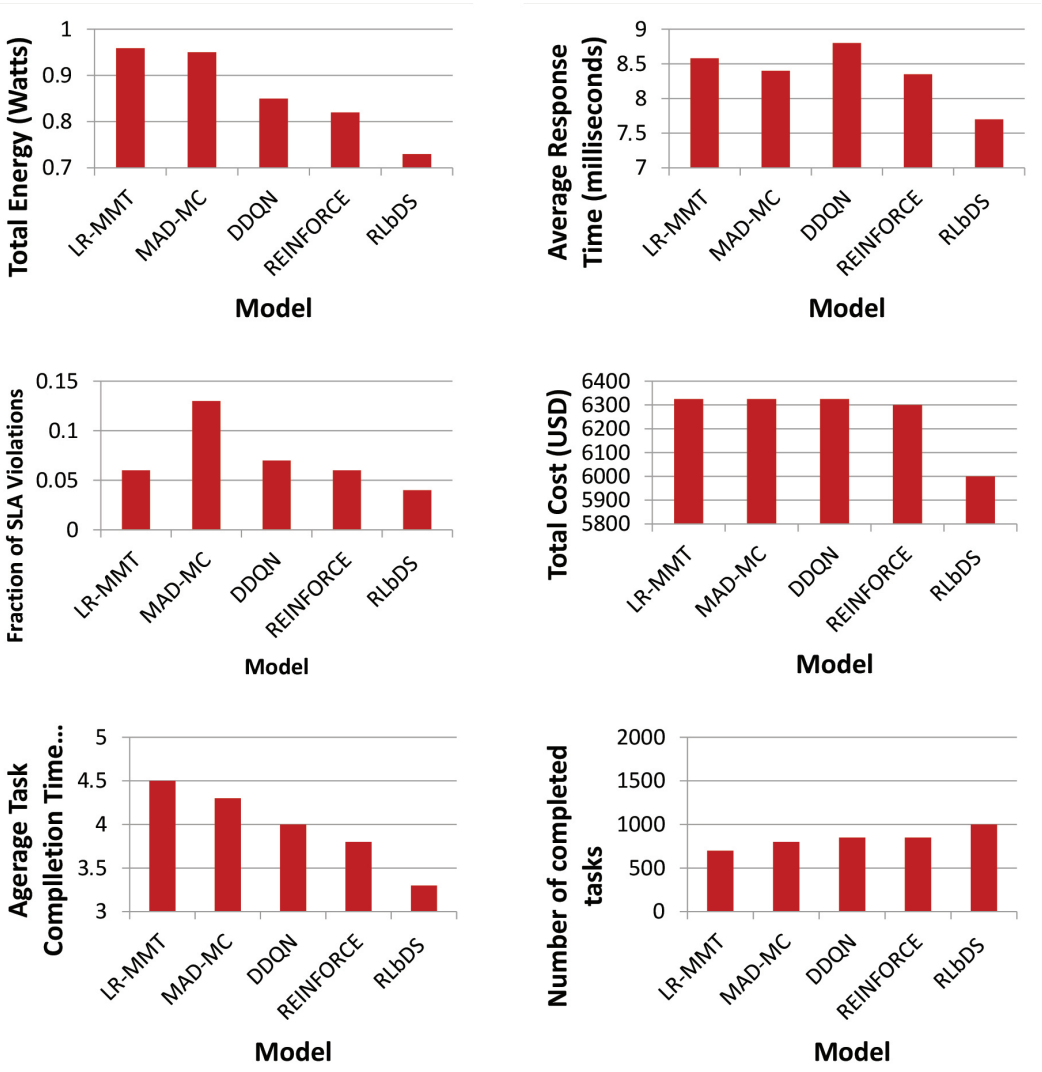

Fig. 5. Performance of proposed RLbDS algorithm compared with the state of the art

LR-MMT completed 700 tasks, MAD-MC 800, DDQN 850, REINFORCE 850 and the proposed completed 1000 tasks. The average task completion time of RLbDS is found to be the least among the scheduling algorithms.

4.2.3. Performance with Number of Recurrent Layers

Considering optimal values for hyperparameters scheduling overhead and loss dynamics against the number of recurrent layers are analysed. Overhead is computed as the ratio between the total duration of execution and the time taken for scheduling. Empirical study has revealed that the number of recurrent layers in the proposed architecture (Fig. 3) influences the loss and overhead.

As presented in Table 6, loss value and scheduling overhead against several recurrent layers are observed. Loss value and scheduling overhead are analysed against several recurrent layers as presented in Fig. 6.

Table 6. Performance against the number of recurrent layers

Number of recurrent layers	Performance	
	Loss value	Scheduling overhead (%)
0	3.69	0.009
1	3.4	0.010
2	2.9	0.010
3	2.6	0.010
4	2.5	0.019
5	2.4	0.029

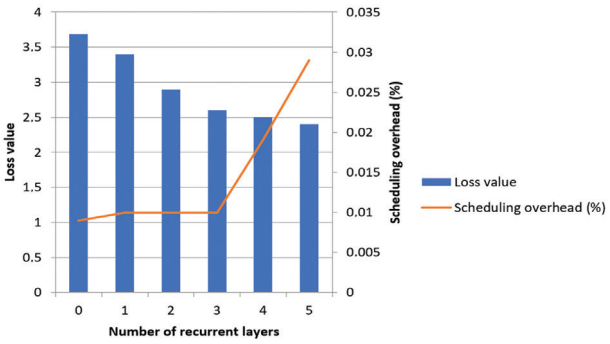

Fig. 6. Performance analysis with the number of recurrent layers

Several layers influence the loss value. Loss value decreases (performance increases) as the number of layers is increased. However, the scheduling overhead is increased with the number of recurrent layers.

4.2.4. Scalability Analysis

The scalability of the proposed algorithm is analysed in terms of speedup and efficiency. The analysis is made against the number of hosts. As presented in Table 7, the performance of the proposed algorithm in terms of its scalability is provided.

Table 7. Scalability analysis

Number of recurrent layers	Performance	
	Speed-up	Efficiency
1	1	1
5	5	0.8
10	9	0.785
15	13	0.775
20	17	0.765
25	19	0.725
30	21	0.7
35	23	0.650
40	25	0.630
45	26	0.570
50	27	0.525

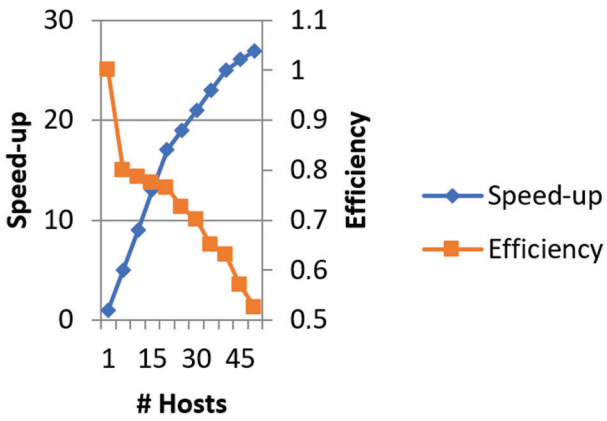

Fig.7. Scalability analysis in terms of speedup and efficiency

There is a trade-off observed between scalability and efficiency as presented in Figure 7. When the number of hosts is increased, there is a gradual decrease in efficiency while there is a gradual increase in speedup. From the experimental results, it is observed that the proposed RLbDS is found to be dynamic and can adapt to runtime situations as it is a learning-based approach.

Its asynchronous approach helps it in faster convergence. In the presence of dynamic workloads and device characteristics, RLbDS adapts to changes with ease.

5. CONCLUSION AND FUTURE WORK

We proposed a learning-based framework known as the Deep Reinforcement Learning Framework (DRLF).

This is designed in such a way that it exploits Deep Reinforcement Learning (DRL) with underlying mechanisms and enhanced deep network architecture based on Recurrent Neural Network (RNN). We also proposed an algorithm named Reinforcement Learning Dynamic Scheduling (RLbDS) which exploits different hyperparameters and DRL-based decision-making for efficient scheduling. Real-time traces of edge-cloud infrastructure are used for empirical study. We implemented our framework by defining new classes for CloudSim and iFogSim simulation frameworks. We evaluated the performance of the proposed algorithm named RLbDS by comparing it with state-of-the-art methods such as LR-MMT, MAD-MC, DDQN and REINFORCE. The results reveal the sensitivity dynamics hyperparameters, such as (α, β, γ, δ, ε), of the proposed RLbDS about model learning and its impact on different performance metrics. Our empirical study has revealed that RLbDS outperforms many existing scheduling methods. In future, we intend to improve our framework for container scheduling and load balancing.

6. REFERENCES

[1] A. Beloglazov, R. Buyya, "Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centres", Concurrency and Computation: Practice and Experience, Vol. 24, No. 13, 2012, pp. 1397–1420.

[2] M. Cheng, J. Li, S. Nazarian, "DRL-cloud: Deep reinforcement learning-based resource provisioning and task scheduling for cloud service providers", Proceedings of the 23rd Asia and South Pacific Design Automation Conference, Jeju, Korea, 22-25 January 2018, pp. 129-134.

[3] X.-Q. Pham, E.-N. Huh, "Towards task scheduling in a cloud-fog computing system", Proceedings of the 18th Asia-Pacific Network Operations and Management Symposium, Kanazawa, Japan, 5-7 October 2016, pp. 1–4.

[4] D.-M. Bui, Y. Yoon, E.-N. Huh, S. Jun, S. Lee, "Energy efficiency for a cloud computing system based on predictive optimization", Journal of Parallel and Distributed Computing, Vol. 102, 2017, pp. 103-114.

[5] L. Huang, S. Bi, Y. J. Zhang, "Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks", IEEE Transactions on Mobile Computing, Vol. 19, No. 11, 2020, pp. 2581-2593.

[6] H. Mao, M. Alizadeh, I. Menache, S. Kandula, "Resource management with deep reinforcement

learning", Proceedings of the 15th ACM Workshop on Hot Topics in Networks, Atlanta, GA, USA, 9-10 November 2016, pp. 50-56.

[7] D. Basu, X. Wang, Y. Hong, H. Chen, S. Bressan, "Learn-as-you-go with Megh: Efficient live migration of virtual machines", IEEE Transactions on Parallel and Distributed Systems, Vol. 30, No. 8, 2019, pp. 1786-1801.

[8] M. Xu, S. Alamro, T. Lan, S. Subramaniam, "Laser: A deep learning approach for speculative execution and replication of deadline-critical jobs in the cloud", Proceedings of the 26th International Conference on Computer Communication and Networks, Vancouver, BC, Canada, 31 July - 3 August 2017, pp. 1-8.

[9] Q. Zhang, M. Lin, L. T. Yang, Z. Chen, S. U. Khan, P. Li, "A double deep Q-learning model for energy-efficient edge scheduling", IEEE Transactions on Services Computing, Vol. 12, No. 5, 2019, pp. 739-749.

[10] F. Li, B. Hu, "Deepjs: Job scheduling based on deep reinforcement learning in the cloud data centre", Proceedings of the 4th International Conference on Big Data and Computing, Guangzhou, China, 10-12 May 2019, pp. 48-53.

[11] G. Rjoub, J. Bentahar, O. A. Wahab, A. S. Bataineh, "Deep and reinforcement learning for automated task scheduling in large-scale cloud computing systems", Concurrency and Computation: Practice and Experience, Vol. 33, No. 23, 2020, pp.1-14.

[12] O. Skarlat, M. Nardelli, S. Schulte, M. Borkowski, P. Leitner, "Optimized IoT service placement in the fog", Service Oriented Computing and Applications, Vol. 11, No. 4, 2017, pp. 427-443.

[13] X.-Q. Pham, N. D. Man, N. D. T. Tri, N. Q. Thai, E.-N. Huh, "A cost-and performance-effective approach for task scheduling based on collaboration between cloud and fog computing", International Journal of Distributed Sensor Networks, Vol. 13, No. 11, 2017, pp. 1-16.

[14] A. Brogi, S. Forti, "QoS-aware deployment of IoT applications through the fog", IEEE Internet of Things Journal, Vol. 4, No. 5, 2017, pp. 1185-1192.

[15] T. Choudhari, M. Moh, T.-S. Moh, "Prioritized task scheduling in fog computing", Proceedings of the ACMSE Conference, New York, NY, USA, March 2018, pp. 22:1-22:8.

[16] Q. Zhang, M. Lin, L. T. Yang, Z. Chen, P. Li, "Energy-efficient scheduling for real-time systems based on deep learning model", IEEE Transactions on Sustainable Computing, Vol. 4, No. 1, 2017, pp. 132-141.

[17] Z. Xiong, Y. Zhang, D. Niyato, R. Deng, P. Wang, L.-C. Wang, "Deep reinforcement learning for mobile 5g and beyond Fundamentals, applications, and challenges", IEEE Vehicular Technology Magazine, Vol. 14, No. 2, 2019, pp. 44-52.

[18] J. Almutairi, M. Aldossary, "A novel approach for IoT tasks offloading in edge-cloud environments. Journal of Cloud Computing", Journal of Cloud Computing, Vol. 10, 2021, p. 28.

[19] S. Ding, L. Yang, J. Cao, W. Cai, M. Tan, Z. Wang, "Partitioning Stateful Data Stream Applications in Dynamic Edge Cloud Environments", IEEE Transactions on Services Computing, Vol. 15, No. 4, 2021, pp. 2368-2381.

[20] S. S. Murad, R. Badeel, N. S. A. Alsandi, Rafi, "Optimized Min-Min Task Scheduling Algorithm For Scientific Workflows In A Cloud Environment", Journal of Theoretical and Applied Information Technology, Vol. 100, No. 2, 2022, pp. 480-506.

[21] L. Bulej et al. "Managing latency in edge cloud environment", Journal of Systems and Software, Vol. 172, 2021, pp. 1-15.

[22] J. Almutairi, M. Aldossary, "Investigating and Modelling of Task Offloading Latency in Edge-Cloud Environment. Computers", Materials & Continua, Vol. 68, No. 3, 2021, pp. 1-18.

[23] R. Zhang, W. Shi, "Research on Workflow Task Scheduling Strategy in Edge Computer Environment", Journal of Physics: Conference Series, Vol. 1744, 2021, pp. 1-6.

[24] X. Zhao, G. Huang, L. Gao, M. Li, Q. Gao, "Low load DIDS task scheduling based on Q-learning in an edge computing environment", Journal of Network and Computer Applications, Vol. 188, 2021, pp. 1-12.

[25] Y. Zhang, B. Tang, J. Luo, J. Zhang, "Deadline-Aware Dynamic Task Scheduling in Edge-Cloud Collaborative Computing", Electronics, Vol. 11, 2022, pp. 1-24.

[26] A. Lakhan et al. "Delay Optimal Schemes for Internet of Things Applications in Heterogeneous Edge Cloud Computing Networks", Sensors, Vol. 22, pp. 1-30.

[27] M. Singh, S. Bhushan, "CS Optimized Task Scheduling for Cloud Data Management", International Journal of Engineering Trends and Technology, Vol. 70, No. 6, 2022, pp. 114-121.

[28] Y. Hao, M. Chen, H. Gharavi, Y. Zhang, K. Hwang, "Deep Reinforcement Learning for Edge Service Placement in Softwarized Industrial Cyber-Physical System", IEEE Transactions on Industrial Informatics, Vol. 17, No. 8, 2021, pp. 5552-5561.

[29] S. Tuli et al. "Dynamic Scheduling for Stochastic Edge-Cloud Computing Environments using A3C learning and Residual Recurrent Neural Networks", IEEE Transactions on Mobile Computing, Vol. 21, No. 3, 2022, pp. 1-15.

[30] Q. Zhang, L. Gui, S. Zhu, X. Lang, "Task Offloading and Resource Scheduling in Hybrid Edge-Cloud Networks", IEEE Access, Vol. 9, 2021, pp. 940-954.

[31] L. Roselli, C. Mariotti, P. Mezzanotte, F. Alimenti, G. Orecchini, M. Virili, N. Carvalho, "Review of the present technologies concurrently contributing to the implementation of the Internet of things (IoT) paradigm: RFID, green electronics, WPT and energy harvesting", Proceedings of the Topical Conference on Wireless Sensors and Sensor Networks, San Diego, CA, USA, 25-28 January 2015, pp. 1-3.

[32] S. Tuli, N. Basumatary, S. S. Gill, M. Kahani, R. C. Arya, G. S. Wander, R. Buyya, "Healthfog: An ensemble deep learning based smart healthcare system for automatic diagnosis of heart diseases in integrated IoT and fog computing environments", Future Generation Computer Systems, Vol. 104, 2020, pp. 187-200.

[33] S. Sarkar, S. Misra, "Theoretical modelling of fog computing: a green computing paradigm to support IoT applications", IET Networks, Vol. 5, No. 2, 2016, pp. 23-29.

[34] Z. Abbas, W. Yoon, "A survey on energy conserving mechanisms for the Internet of things: Wireless networking aspects", Sensors, Vol. 15, No. 10, 2015, pp. 24818-24847.

[35] P. Kamalinejad, C. Mahapatra, Z. Sheng, S. Mirabbasi, V. C. Leung, Y. L. Guan, "Wireless energy harvesting for the Internet of things", IEEE Communications Magazine, Vol. 53, No. 6, 2015, pp. 102-108.

[36] A. M. Rahmani, T. N. Gia, B. Negash, A. Anzanpour, I. Azimi, M. Jiang, P. Liljeberg, "Exploiting smart e-Health gateways at the edge of healthcare Internet-of-Things: A fog computing approach", Future Generation Computer Systems, Vol. 78, 2018, pp. 641-658.

[37] J. Achiam, D. Held, A. Tamar, and P. Abbeel, "Constrained policy optimization", Proceedings of the 34th International Conference on Machine Learning, Sydney, NSW, Australia, 6-11 August 2017, pp. 22-31.

[38] R. Doshi, K.-W. Hung, L. Liang, K.-H. Chiu, "Deep learning neural networks optimization using hardware cost penalty", Proceedings of the IEEE International Symposium on Circuits and Systems, Montreal, QC, Canada, 22-25 May 2016, pp. 1954-1957

[39] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, R. Buyya, "Cloudsim: a toolkit for modelling and simulation of cloud computing environments and evaluation of resource provisioning algorithms", Software: Practice and Experience, Vol. 41, No. 1, 2011, pp. 23-50.

[40] H. Gupta, A. Vahid Dastjerdi, S. K. Ghosh, R. Buyya, "ifogsim: A toolkit for modelling and simulation of resource management techniques in the internet of things, edge and fog computing environments", Software: Practice and Experience, Vol. 47, No. 9, 2017, pp. 1275-1296.

[41] Bhagyalakshmi Magotra. (2023). "Adaptive Computational Solutions to Energy Efficiency in Cloud Computing Environment Using VM Consolidation". Archives of Computational Methods in Engineering. (2022), pp.1790-1818

[42] S. Shen, V. van Beek, A. Iosup, "Statistical characterization of business-critical workloads hosted in cloud datacenters", Proceedings of the 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, Shenzhen, China, 4-7 May 2015, pp. 465-474.

[43] Bitbrain Dataset, http://gwa.ewi.tudelft.nl/datasets/gwa-t-12-bitbrains (accessed: 2024)

[44] Q. Zhang, M. Lin, L. T. Yang, Z. Chen, P. Li, "Energy efficient scheduling for real-time systems based on the deep q-learning model", IEEE Transactions on Sustainable Computing, Vol. 4, No. 1, 2017, pp. 132-141.

[45] Microsoft Azure Pricing Calculator, https://azure.microsoft.com/en-au/pricing/calculator/ (accessed: 2024)