International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

Residual Recurrent Neural Network (R2N2) and Intelligent Resource Optimization based Dynamic Scheduling for Edge-Cloud Computing Environments

S. Supriya[*1], K. Dhanalakshmi[2]

Submitted: 09/10/2023 **Revised**: 28/11/2023 **Accepted**: 09/12/2023

Abstract: Internet-of-Things (IoT) based applications has resulted use of Fog computing paradigm, which permit effortlessly exploiting both mobile-edge and cloud resources. Applications are hard to schedule due to restricted resource capabilities, IoT mobility factors, heterogeneity of resources, networking hierarchy, stochastic behaviors. Task arrival rates, task durations, resource needs are unpredictable in the edge-cloud ecosystem, making scheduling and resource monitoring problematic. In order to reduce parameters like Average Energy Consumption (AEC), Average Response Time (ART), Average Migration Time (AMT), Service Level Agreement Violations (SLAV), Chaotic Crossover Tuna Swarm Optimizer (CCTSO) presented in this work. CCTSO algorithm has also optimized application settings of hyper-parameters based on various user requirements. These requirements, together with information from the Resource Monitoring Service about the computer characteristics, are used by Residual Recurrent Neural Network (R2N2) model to predict the next scheduling options. R2N2 is known to update model parameters fast, whereas Asynchronous-Advantage-Actor-Critic (A3C) adaptation is recognized to dynamic conditions swiftly with less input. For stochastic Edge-Cloud contexts, A3C learning-based real-time scheduler that enables concurrent decentralized learning among many agents. When compared to other existing approaches, trials done on real-world data sets indicate substantial gains in terms of energy usage, reaction time, SLA, operation cost.

Index Terms: *Asynchronous-Advantage-Actor-Critic (A3C), Cloud Technology, Chaotic Crossover Tuna Swarm Optimizer (CCTSO), Deep Reinforcement Learning, Edge Computing, Residual Recurrent Neural Network (R2N2) and Task Scheduling.*

1. Introduction

The Internet of Things (IoT) has made strides, which has led to a tremendous quantity of data being generated at an enormous velocity. Applications that use this data to analyze and take actions in accordance with defined goals need a sufficient computational infrastructure to meet user needs [1]. Many time-critical applications, like those used in healthcare, emergency response, and traffic monitoring, find it challenging to implement cloud-centric IoT apps because of rising network latency. Traditional cloud computing has been created to assist IoT (Internet of Things) successfully offload its responsibilities since consumer resources like storage capacity and signal resolution are limited [2-3]. On the other side, outsourcing localized tasks to a distant cloud center can result in significant delays, especially if a lot of task data needs to be exchanged across local computers and cloud centers. Almost all programs and scenarios that depend on latency find this sort of delay intolerable. Additionally, sending a lot of data through a network connected to a central server might seriously congest the network. To meet growing resource demands and increasing QoS (Quality of Service) requirements, new technologies are needed [4].

ECC (Edge Cloud Computing) paradigm is promising paradigm which gives lower latency response for IoT applications [5-7]. Since network edges have limitations in their processing capabilities and do not respond immediately in time-sensitive applications, ECC has been proposed [8]. ECC is an innovative component that is implemented among the on-premises level and the computing environment cloud layer in order to enhance service quality [9]. Unlike cloud computing, which requires data to be transferred across the core network, ECC places server hardware closest to the local side. ECC has an opportunity to prevent the undesirable delay caused by networks while offering high downstream bandwidth, which can lower the system latency [10]. As a result, ECC has garnered growing academic and industrial research attention [11]. The past several years have seen a lot of research on outsourcing computation in an ECC environment [12].

However, due to various circumstances, it is quite challenging to plan the computational paradigm of the Edge program. Computer servers adjust to the heterogeneity between faraway clouds and nearby edge nodes in terms of capacity, speed, reaction time, and energy usage. In addition, many types of computers may be present between the cloud and peripheral levels. Furthermore, the mobility element of the Edge paradigm

[1] *Research Scholar, Department of Computer Science, Kongunadu Arts and Science College, Coimbatore.*
[2] *HOD, Department of Information Technology, Kongunadu Arts and Science College, Coimbatore. Email: kdhanalakshmimca@gmail.com*
** Corresponding Author Email: supriyasundaram@gmail.com*

causes the bandwidth to fluctuate continually between data sources and processing nodes, demanding ongoing dynamic tweaking to meet application requirements. The scheduling problem is made more complex by the ECC unpredictable task arrival rate, work duration, and resource needs. By efficiently leveraging multi-layer resources, dynamic task scheduling becomes necessary to reduce energy consumption while simultaneously raising application's quality of service under uncertain circumstances.

Heuristic computations, meta-heuristic techniques that employ biological motivation and swarm intellect, and hybrid task-scheduling algorithms are the several types of task-scheduling methods used in cloud centers [13]. Several performance metrics such as system utilization, execution time, load balancing, network communication cost, and delay have been used in the scheduling process [14]. The optimum answer can be found using the heuristic work scheduling technique. However, it cannot always be relied upon to produce the optimum outcomes and is prone to partial selection. The meta-heuristic approach is an enhanced version of a heuristic algorithm, which combines local search and random algorithms [15–16]. It manages a lot of search space knowledge and allows for the investigation and expansion of the search space. Additionally, it has the ability to find approximations of optimal solutions by using learning procedures to learn and master new information.

Heuristics models are failed to include the ever changing conditions resulting from the requirements and the emergence of Edge-Cloud computing paradigm. In addition, they have a hard time adjusting to continuous system changes, which are common in Edge-Cloud environments. To do this, dynamic optimization of the system may benefit from a scheduling approach based on Reinforcement Learning (RL) [17]. Since the models are created using actual data, they are more precise and can recognize complex interactions between several interdependent components. Resource Management Systems (RMS) in remote systems has recently been optimized using a variety of value-based RL approaches [18]. The states of ECC environments which represent predicted cumulative rewards in RL setup, these solutions either employ neural networks or store Q-value functions in tables. Furthermore, no prior study has utilised temporal patterns in workload, network, or node activity to enhance scheduling decisions. These works also employ a centrally administered scheduling policy which is inappropriate for hierarchical or decentralized systems. The scheduling issue in stochastic edge-cloud systems are mapped and addressed using asynchronous policy gradient approaches. These methods may continually adjust to the dynamics of the system to provide better outcomes by employing recurrent neural networks to recognize behavioural patterns.

In order to reduce indicators like AEC, ART, AMT, Cost (C), and SLAV, the CCTSO (Chaotic Crossover Tuna Swarm Optimizer) has been presented in this paper. Contrary to traditional DQN (Deep Q-Network) systems, the suggested solution may swiftly modify the allocation strategy in accordance with dynamic workloads, host behaviour, and QoS requirements. This research also shows how to plan in a hybrid Edge-Cloud scenario using an R2N2-based approach that takes temporal trends into account.

2. Literature Review

Liu et al. [19], a QoS-guaranteed edge user data deployment strategy is introduced to increase the reduction in service latency and reduces the overall system cost (SC) with available resources. To compare the suggested technique with three alternative benchmark methods utilising real-world datasets, theoretical testing and performance analysis of the strategy are required. Experiments show how effective and efficient the proposed strategy than the existing methods.

In order to lessen the strain on server facilities and other centrally managed computing resources, Alamouti et al. [20] established a novel architectural method for cloud decentralized governance called the hybrid edge cloud (HEC). In using the resources of smart devices, HEC reduces communication latencies, frees up network capacity, and uses them. Modern network technologies like 5G and WiFi-6 are combined with benefits at HEC, which are used in public as well as private clouds to take advantage of computational power on smart devices and create a decentralized infrastructure that is scalable and resilient in the hyperconnected future.

To maximize resource use and reduce transferring disapproval in the edge-cloud computing system, Ullah et al. [21] improved the transfer of tasks under delay constraints. Deep reinforcement learning is used while taking into account the best choices for work offloading and resource allocation. Stochastic decision process is used to formulate this optimization issue, and DQN to identify the best task offloading strategy. To change the policy and offload the task as efficiently as possible in a flexible edge-cloud system while considering resource consumption, the DQNEC (DQN-edge-cloud) computational scheme was created. Simulations of DQNEC shows that it performs better than heuristic approaches in terms of optimising task offloads with low task rejection rates and resource utilisations at cheap costs.

To swiftly adapt to dynamic circumstances with minimal data, Tuli et al. [22] proposed the use of a R2N2 with A3C

learning. For stochastic ECC systems that provide concurrent decentralized learning across several agents, an A3C-based real-time scheduler is presented. In order to make effective scheduling decisions, the R2N2 architecture is introduced. In addition to temporal patterns, it may gather a variety of host and task data. According to the requirements of the application, the provided framework is adaptable and may change numerous hyper-parameters. Experiments on real datasets reveal that when compared to state-of-the-art algorithms, there are substantial increases in power consumption, response time, Service Level Agreement (SLA), and operational expenses of 14.40%, 7.74%, 31.90%, and 4.64%, respectively.

Deep learning as a tool for IoT was brought to the frontier computing environment by Li et al. [23]. A new offloading approach is developed to increase the effectiveness of IoT applications employing deep learning with edge computing because the processing power of the present edge nodes is constrained. Multiple deep learning tasks are performed in the performance evaluation using an offloading method in an edge computing environment. The suggested technique outplay other optimization strategies for IoT , according to evaluation findings.

LASER, a method using deep learning for the speculative performance and replication of time-sensitive tasks, was presented by Xu et al. [24]. A wide range of categorization and prediction issues have been effectively solved through machine learning. Compared to conventional machine learning methods, the DNN (Deep Neural Network) may provide more accurate regression (prediction) because between the input layer and the output layer, there are several layers of hidden units. LASER with SRQuant is the speculative CV approach based on quantitative analysis. They aim to reduce the cost of speculative execution, measured by the total (virtual) duration of tasks on the device, while increasing PoCD (probability of completion before deadlines), or the possibility that MapReduce tasks will be finished on time. The two approaches should be assessed and compared using conventional experiments.

Zhang et al. [25] implementing a double deep Q-learning model for edge planning that is energy-efficient. The proposed model specifically employs the target network to learn the parameters and the created framework to generate Q values for each DVFS (dynamic voltage and frequency scaling) technique. QDL-EES uses the ReLU (Rectified Linear Unit) function avoids gradient vanishing in the double-deep Q-learning model. The parameters of the proposed model are then trained using an algorithm for learning based on experience replay. On Edge CloudSim, the proposed model is contrasted with DQL-EES with regard to of energy savings and training time. Results show that proposed approach can improve training efficiency over QQL-EES and save an average of 2.00% to 2.40% of energy, demonstrating its promise towards energy-efficient edge scheduling.

In order to reduce energy costs for large-scale Cloud Service Provider (CSP), Cheng et al. [26] proposed a unique Deep Reinforcement Learning (DRL)-based allocation of resources and task scheduling system. These CSP have a very high number of servers and process vast amounts of user requests each day. The two-stage RP-TS processor with deep Q-learning seeks to automatically make long-term judgments optimal limit while training in changing contexts, such as customer demand trends and actual power costs. In order to achieve exceptionally high power efficiency, a low rejection rate, a short runtime, and speedy convergence, the proposed DRL-Cloud employs training methods such network targeting, test replay, exploration, and mining. The proposed DRL-Cloud outperforms one of the most advanced energy-efficient algorithms by up to 320% while maintaining an average reject rate that is lower. When contrasted to a rapid round-rob starting point, the proposed DRL-Cloud can save runtime by up to 144% for a CSP setup comprising 5,000 servers and 200,000 tasks.

For periodical tasks in real-time systems, Zhang et al. [27] presented DQL-EES (Deep Q-learning model- Energy-Efficient Scheduling) method. In particular, a deep Q-learning model is produced by combining the multi-layer autoencoder and the Q-learning technique. When applying the Q function inside the Q-learning deep learning model, a stacked autoencoder is employed to calculate the Q value for each system state. The training methodology also emphasizes the real-world replay scheme created to capture the essential components of the deep Q-learning model. In comparison to hybrid DVFS (Dynamic Pressure and Frequency Scaling) scheduling based on Q-learning (QL-HDS), the findings demonstrate that the suggested technique offers an average energy saving advantage of 4.20%.

In order to simplify the process of scheduling huge tasks upon cloud computing resources and decrease both usage of resources and task waiting time, four proficient and reinforced learning-based scheduling techniques were given by Rjoub et al. [28]. These techniques include DQN, DRL-LSTM (Deeper Reinforcement Learning Combined with LSTM), and RNN-LSTM (Recurrent Neural Network- Long Short-Term Memory). Reinforcement learning is the first method. DRL-LSTM outperforms the other three techniques in testing on the Google Cloud Platform. Additionally, it was demonstrated that the DRL-LSTM classifier reduced CPU overhead by 35.00% compared to the RR (Round Robin) and PSO (Particle Swarm Optimization) methods and by 67.00% compared to the SJF (Shortest Job First) technique. DRL-LSTM system reduces RAM (Random-Access Memory)

utilization costs by 31.25%, 65.00%, and 72.00% compared to the improved PSO, RR, and SJF.

First, Bui et al. [29] use the Gaussian procedure regression method to forecast how resources would be used in the forthcoming era. The convex optimization method is then used to determine the right number of actual hosts for every single monitoring window. To guarantee that just a handful of systems can still deliver an acceptable level of service, this amount of interest is determined. In order to fulfill the goal of energy savings, a similar migrating directive is then sent out to pile up virtual computers and shut down the idle physical servers. In order to test the effectiveness of the suggested tactic, experiments were carried out utilizing real workloads produced by the free source Montage toolkit and simulated data from 29 days of Google monitoring. Using the evaluation, it demonstrates that the proposed strategy may significantly reduce energy usage and preserve system performance. IoT, an emerging technology, makes it easy and advantageous to share data with additional devices across wireless networks. However, due of their continual development and technological advancements, IoT systems are more vulnerable to cyber attacks, which could result in strong assaults. Due to limited resource capabilities, IoT mobility factors, heterogeneity of resources, networking hierarchy, and stochastic behaviors, scheduling application tasks effectively in such situations is difficult. The stochastic nature of the edge-cloud ecosystem, which includes factors like task arrival rates, task durations, and resource requirements, makes scheduling and resource monitoring difficult. They are ineffective at using temporal demand patterns and are only suitable for centralized installations.

3. System Model and Problem Formulation

In this work, it is assumed that each of the peripheral and cloud nodes make up the underlying architecture. Fig. 1 depicts the system model in wide strokes. The network hierarchy diverse resources from edges to multi-hops distant clouds form environment in edge clouds. Computers host many application activities. There is significant variability in computing capability and response times between these servers. Since they are closer to the customers, edge devices offer substantially faster response times but have limited computational capacity due to resource constraints. The response time of cloud resources (Virtual Machine), on the other hand, is substantially higher when they are several hops distant from the users. However, cloud nodes have more resources and better computational power, allowing them to execute several tasks at once.

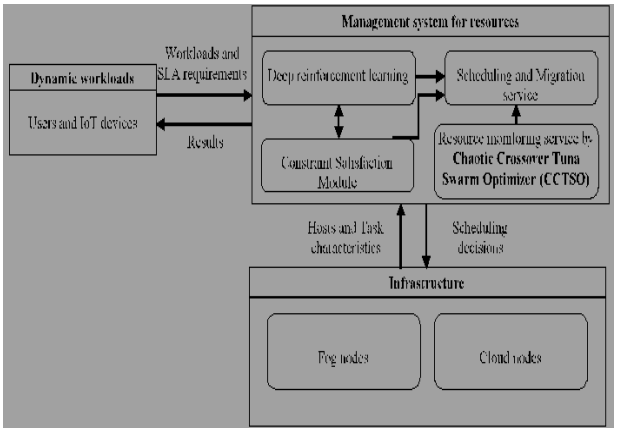

Fig. 1. System Model

Resource Management System (RMS) schedule, migrate, and monitor infrastructure. Along with their QoS and SLA requirements, IoT users and devices submit tasks to the RMS. It regularly decides whether to relocate ongoing work to new hosts and arranges new activities in line with the optimization objectives. The development expected completion dates or deadlines, as well as the CPU, RAM, bandwidth, and storage have an impact on the RMS decision. In order to simulate this effect, tasks are generated using the WGM (Workload Generation Module), a stochastic task generator.

DRLM (Deep Reinforcement Learning Module), interacts with the Scheduling and Migration services to offer locations for each task on the server. Run many alternative schedulers in DRLM with distinct tasks and node partitions. These schedulers can be applied to a single node or a number of edge cloud nodes. The computational load may be dispersed among many servers, as demonstrated by earlier research, enabling numerous agents to learn parameter changes concurrently and enabling quicker learning within the limitations of various devices. resources are scarce [30]. In order to update each host model separately, it is assumed that all edge and cloud nodes will add and synchronize regional gradients to their schedulers. A policy learning model in the DRLM gives each planner a distinct instance of the global neural network, enabling asynchronous updates. Another crucial component of RMS is the CSM (Constraint Satisfaction Module), which assesses limitations such whether a task is being moved or if the destination server has enough capacity. nay, to assess the suitability of the DRLM idea.

Workload Model: Every task has a changing workload, and tasks are generated randomly. As done in other studies [8], [30], subdivide the performance duration into periodic intervals of equal length. According to the chronological sequence of occurrence, the planned intervals are numbered as illustrated in Fig. 2.

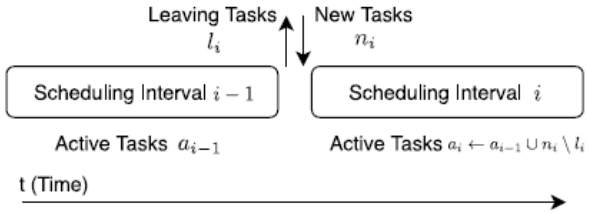

Fig. 2. Dynamic Task Workload Model

The i[th] scheduling interval, denoted as SI_i, begins at time t_i and lasts until the start of the subsequent interval, or t_{i+1}[th] subscript were active in each $SI_i$and designated as a_i. Further, in initial phases of SI_i, sets of tasks completed denoted by$l_i$and new tasks are sent by WGM are denoted as n_i.New tasks (n_i) are inserted into systems as tasks (l_i) are removed. As a result, the active tasks at the very start of the time interval SI_i are $a_i$is $a_{i-1} \cup n_i \setminus l_i$.

A metric reflecting Loss that is determined for each scheduling period is used to measure the scheduler's performance. The scheduler performs better the smaller the loss value. Denote loss of the interval SI_ias $Loss_i$.$H_i$is denoted as the i[th] host in an enumeration of Hosts.

Problem Formulation: The metric referred to as Loss provided for every scheduling interval serves as a measure of the scheduler effectiveness. The lower the loss value has given better scheduling. Declare the intervalSI_ias $Loss_i$. In an array of hosts, H is designated as the i[th] host. The group of hosts is referred to as Hosts in the edge-cloud environment, and its enumeration is $[H_0, H_1, \dots, H_n]$. Assume that there are n hosts in total at any given time during execution. Add the symbol T to the host that is allocated to a task. Define a scheduling tool as a mapping from a system state to an action that includes choosing a host for fresh assignments and deciding whether to migrate existing tasks. The state of the system at the beginning of SI_i, denoted as State$_i$ which consists of the parameter values of Hosts, remaining active tasks of the previous interval which ($a_{i-1} \setminus l_i$) and new tasks (n_i).

Schedulers must choose hosts to allocate or migrate which denotes $Action_i$for SI_i, for each task ina_i ($= a_{i-1} \cup n_i \setminus l_i$). Let the migratable tasks be $m_i \subseteq a_{i-1} \cup l_i$. As a result, $Action_i = \{h \in$ Hosts for task$T | T \in m_i \cup n_i\}$, which is a decision on migration for activities in m_i and allocation for activities in n_i. As a result, the function scheduler, denoted as Model, is $State_i \rightarrow Action_i$the model allocates tasks to hosts, with n being the number of servers in the Edge-Cloud data center, and determines how tasks are assigned to hosts over an interval. Thus, Equation (1) may be used to explain the issue for an ideal model.

$$\sum_i Loss_i \ \ subject\ to\ \forall i, Action_i = Model(State_i)\ \forall i\ \forall T \in m_i \cup n_i, \quad \{T\} \leftarrow Action_i(T) \tag{1}$$

4. Reinforcement Learning Model

Reinforcement Learning models are introduced for handling issues described in Section 3 as they are suitable for policy gradient learning.

Input Specification: State$_i$, which is made up of host factors including CPU, RAM, bandwidth, and disk utilization and capacity, is the input for the scheduler Model [16]. The host Million Instructions Per Second (MIPS), response time, cost per unit time, power features, cost per hour, cost per minute, and number of tasks assigned to this host are also listed. The computational (CPU), memory (RAM), and I/O (disk and bandwidth) capabilities of different hosts would vary. Such factors are essential for scheduling decisions because tasks in an edge-cloud environment compute, memory, and I/O limits. Additionally, by hiding the hosts that have no tasks, allowing several tasks to be assigned to a compact group of hosts could guarantee low energy use. Faster disk read/write rates on the host could enable the completion of I/O-intensive operations and avoid SLA violations. In the feature vector referred to asFV_i^{Hosts}, each of these parameters is specified for every host. The assignments in a_i are divided into: $n_i$and $a_{i-1} \setminus l_i$, two separate groups. The first set of parameters includes the task CPU, RAM, bandwidth, and storage space needs.

Output Specification: Depending on the input State$_i$, proposed model needs to assign hosts for tasks in a_i in the beginning of interval SI_i, and results referred $Action_i$ include host assignments for new tasks n_i and migration decisions for tasks from previous periods that are still in progress $\in a_{i-1} \setminus l_i$.Each task is transferred must be migratable to the new server asm_i which is $\subseteq a_i$ according to the feasibility criteria. Additionally, whenever a host h is assigned to a particular task T, h should not become overloaded after allocation, i.e., h is appropriate for T. As a result, $Action_i$ by Equation (2) in such a way that it applies to the intervalSI_i, $\forall T \in n_i \cup m_i, \{T\} \leftarrow Action_i(T)$,

$$= \{h \in Hosts \forall t \in n_i\ h_{new} \in Hosts \forall t \in m_i\ if\ t\ is\ to\ be\ migrated \tag{2}$$

appropriate for $t\ \forall\ t \in n_i \cup m_i$is subject to Action$_i$. Neural networks may produce host-task allocation preferences. This indicates that the model offers an ordered list of hosts instead of one for every task. Additionally, a

penalty is applied to uncontrolled behavior. Specifically, this captures two features of punishment: (1) the percentage of tasks that the simulation tried to transfer but was unable to do so is known as the migration penalty; and (2) the number of hosts who were granted greater priority but were unable to complete a task is added for each task to determine the host allocation penalty.

5. Proposed Methodology

For reducing parameters such as AEC, ART, AMT, and SLAV, the CCTSO has been developed. CCTSO algorithm has also optimized multiple values of hyper-parameters based on various user needs and application settings. These requirements, along with the features of the Resource Monitoring Service server, are used by R2N2 to forecast future scheduling decisions. R2N2 is known to update model parameters fast, but A3C adaptation is recognized to adapt swiftly to dynamic conditions with less data. An A3C-based real-time scheduler is proposed that supports concurrent decentralised learning among several agents in the stochastic Edge-Cloud scenario.

AEC: The term "AEC" refers to the power consumption of hardware and software, which includes all Edge and Cloud servers, normalised to the environment's maximum carrying capacity for any given time period. However, the energy sources used by periphery and cloud nodes could vary with the energy-saving devices for the edge and a primary power source for the cloud. Add a factor $\alpha_h \in [0,1]$ to the energy used by a host, that is possible to be adjusted for peripheral and cloud nodes based on user requirements and deployment strategy. Following is how the power is normalized:

$$= \frac{\sum_{h\in Hosts}\ \alpha_h \int_{t=t_1}^{t_{t+1}}\ P_h(t)dt}{\sum_{h\in Hosts}\ \alpha_h P_h^{max}(t_{i+1}-t_i)} \quad (3)$$

where the power function of host h with respect to time is denoted by $P_h(t)$, while the maximum power of h is indicated by P_h^{max}.

ART : The largest response time up to the current interval, normalized by the average reaction time for all tasks l_{i+1} in an interval SI_i, is the average reaction time. Task response time is the product of task execution time and the task planned host response time. ART is described,

$$ART_i = \frac{\sum_{t\in l_{i+1}}\ Response\ Time\ (t)}{|l_{i+1}|Response\ Time\ (t)} \quad (4)$$

AMT: AMT for all active tasks (a_i) during an interval SI_i is the mean migration time for every task, normalized by the longest possible migration time up to the preceding interval. The definition of AMT is described,

$$= \frac{\sum_{t\in a_i}\ Migration\ Time(t)}{|a_i|Response\ Time\ (t)} \quad (5)$$

Average SLA Violations (SLAV): Average SLA Violations (SLAV) is the mean number of SLA violations for departing task (li+1) during an interval SI_i.SLA (t) of task T is the combination made up of two metrics such as (i) performance decrease as a result of migrations and (ii) breach of SLA time per active host. Thus,

$$= \frac{\sum_{t\in l_{i+1}}\ SLA(t)}{|l_{i+1}|} \quad (6)$$

The oceanic predatory fish known as tuna goes by the scientific name Tunnini. It has been used to reduce host characteristics based on RMS. Spiral foraging is the first tactic. When feeding, tuna swim in a spiral pattern to minimize host features and move towards shallower waters where they may be more readily attacked by RMS. The next tactic is called parabolic foraging. As they follow the last host, each tuna forms a parabolic form to envelop it.

Initialization: Initial host populations are uniformly generated at random by CCTSO in the search space to begin the optimization process [31],

$$X_i^{int} = rand \cdot (u_b - l_b) + l_b, i = 1,2,\dots,NP \quad (7)$$

where NP is a count of tuna communities by host characteristics, $X_i^{int}$is the ith baseline individual, $u_b$and $l_b$are is the search space is defined by its lower and higher boundaries. Additionally, the vector "rand" follows a uniform distribution with values ranging from 0 to 1.

Spiral Foraging: Tuna exhibit a behavior in which they actively chase preceding individuals, hence facilitating the transfer of host-specific information among proximate tuna. The spiral foraging technique has the following mathematical formula [31],

$$X_i^{t+1} = \{\alpha_1 \cdot (X_{best}^t + \beta \cdot |X_{best}^t - X_i^t|) + \alpha_2 \cdot X_i^t, i \quad (8)$$
$$= 1\ \alpha_1 \cdot (X_{best}^t + \beta \cdot |X_{best}^t - X_i^t|) + \alpha_2 \cdot X_{i-1}^t, i = 2,3,\dots,NP$$

$$\alpha_1 = a + (1-a) \cdot \frac{t}{t_{max}} \quad (9)$$

$$\alpha_2 = (1-a) - (1-a) \cdot \frac{t}{t_{max}} \quad (10)$$

$$\beta = e^{bl} \cdot cos\ cos\ (2\pi b) \quad (11)$$

$$l = e^{3coscos\left(((t_{max}+1/t)-1)\pi\right)} \tag{12}$$

where $X_i^{t+1}$is the ith host characteristic of the ith iteration, $X_{best}^t$is the most recent optimal host characteristic individual (food), $\alpha_1$and $\alpha_2$are coefficients of mass which regulate the propensity of the host features to migrate regarding the optimal resources and the previous resource, and is an unchanged used to determine how closely the tuna will adheres to the most effective asset and the previous resource throughout the initial phase, t is the current iteration number, and all tuna have good ability to utilize the search space surrounding the meal when they forage spirally. In order to perform a spiral search, think about creating a random coordinate in the search space. This makes it easier for each person to search a larger area and equips CCTSO with the capacity for global exploration. The following describes a mathematical model [31],

$$X_i^{t+1} = \{\alpha_1 \cdot (X_{rand}^t + \beta \cdot |X_{rand}^t - X_i^t|) + \alpha_2 \cdot X_i^t, i \\ = 1\ \alpha_1 \cdot (X_{rand}^t + \beta \cdot |X_{rand}^t - X_i^t|) + \alpha_2 \cdot X_{i-1}^t, i = 2,3, \dots, NP \tag{13}$$

where$X_{rand}^t$is a resource monitoring service reference point that was generated at random. As the number of iterations rises, from randomized people to optimum individuals, TSO changes the spiral foraging reference points. Therefore, the spiral foraging strategy ultimate mathematical model is: [31],

$$X_i^{t+1} = \{\alpha_1 \cdot (X_{rand}^t + \beta \cdot |X_{rand}^t - X_i^t|) + \alpha_2 \cdot X_i^t, i = 1, \alpha_1 \cdot (X_{rand}^t + \beta \cdot |X_{rand}^t - X_i^t|) + \alpha_2 \cdot X_{i-1}^t, i \\ = 2,3, \dots, NP, if\ rand < \frac{t}{t_{max}}\ \ \alpha_1 \cdot (X_{best}^t + \beta \cdot |X_{best}^t - X_i^t|) + \alpha_2 \cdot X_i^t, i \\ = 1, \alpha_1 \cdot (X_{best}^t + \beta \cdot |X_{best}^t - X_i^t|) + \alpha_2 \cdot X_{i-1}^t, i = 2,3, \dots, NP, if\ rand \geq \frac{t}{t_{max}} \tag{14}$$

Parabolic Foraging: A parabolic pattern, in addition to a spiral pattern, is produced by tunas working together to feed. Tuna uses food to form a parabolic shape. Tuna search for Resources Monitoring Services while scanning. The precise mathematical model specifics are listed below [31],

$$X_i^{t+1} = \{X_{best}^t + rand \cdot (X_{best}^t + X_i^t) + TF \cdot p^2 \cdot (X_{best}^t + X_i^t), if\ rand < 0.5\ TF \cdot p^2 \cdot X_i^t, if\ rand \geq 0.5 \tag{15}$$

$$p = \left(1 - \frac{t}{t_{max}}\right)^{\left(\frac{t}{t_{max}}\right)} \tag{16}$$

wherein TF is a random number with a value of 1 or –1.Although the hosts themselves are always the same, crossover with collected host traits from the host provides unique characteristics in order to solve the local optima problem. Equation (17) gives the CC operator,

$$c(x_i) = m * p_1(x_i) + (1 - m) * p_2(x) \tag{17}$$

where, $p_1(x_i)$ and $p_2(x_i)$ are the relative positions of parents 1 as well as parent 2 in dimensions (i) for host characteristics and $x_i$is the position of the newly formed child in dimensions (i), m is the random integer created by the chaotic the random number generator, and so on. Equations (18–19) provide the chaotic number generator mathematical expression,

$$map = 4 * z * (1 - z) \tag{18}$$

$$m = (0.5 * r) + (0.5 * map) \tag{19}$$

Equation (18) uses the created random numbers z and r from the compiler random number generator, while equation (19) uses the randomly generated arbitrary number m from the chaotic number generator. Together; tuna use the two foraging techniques of spiral and parabolic foraging to determine their host traits. The sample is first generated randomly in the asset monitoring service for the CCTSO optimization process. The parameter setting will cover the contents of parameter z. Until the end condition is met, all CCTSO persons are updated and computed continuously during the whole optimization process, at which point the most appropriate tracking service and the associated efficiency value are returned. In Algorithm 1, CCTSO pseudocode is displayed.

Algorithm 1: Pseudocode for CCTSO

Input: Population size (N) and highest iteration (t_{max})

Result: The best person in the prey position and its range of fitness

Begin

Initialize the arbitrary inhabitant of tuna swarms $X_i^{int}(i = 1, \dots, N)$

Set variables $a = 0.7$and $z = 0.05$;

while$(t < t_{max})$

Determine the fitness values (AEC, ART, AMT, C, and SLAV) of tuna swarms (which includes all edge and cloud hosts);

Update X_{best}^{t};

for$(all\ tuna\ swarms)$

Update $\alpha_1, \alpha_2$and p;

if$(rand < z)$

Update the location $X_i^{t+1}$by equation (8);

else if $(rand \geq z)$

if$\left(rand < \frac{1}{2}\right)$

if$\left(\frac{t}{t_{max}} < rand\right)$

Update the location $X_i^{t+1}$by equation (13)

else if $\left(\frac{t}{t_{max}} \geq rand\right)$

Update the location $X_i^{t+1}$by equation (8)

else if $\left(rand \geq \frac{1}{2}\right)$

Update the location $X_i^{t+1}$by equation (15)

Update the new CC operator by equation (17)

end if

end if

end if

end for

$t = t + 1$;

end while

Return the best individual X_{best} (AEC, ART, AMT, C, and SLAV) and the best fitness value $F(X_{best})$

End

6. Stochastic Dynamic Scheduling using Policy Gradient Learning

The entire system operates as follows: (1) As each scheduling period begins, the RMS gets task requests and task criterion like computations, bandwidths, and SLAs. (2) To predict the next scheduling options, the DRL model makes use of these requirements as well as the host features from Resources Monitoring Service. (3) The output of the DRL model is used by the constraint fulfillment module to identify probable migration and scheduling options. (4) RMS will notify users or IoT devices to transmit their inquiries straight to the task device or cloud device for newly generated tasks. (5) The DRL model's parameters and loss function are changed at the same time. Modelling this function with Q-tables or neural networks leads to a strict deterministic approach in stochastic conditions. However, this method modifies the network and optimizes the policy gradient to approximation the policy by using $Loss_i^{PG}$ as input signal. Use the R2N2 network to approximation the parameters of the function from $State_i$ to $Action_i^{PG}$for each interval SI_i. The R2N2 network's capacity to detect intricate temporal correlations between inputs and outputs is one benefit. After this certain amount of time, a single network can forecast both the strategy (agent leader) and the total loss. Use an R2N2 model with stage states, losses, and penalties to iteratively preprocess data and choose the best schedule for each planning. Because of this, the model can quickly adjust to the requirements of the user, the environment, and the specific application.

The R2N2 network has three repeating levels with skip connections once the first two layers are completely linked. The dense layers are initially routed through a flattened, two-dimensional input. The two separate network heads receive the outcome of the final recurrent layer. The performer head output is moulded into a 2-dimensional, 100*100 vector with a size of 104. This indicates that a maximum of 100 assignments and 100 hosts can be managed by the model. The infrastructure must be adjusted in accordance with this so that an equitable comparison with a larger system may be made. To make sure that all values are in the range [0, 1] and that the total of all values in a row is 1, SoftMax is executed across the second dimension. This shows that the model can handle an aggregate of 100 tasks and 100 hosts. It is necessary to modify the infrastructure to correspond with this in order to compare it fairly to a bigger system. SoftMax is used over the second dimension to verify that all entries fall within the range of between 0 and 1 and that the sum of all numbers in a row is 1.

Use the notation $f_e$for the characteristic of element e and f as f_e and f_e for the lowest and highest values of the feature f respectively. Two scheduling strategies based on heuristics On the basis of a sample dataset, the minimum and maximum values are calculated using Local-Regression (LR) for task assignment and Maximum-Migration-Time (MMT) for task selection [32]. Then, using Equation (20), feature-wise standardization is carried out.

$$e = \{0\ if f_e = f_e \left(1, \left(0, \frac{e - f_e}{f_e - f_e}\right)\right), else \qquad (20)$$

The R2N2 model receives this pre-processed input and flattens it so that it can travel through the thick layers. By first producing a categorized list of hosts $SortedHosts_i$ by diminishing frequency in $O_i$for every i, the resultant generated O is transformed into$Action_i^{PG}$. To minimize total

loss, gradients contain a minus sign and are proportionate to this quantity. The Mean Squared Error (MSE) of the projected accumulated loss, which corresponds to the cumulative loss after a one-step look-ahead, is the second gradient component. CSM translates the output $Action_i^{PG}$ PG to $Action_i$ and delivers it to the RMS at each scheduling interval. As a consequence, at each interval, the R2N2 network does a forward pass. Iterative pre-processing and input of the window state, losses and penalties into the R2N2 model yields the optimal scheduling strategy for each scheduling window. This enables the model to respond swiftly to particular user, environment, and application needs.

7. Results and Discussion

This section offers a detailed analysis of the findings comparing the model to a number of industry standard methodologies. Also, describes experimental setup, evaluation metrics, and data collection. CloudSim is used to enable edge node properties including reaction time, cost, power. For the constraint fulfillment module, new software as well as input and output preprocessing have been developed. The loss function is computed using CloudSim performance tracking and storage service. Tasks (cloudlets) are given to VM in the simulation setting, which is then given to hosts.

DATASET: Tasks from the cloud are sent from the VM to the servers in the virtualized environment. For the present setting of task on edge-cloud environment, regard as a bijection from cloudlets to VM by assigning i^{th} formed Cloudlet to i^{th} formed VM and discards the VM when the corresponding Cloudlet is finished.

Cloudlet dynamic workload is designed using the open-source Bitbrain collection of real-world data. Bitbrain dataset shows statistics on resource use for critical business workloads on its infrastructure [33]. Because they reflect actual infrastructure utilisation patterns and may be used to create trustworthy input feature vectors for machine learning models, more than 1,000 VM workload logs from two machine kinds were chosen. The dataset has workload information for each timestamp (5 minutes apart), including CPU cores, MIPS, RAM, and network (receive/transmit) and storage (read/write) bandwidth. It can be obtained from the BitBrain dataset at http://gwa.ewi.tudelft.nl/datasets/gwa-t-12-bitbrains. Distribute the information set into two portions with VM workloads of 25.00% and 75.00% each. The R2N2 network is trained using the bigger partition, whereas the smaller fraction is used for testing, sensitivity analysis, and comparing to other similar efforts. Microsoft Windows IaaS cloud service is the foundation of the pricing mechanism for the cloud layer.

Metrics: The following metrics has been used for results comparison.

Here is the average response time analysis,

$$ART = \frac{\sum_{t \in l_{i+1}} Response\ Time(t)}{|l_{i+1})} \tag{21}$$

The list of SLA Infractions is as follows,

$$SLAV = \frac{\sum_i SLAV_i . |l_{i+1}|}{\sum_i l_i} \tag{22}$$

Average Task Completion Time: It is determined by adding the average scheduling time of a task, the task's execution time, and the server's response time during the most recent scheduling interval. The overall number of tasks completed, the percentage of tasks finished within the anticipated execution time (based on the desired MIPS), the number of task migrations each time interval, and the total migration time transfer over the course of the period are all taken into account.

Results Comparison Methods: Many heuristics have been used to approach dynamic planning. Three of the top methods were chosen to solve the subproblems of server overload detection and task/VM selection. All of these variations locate the target host using the Best Fit Decreasing (BFD) algorithm. Additionally, contrast the outcomes with two popular RL techniques are frequently employed in the literature.

LR-MMT: For task selection and overloading detection, using the Local Regression (LR) and Minimum Migration Time (MMT) methods, dynamically schedules workloads.

MAD-MC: Dynamically arranges workloads based on the task selection and overload detection heuristics of Median Absolute Deviation (MAD) and Maximum Correlation Policy (MC), respectively.

DDQN: Many papers in the literature have employed the Deep Q-Learning is based RL approach.

REINFORCE: Fully integrated neural network using the REINFORCE approach based on policy gradients.

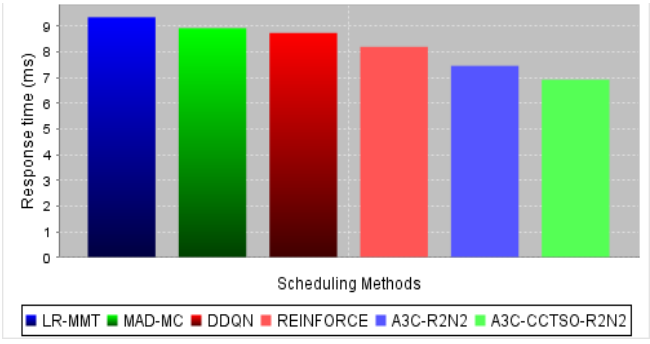

Fig. 3. Average Response Time vs. Scheduling Methods

According to Fig. 3, proposed A3C-CCTSO-R2N2

scheduling policy has the shortest average response time compared to the other scheduling policies. Proposed model explicitly asks if a particular node is a perimeter or cloud node and distributes tasks based on RMS (CCTSO) without requiring repeated migrations and embedding AMT in the loss function. This demonstrates that, in contrast to previous approaches like LR-MMT, MAD-MC, DDQN, REINFORCE, and A3C-R2N2, which have increasing ART of 9.35 ms, 8.92 ms, 8.74 ms, 8.20 ms, and 7.46 ms correspondingly, the proposed system has a lower ART of 6.92 ms (refer to Table 1).

Table 1. Average response time vs. scheduling methods

Scheduling Methods	***Response Time (ms)***
LR-MMT	9.35
MAD-MC	8.92
DDQN	8.74
REINFORCE	8.20
A3C-R2N2	7.46
A3C-CCTSO-R2N2	6.92

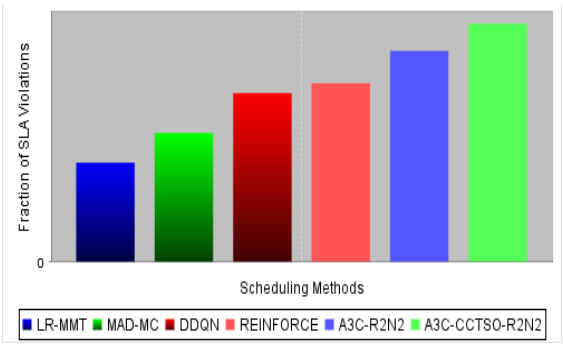

Fig. 4. Fraction of SLA Violations vs. Scheduling Methods

The A3C-CCTSO-R2N2 model, as depicted in Fig. 4, has fewer SLA breaches than the A3C-R2N2 policy. Again, this is attributable to fewer migrations and clever work scheduling to avoid the huge loss of value brought on by SLA violations. While other approaches like LR-MMT, MAD-MC, DDQN, REINFORCE, and A3C-R2N2 have resulted in higher ART of 0.093, 0.084, 0.072, 0.064, and 0.052 correspondingly (see Table 2), the proposed system has fewer SLA breaches of 0.040.

Table 2. Fraction of SLA violations vs. scheduling methods

Scheduling Methods	***Fraction of SLA Violations***
LR-MMT	0.093
MAD-MC	0.084
DDQN	0.072
REINFORCE	0.068
A3C-R2N2	0.052
A3C-CCTSO-R2N2	0.040

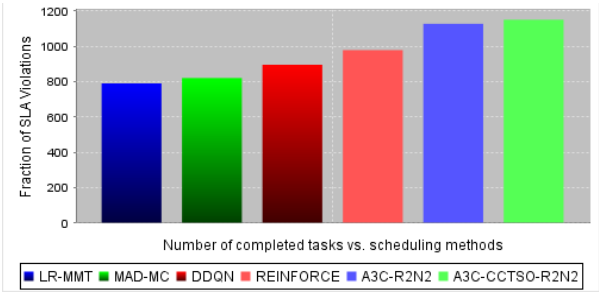

Fig. 5. Number of Completed Tasks vs. Scheduling Methods

The A3C-CCTSO-R2N2 model has a higher percentage of completed tasks, as shown in Fig. 5, and it can guarantee that tasks can be distributed to as few cloud VMs as feasible to minimize cost. The number of tasks completed by the proposed system is greater at 1150, compared to the numbers of tasks finished by the LR-MMT, MAD-MC, DDQN, REINFORCE, and A3C-R2N2 techniques, which are 790, 820, 895, 978, and 1127, respectively (see Table 3).

Table 3. Number of completed tasks vs. scheduling methods

Scheduling Methods	***Fraction of SLA Violations***
LR-MMT	790
MAD-MC	820
DDQN	895
REINFORCE	978
A3C-R2N2	1127
A3C-CCTSO-R2N2	1150

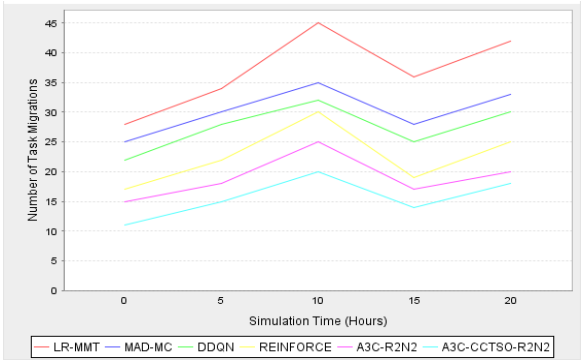

Fig. 6. Number of Task Migration in Each Interval vs. Scheduling Methods

The results of the total amount of assignment migrations with regard to simulation time are displayed in Fig. 6. Several techniques, including A3C-CCTSO-R2N2, LR-MMT, MAD-MC, DDQN, REINFORCE, and A3C-R2N2, are used to evaluate the outcomes. According to the results, the proposed

system only requires 18 task migrations, compared to 42, 33, 30, 25, and 20 for other approaches like LR-MMT, MAD-MC, DDQN, REINFORCE, and A3C-R2N2 during a 20-hour simulation (see Table 4).

Table 4. Number of task migration vs. scheduling methods

Scheduling Methods	***Simulation time (Hours)***				
	0	***5***	***10***	***15***	***20***
LR-MMT	28	34	45	36	42
MAD-Mc	25	30	35	28	33
DDQN	22	28	32	25	30
REINFORCE	17	22	30	19	25
A3C-R2N2	15	18	25	17	20
A3C-CCTSO-R2N2	11	15	20	14	18

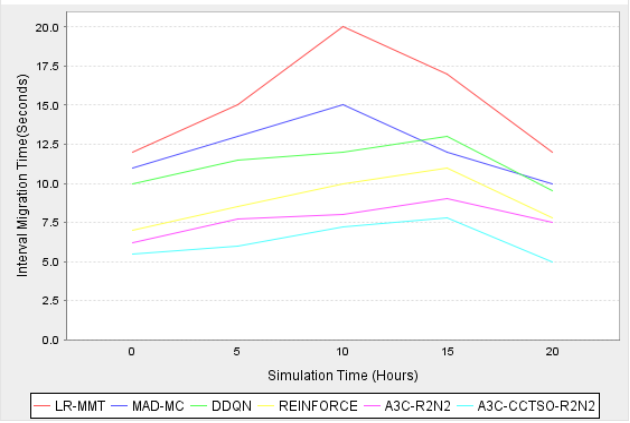

Fig. 7. Total Migration Time in Each Interval vs. Scheduling Methods

The findings of the task interval migration time series with regard to the simulation time are displayed in Figure 7. Several techniques, including A3C-CCTSO-R2N2, LR-MMT, MAD-MC, DDQN, REINFORCE, and A3C-R2N2, are used to evaluate the outcomes. It demonstrates that the proposed system has a shorter overall migration time of 5.0 seconds, while other approaches like LR-MMT, MAD-MC, DDQN, REINFORCE, and A3C-R2N2 have migration times of 12.00 minutes, ten seconds, 9.50 seconds, 7.80 seconds, and 7.50 seconds for simulated times of 20 hours, respectively (refer to Table 5). In Fig. 6 and 7, an A3C-CCTSO-R2N2 model may achieve optimum metric outcomes by limiting task migrations since migration length and quantity affect task response.

Table 5. Total migration time in each interval vs. scheduling methods

Scheduling Methods	***Simulation time (Hours)***				
	0	***5***	***10***	***15***	***20***
LR-MMT	12	15	20	17	12
MAD-Mc	11	13	15	12	10
DDQN	10	11.5	12	13	9.5
REINFORCE	7	8.5	10	11	7.8
A3C-R2N2	6.2	7.7	8	9	7.5
A3C-CCTSO-R2N2	5.5	6	7.2	7.8	5

8. Conclusion and Future Work

It is complex to effectively use edge and cloud resources in uncertain situations with changing workloads to enhance service quality and response times. This study introduces on-the-fly end-to-end task scheduling for hybrid devices and cloud computing platforms. For reducing metrics like AEC, ART, AMT, SLAV, CCTSO have been developed. CCTSO algorithm has also optimized multiple values of hyper-parameters based on various user needs and application settings. Together, tuna use the two foraging techniques of spiral and parabolic foraging to determine their host traits. Policy gradient based Reinforcement learning method (A3C) has been introduced for stochastic dynamic scheduling method. An A3C-R2N2-based scheduling algorithm considers all-important task and host factors to improve performance. The results of the experiment utilizing an iFogSim Toolkit upgraded to CloudSim 5.0 demonstrate the model superiority over other approaches and previously proposed RL models. The proposed method can reduce response time by 5.49% and SLA violation by 15.55% as compared to A3C-R2N2, according to extensive simulation studies utilizing iFogSim and CloudSim on practical Bitbrain dataset. Furthermore, the A3C clients in the edge-cloud configuration would need to have their CPU, RAM, storage, and bandwidth utilization tracked and synced. In addition of the scalability analysis, tests to determine the scalability of the proposed framework with regards to the quantity of hosts and tasks are also planned. The existing architecture has a limited capacity for scheduling nodes at the edges and tasks.

References

[1] R. Mahmud, S. N. Srirama, K. Ramamohanarao and R. Buyya, “Quality of Experience (QoE)-aware placement of applications in Fog computing environments,” *Journal of Parallel and Distributed Computing*, vol. 132, pp. 190-203, 2019.

[2] T. Wang, Y. Mei, W. Jia, X. Zheng, G. Wang and M. Xie, "Edge-based differential privacy computing for sensor–cloud systems," *Journal of Parallel and Distributed computing*, vol. 136, pp. 75-85, 2020.

[3] J. Feng, L. Zhao, J. Du, X. Chu and F. R. Yu, "Energy-efficient resource allocation in fog computing supported IoT with min-max fairness guarantees," *IEEE International Conference on Communications (ICC)*, 2018, pp. 1-6.

[4] R. Bi, Q. Liu, J. Ren and G. Tan, "Utility aware offloading for mobile-edge computing," *Tsinghua Science and Technology*, vol. 26, no. 2, pp. 239-250, 2020.

[5] J. Diechmann, K. Heineke, T. Reinbacher and D. Wee, "The Internet of Things: How to capture the value of IoT," Technical Report, 2018, pp. 1-124.

[6] S. Tuli, R. Mahmud, S. Tuli and R. Buyya, "FogBus: A Blockchain based Lightweight Framework for Edge and Fog Computing," *Journal of Systems and Software*, vol. 154, pp. 22 – 36, 2019.

[7] J. Wang, K. Liu, B. Li, T. Liu, R. Li and Z. Han, "Delay-sensitive multi-period computation offloading with reliability guarantees in fog networks," *IEEE Transactions on Mobile Computing*, vol. 19, no. 9, pp.2062-2075, 2019.

[8] D. Kim, J. Son, D. Seo, Y. Kim, H. Kim and J. T. Seo, "A novel transparent and auditable fog-assisted cloud storage with compensation mechanism," *Tsinghua Sci. Technol.*, vol. 25, no. 1, pp. 28–43, 2019.

[9] W. Shi, J. Cao, Q. Zhang, Y. Li and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, 2016.

[10] A. Al-Shuwaili and O. Simeone, "Energy-efficient resource allocation for mobile edge computing-based augmented reality applications," *IEEE Wirel. Commun. Lett.*, vol. 6, no. 3, pp. 398–401, 2017.

[11] L. U. Khan, I. Yaqoob, N. H. Tran, S. A. Kazmi, T. N. Dang and C. S. Hong, "Edge-computing-enabled smart cities: A comprehensive survey," *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 10200-10232, 2020.

[12] M. Merluzzi, P. Di Lorenzo, S. Barbarossa and V. Frascolla, "Dynamic computation offloading in multi-access edge computing via ultra-reliable and low-latency communications," *IEEE Trans. Signal Inf. Process. Netw.*, vol. 6, pp. 342–356, 2020.

[13] T. Jena and J. R. Mohanty, "Disaster recovery services in intercloud using genetic algorithm load balancer," *International Journal of Electrical and Computer Engineering*, vol. 6, no. 4, pp. 1828-1838, 2016.

[14] M. A. Elaziz, S. Xiong, K. P. N. Jayasena and L. Li, "Task scheduling in cloud computing based on hybrid moth search algorithm and differential evolution," *Knowledge-Based Systems*, vol. 169, pp. 39–52, 2019.

[15] Y. Xiong, S. Huang, M. Wu, J. She and K. Jiang, "A johnson'-rule- based genetic algorithm for two-stage-task scheduling problem in data-centers of cloud computing," *IEEE Transactions on Cloud Computing*, vol. 7, no. 3, pp. 597–610, 2017.

[16] M. Akbari, R. Hassan, and S. H. Alizadeh, "An enhanced genetic algorithm with new operators for task scheduling in heterogeneous computing systems," *Engineering Applications of Artificial Intelligence*, vol. 61, pp. 35–46, 2017.

[17] G. Fox, J. A. Glazier, J. C. S. Kadupitiya, V. Jadhao, M. Kim, J. Qiu, J. P. Sluka, E. Somogyi, M. Marathe, A. Adiga and J. Chen, "Learning everywhere: Pervasive machine learning for effective high-performance computation," *IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2019, pp. 422-429.

[18] D. Basu, X. Wang, Y. Hong, H. Chen and S. Bressan, "Learn-asyou-go with MEGH: Efficient live migration of virtual machines," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 8, pp. 1786–1801, 2019.

[19] B. Liu, S. Meng, X. Jiang, X. Xu, L. Qi and W. Dou, "A QoS-guaranteed online user data deployment method in edge cloud computing environment," *Journal of Systems Architecture*, vol. 118, pp. 1-11, 2021.

[20] S. M. Alamouti, F. Arjomandi and M. Burger, "Hybrid edge cloud: A pragmatic approach for decentralized cloud computing," *IEEE Communications Magazine*, vol. 60, no. 9, pp. 16-29, 2022.

[21] I. Ullah, H. K. Lim, Y. J. Seok and Y. H. Han, "Optimal Task Offloading with Deep Q-Network for Edge-Cloud Computing Environment," *13th International Conference on Information and Communication Technology Convergence (ICTC)*, 2022, pp. 406-411.

[22] S. Tuli, S. Ilager, K. Ramamohanarao and R. Buyya, "Dynamic scheduling for stochastic edge-cloud computing environments using a3c learning and residual recurrent neural networks," *IEEE transactions on mobile computing*, vol. 21, no. 3, pp. 940-954, 2020.

[23] H. Li, K. Ota and M. Dong, "Learning IoT in edge: Deep learning for the Internet of Things with edge computing," *IEEE Network*, vol. 32, no. 1, pp. 96–101, 2018.

[24] M. Xu, S. Alamro, T. Lan and S. Subramaniam, "Laser: A deep learning approach for speculative execution and replication of deadline-critical jobs in cloud," *Proceedings of the 26th International Conference on Computer Communication and Networks (ICCCN)*, 2017, pp. 1–8.

[25] Q. Zhang, M. Lin, L. T. Yang, Z. Chen, S. U. Khan and P. Li, "A double deep Q-learning model for energy-efficient edge scheduling," *IEEE Transactions on Services Computing*, vol. 12, no. 5, pp.739-749, 2018.

[26] M. Cheng, J. Li and 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. IEEE Press*, 2018, pp. 129–134.

[27] Q. Zhang, M. Lin, L. T. Yang, Z. Chen and P. Li, "Energy efficient scheduling for real-time systems based on deep qlearning model," *IEEE Transactions on Sustainable Computing*, vol. 4, no. 1, pp. 132–141, 2017.

[28] G. Rjoub, J. Bentahar, O. Abdel Wahab and A. Saleh Bataineh, "Deep and reinforcement learning for automated task scheduling in large-scale cloud computing systems," *Concurrency and Computation: Practice and Experience*, vol. 33, no. 23, pp. 1-13, 2021.

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

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

[31] L. Xie, T. Han, H. Zhou, Z. R. Zhang, B. Han and A. Tang, "Tuna swarm optimization: a novel swarm-based metaheuristic algorithm for global optimization," *Computational intelligence and Neuroscience*, 2021, pp.1-22.

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

[33] S. Shen, V. van Beek and A. Iosup, "Statistical characterization of business-critical workloads hosted in cloud datacenters," *15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, 2015, pp. 465–474.