Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2023.DOI

Towards An Optimal Latency-Energy Dynamic Offloading Scheme for Collaborative Cloud Networks

JUI MHATRE[1], (Student Member, IEEE), AHYOUNG LEE[2], (Senior Member, IEEE), and TU N. NGUYEN[3], (Senior Member, IEEE)

[1]Department of Computer Science, Kennesaw State University, Marietta, GA, 30144, USA (e-mail: jmhatre1@students.kennesaw.edu)
[2]Department of Computer Science, Kennesaw State University, Marietta, GA, 30144, USA (e-mail: ahyoung.lee@kennesaw.edu)
[3]Department of Computer Science, Kennesaw State University, Marietta, GA, 30144, USA (e-mail: tu.nguyen@kennesaw.edu)

Corresponding author: Ahyoung Lee (e-mail: ahyoung.lee@kennesaw.edu).

ABSTRACT Growing technologies like virtualization and artificial intelligence have become more popular nowadays because they are more handy and accessible on mobile devices. But lack of resources for processing these applications at the user end and the limited energy of mobile devices are still significant hurdles. Collaborative edge and cloud computing are one of the solutions to this problem. An optimal offloading strategy is required to balance transmission latency for the cloud and limited resources at edge servers. We have proposed a multi-period deep deterministic policy gradient (MP-DDPG) algorithm to find an optimal offloading policy to the collaborative cloud network including the central cloud server, edge cloud servers, and mobile devices constrained by minimization of computation, transmission delay, and energy consumption. The novelty of this algorithm lies in partitioning the task to offload in multiple time slots and reusing cloud and edge resources in every slot, rather than taking a single offloading decision and running out of remote resources by offloading a single large task. Our results show that MP-DDPG achieves the minimum latency and energy consumption in the collaborative cloud network.

INDEX TERMS Collaborative cloud computing, computation offloading, latency, energy efficiency, deep reinforcement learning, multi-period deep deterministic policy gradient

I. INTRODUCTION

WITH boom in the field of high-speed technologies like the Internet of Things (IoT), big data technologies, and machine learning tech trends are growing to new heights. As a consequence of mobile computing, these technologies are made available to users on their mobile devices, thus making it more convenient for them to access those. But this convenience remains at risk due to mobile devices' energy and computation resource limitations. Processing power and memory resources can be increased to improve heavy computation on mobile devices. But this improvement comes along with economic cost. A better solution to this problem is to offload computation to cloud or edge servers which is originally proposed in [1]. With the advent of computation-intensive technologies, insufficiency of computing and storage resources was faced at edge servers. Leveraging cloud resources for excess computation would solve this problem at the cost of transmission latency and energy. Since cloud servers are located at remote locations, both time and energy are consumed in sending queries and receiving the results. Edge servers are a good solution in cases where we require immediate results or in case of inadequate battery life. Edge servers are near mobile devices, because of which time and energy are conserved. However close edge servers are, there is some time and energy consumed, but it wins when it comes to computation on the mobile device itself. But mobile devices do not have sufficient processing power for large artificial intelligence and machine learning tasks. Thus it is necessary to find an equilibrium of offloading ratios for transmitting tasks to the cloud and edge servers. Offloading should be such that, after completing the task, minimum energy is consumed and we obtain results within the time limits specified by the application. Offloading entire tasks to the cloud servers eats up a lot of time and energy. Authors in [2] proposed a partitioning-and-offloading scheme for the heterogeneous tasks-server system to reduce the overall system latency and energy consumption. Partitioning appli-

cations are also discussed in [3] so that the device can obtain the most benefit from computation offloading.

In this paper, we are trying to optimize energy consumption and latency by finding an optimal offloading ratio. The solution to this problem depends on various factors, such as the location of mobile devices, battery capacity of each device and edge servers, accepted delay for applications on their respective devices, and the size of the task to be computed. Previously this problem is solved using machine learning-based algorithms. The novelty of our approach lies in multi-step problem solutions. We present an algorithm to generate a series of offloading ratios such that not necessarily the entire task is completed in the first iteration. It may take multiple offloading iterations. We ensure that the total energy consumed in completing the task evaluation is lower compared to an all-in-one-go task evaluation. We find minimum energy and latency consumption by adjusting the offloading ratios for given inputs of previously mentioned factors. It can be compared to multi-linear problem solving which is an NP-hard problem. Thus our problem of optimizing these continuous features is an NP-hard problem and it cannot be solved in polynomial time. Vast and detailed work is done to solve this optimization problem considering different situations. We formulate our offloading problem as a Multi-Period Markov Decision Process (MP-MDP) [4] and propose an algorithm based on deep deterministic reinforcement learning for a multi-step offloading strategy. Thus, the main **contributions** and **intellectual merits** of this paper are summarized as follows:

- **Research formulation:** We formulate our offloading strategy based on communication and computation time and energy consumption by mobile devices, edge servers, and cloud servers.
- **Algorithm:** We propose *"Multi-Period Deep Deterministic Policy Gradient"* (MP-DDPG) based on the reinforcement learning method for finding optimal offloading strategy by scheduling at each time slot consequently using network resources optimally.
- **Evaluation:** We conduct a comparative study of edge-enabled 1_TIER and 2_TIER architectures based on DDPG and MP-DDPG algorithms. We show how MP-DDPG optimized *delay* and *energy* consumption both.

The remainder of this paper is organized as follows. Section II gives a detailed literature survey of existing offloading strategies using heuristic as well as machine learning-based solutions. In section III, we present the system model of the offloading strategy. We also discuss communication and computation models and formulate the optimization problem. In Section IV, we discuss the offloading problem is MP-MDP and propose an MP-DDPG algorithm based on a Reinforcement Learning approach to solve it. Section V gives simulation results and analysis of the system. Finally, the paper is concluded in section VI.

II. RELATED WORK

As we discussed in the previous section, finding offloading ratio is an optimization problem and is NP-Hard, most of the solutions are based on Machine learning approaches.

A. HEURISTIC BASED OFFLOADING

Initial work is done using heuristic approaches. In [5] online task offloading algorithm that minimizes the completion time of the application on the mobile device is proposed. For sequential task line topology task graphs are used whereas, for the concurrent task, general topology task graphs are used. They design an algorithm using a load-balancing heuristic to offload tasks to the cloud, such that the parallelism between the mobile and the cloud is maximized. In [6] a set of online and batch scheduling heuristics are proposed to offload dynamically arriving independent tasks among mobile nodes. Graph-based partitioning technique is used in [7] to allocate software components to machines in the cloud with heterogeneous infrastructure while minimizing the required bandwidth. Most of the heuristic approaches use graph-based techniques to find the offloading scheme. But they fail to consider factors such as the mobility of devices. Heuristic algorithms cannot evaluate multiple factors such as location, current load of servers, battery capacity, and application requirements. These machine learning-based approaches are the best solution for such NP-hard problems.

B. MACHINE LEARNING OFFLOADING

Offloading decisions are made based on parameters such as network bandwidth, data computation, the amount of data exchanged over the networks, etc. Many of the proposed algorithms used to make offloading decisions are aimed at improving latency and minimizing energy [8]–[11]. In [8], the perform latency optimization in multi-user Mobile edge computation offloading (MECO) system with partial computation offloading with the design objective is to minimize the weighted-sum delay of all devices under the limited communication and computation resource constraints. Offloading strategy is decided in [9] using convex optimization and the Lagrangian approach. The novelty of this paper lies in finding an offloading strategy for devices in motion. [10] using computation offloading using reinforcement learning using Q-Learning approach for latency and energy optimization in edge and cloud environment. Two offloading strategies for unmanned aerial vehicles are discussed in [11]. In the first approach, UAVs share their load with peer UAVs, whereas cloud and edge servers are used in the second approach.

Other criteria include analyzing parameters: server speed, bandwidth, server load, available memory, and data transfer rate between servers and mobile systems. The solutions to solve these problems include partitioning programs and predicting parametric variations in application behavior and execution environment using machine learning approaches are discussed in [9], [12]. In reference, [13], a large state space scenario such as large battery levels, data rates normalized using CNN to find offloading strategies using Q-

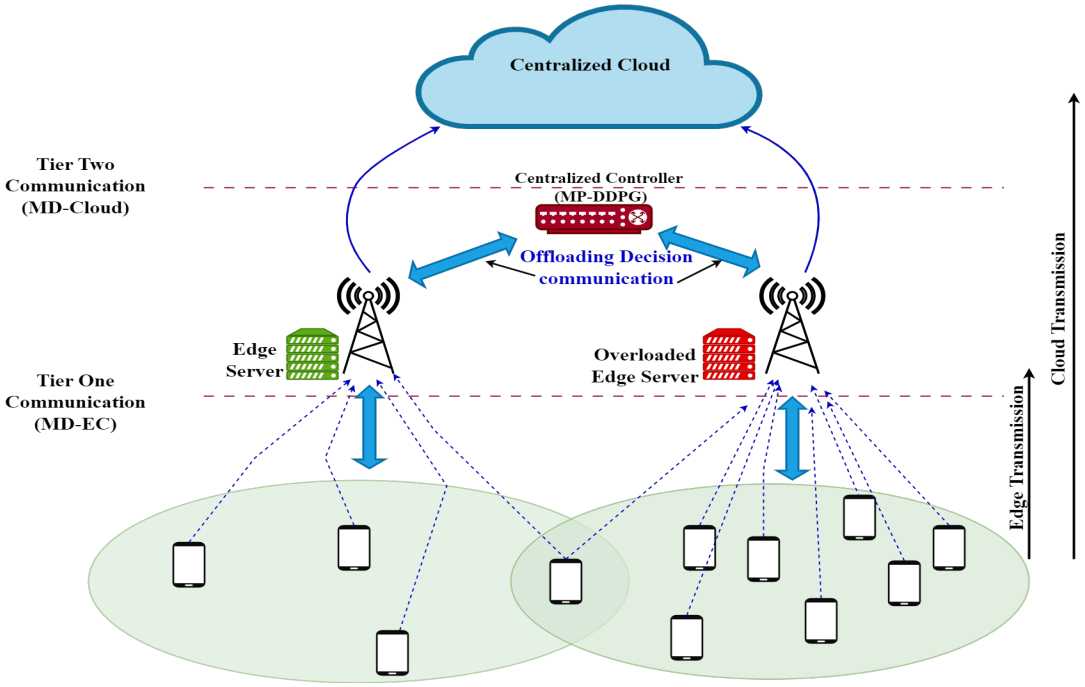

FIGURE 1. Offloading strategies in a collaborative cloud network.

learning. Authors of the reference [14] discuss the advantages of collaborating cloud and edge technologies rather than just using the cloud technology. Reference [15] proposes dynamic offloading decisions based on linear programming and semi-deterministic relaxation algorithms. With improvement in reinforcement learning, deterministic Q-learning [16]–[18] and policy gradient algorithms [19] is brought to use.

It is challenging to do edge server selection and find an offloading strategy for a mobile device that lies in the overlapping region, where the mobile device may be under the coverage of more than one base station. Reference [20] works on overlapping area problems where base stations are deployed in a dense and heterogeneous manner. It converts offloading decision problem to a 0-1 integer optimization problem which makes a binary offloading decision, after that, a binary-coded Genetic Algorithm Based Distributed Offloading Strategy (GABDOS) is applied to obtain a fast near-optimal adaptive offloading decision.

C. REINFORCEMENT LEARNING BASED OFFLOADING

Parallel task offloading for efficient edge server usage is proposed by [21] using a deep deterministic policy gradient (DDPG) approach with an improvement of 19% in ultra-low latency. Reference [22] tackles the mobility issue in the context of web applications focusing on latency minimization using a serialization algorithm. A multi-device, multi-server computation offloading framework for heterogeneous mobile edge computing (MEC) is discussed in [23] to minimize energy consumption, load balance, execution latency, and network usage. They propose Com-DDPG, RL based on LSTM, and a bidirectional recurrent neural network (BRNN). Another similar solution is the FPTT-DDPG algorithm based on DDPG proposed in [19], which uses filter pruned and tensor decomposed deep neural network to solve the optimal offloading decision to minimize user energy consumption and total latency. The reference [24] jointly minimizes the processing delay using DDPG in the unmanned aerial vehicle (UAV) mobile network. The reference [25] compares offloading strategy results of the DQN and DDPG algorithms for the continuous space in their proposed backscatter-assisted hybrid MEC offloading scenario.

Vehicular edge computing is a new upcoming topic. In reference [26] discusses issues of vehicular edge computing and proposes to perform joint service caching and computation offloading in a vehicular edge environment. This optimization is done by formulating the problem as a long-term mixed integer non-linear programming (MINLP). It proposes deep reinforcement learning to obtain a sub-optimal solution with low computation complexity. The simulation results demonstrate an effective performance improvement in a task processing delay. Another reinforcement learning-based solution is proposed in [27]. They first optimized energy consumption using a power scheduling algorithm based on a deep deterministic policy gradient algorithm. Later they performed delay-based optimization using Deep Q-networks (DQN) based algorithm by considering the influence of task queue information, channel state information, and task information. In [28] presents offloading solutions for resource-constrained edge servers and latency-sensitive IoT applications. Hurdles such as insufficient sample diver-

TABLE 1. Time and Energy for Computation of task from MD n on MD itself, edge and CCs

Site	Computation Time (CT)	Transmission Time (TT)	Computation Energy (CE)	Transmission Energy (TE)
Mobile Device	$CT_n^{MD} = \frac{\alpha \times B \times c_n}{f_n} + q_n$	-	$CE_n^{MD} = z \times \alpha \times B \times c_n$	-
Edge Server	$CT_n^{ES} = \frac{\beta \times B \times c_m}{f_m} + q_m$	$TT_n^{ES} = \frac{\beta \times B}{r_{nm}}$	$CE_n^{ES} = \frac{P_m^{idle}}{CT_n^{ES}}$	$TE_n^{ES} = \frac{P_{nm}^{offl}}{TT_n^{ES}}$
Central Cloud	$CT_n^{CC} = \frac{\delta \times B \times c_c}{f_c}$	$TT_n^{CC} = \frac{\delta \times B}{r_{nc}}$	$CE_n^{CC} = \frac{P_c^{idle}}{CT_n^{CC}}$	$TE_n^{CC} = \frac{P_{nc}^{offl}}{TT_n^{CC}}$

sity and high exploration cost, an experience-sharing deep reinforcement learning-based distributed function offloading method called ES-DRL is proposed in the setting of a combined stateful and stateless execution model for serverless edge computing is proposed. In this reference, each Edge Function as a Service (EFaaS) S) obtains the current state of the local environment and inputs them to the local DRL agent, which outputs the function offloading strategy. Reference [29] achieves efficient resource allocation objective by proposing HRL-Edge-Cloud, a novel heuristic reinforcement learning-based multi-resource allocation (MRA) framework, which significantly overcomes the bottlenecks of wireless bandwidth and computes capacity jointly at the Edge cloud and cloud server. They solve the MRA problem by accelerating the conventional Q-Learning algorithm with a heuristic method and applying a novel linear-annealing technique. In [30] a unique problem is highlighted related to the offloading problem of dependent tasks. Care should be taken while dealing with such a problem since there is cause and result relation between modules. They establish the dependent task model as a directed acyclic graph to solve such problems. A Dependent Task-Offloading Strategy (DTOS) based on deep reinforcement learning is proposed with minimizing the weighted sum of delay and energy consumption of network services as the optimization objective.

In previous work, we notice that offloading decision for a particular set of tasks is done once. If edge servers and cloud servers capacity is reached, the remaining workload is computed locally. Due to a lack of resources in a given slot, mobile devices are bound to bear the workload. We identify this loophole and propose a strategy to minimally delay the computation and reap the benefits of resources at edge and cloud servers. While doing this, we also follow delay constraints. This reduces the energy overhead of mobile devices, which remains a concern for all mobile device users.

Thus, through our proposed algorithm, we focus on,

- Optimize energy and time constraints within the limits of the requirement of applications
- Offloading strategy for devices that lie within the range of multiple edge servers.
- Multi-period iterative task offloading, which avoids starvation of devices for results when servers are loaded and occupied with other devices
- Faster availability of servers for processing of newly registered devices for a given base station.

III. SYSTEM MODEL AND PROBLEM FORMULATION

We consider a multi-user, multi-edge computation heterogeneous mobile device network. Our architecture in Fig. 1 has $M = (1, 2, ...m)$ edge servers (ES) deployed at base stations, $N = (1, 2, ...n)$ mobile devices (MD), and there is only one centralized cloud (CC) as $c = 1$. We assume that the tasks are partitionable and do not discuss details of partitioning strategies. Each mobile device $n \in N$ offloads its task to computation sites with different ratios (α for MD, δ for CC, and two ESs with β and γ). This decision of site selection and offloading ratio calculation is made by a centralized controller using the MP-DDPG algorithm. Offloading decisions made at each time slot t gives the opportunity of re-evaluating the available computing resources. Each mobile device $n \in N$ has a computation-intensive and delay-sensitive task to be executed, say its size is B_n for $B_n \geq 0$ and accepted delay, D_n, for $D_n \geq 0$.

A. COMMUNICATION MODEL

For a task generated by MD, there are three types of computation sites available (local computation, edge computation, and cloud computation). Task of size B_n bytes is offloaded to cloud c, edge servers m, and/or computed locally at its mobile device n. Let x can be either an edge computation site or cloud computation site, $x \in (ES, CC)$. For any of these sites, offloading involves some transmission cost. The transmission data rate is determined between n and x by several important factors: transmission power P_{nx} of mobile device n, bandwidth W, and SINR I_{qnx} caused by IoT devices on channel q. The orthogonal frequency division multiple access (OFDMA) methods are adopted to allocate radio and computing resources simultaneously. The transmission rate of mobile device n when offloading to a site x is given by [31] as,

$$r_{nx} = \frac{W}{k} \log_2(1 + \frac{P_{nx} \times h_{qnx}}{(\frac{N_0 W}{k}) + I_{qx}}), \tag{1}$$

where h_{qnx} is channel gain, N_0 is noise power and k is the number of active MDs that offload their computation tasks to site x, then W is divided equally among k.

B. COMPUTATION MODEL

For a device n, the task is executed in two ways: locally or offloaded to an edge server or cloud server. The summary of the computation model is described in Table 1. A detailed explanation of computation is as follows:

Local computation: If the system decides to compute a task of size B bytes locally with ratio α and the computation capacity of mobile device n is f which takes c cycles to compute per task with and queuing delay q, then the total time required can be given as

$$CT_n^{MD} = \frac{\alpha \times B \times c_n}{f_n} + q. \tag{2}$$

Energy consumption for local computation, when z is consumption per cycle, can be given as,

$$CE_n^{MD} = z \times \alpha \times B \times c_n. \tag{3}$$

Edge server computation: If the system decides to compute the task on the edge server with a ratio of α, then in addition to computation time, the transmission time is also involved. Energy consumption is also involved in transmission as well as computation. Transmission energy is incurred from the mobile device which offloads the task to the server. Let the uplink communication data rate for mobile device n to edge server m be r_{nm}. The transmission time and energy are given as,

$$TT_n^{ES} = \frac{\alpha \times B}{r_{nm}}, \tag{4}$$

$$TE_n^{ES} = \frac{P_{nm}^{offl}}{TT_n^{ES}}. \tag{5}$$

Let us consider that the computation resources of the edge server have frequency f_m and take c_m cycles to compute one byte, hence the computation time and energy are defined by,

$$CT_n^{ES} = \frac{\alpha \times B \times c_m}{f_m} + q_m, \tag{6}$$

$$CE_n^{ES} = \frac{P_m^{idle}}{CT_n^{ES}}. \tag{7}$$

Cloud computation: If the system decides to compute the task on the cloud server with a ratio of α, then computation time, transmission time, communication energy, and transmission energy are involved. Let the uplink communication data rate for mobile device n to cloud server c be r_{nc}. The transmission time and energy are given as,

$$TT_n^{CC} = \frac{\alpha \times B}{r_{nc}}, \tag{8}$$

$$TE_n^{CC} = \frac{P_{nc}^{offl}}{TT_n^{CC}}. \tag{9}$$

Let us consider that the computation resources of the cloud server have frequency f_c and take c_c cycles to compute one byte, hence the computation time and energy are defined by,

$$CT_n^{CC} = \frac{\alpha \times B \times c_c}{f_c} \tag{10}$$

$$CE_n^{CC} = \frac{P_c^{idle}}{CT_n^{CC}}. \tag{11}$$

Where P_x^{idle} is power consumed when $x \in (cloud, edgeserver)$ is idle and P_{nx}^{offl} when offloading to x. The total time taken to compute tasks of N mobile devices in τ timeslots is the maximum time spent by any computation site, whereas energy consumed is contributed by each computation site. Mathematically, total time T and energy E can be given by,

$$T = \sum_{t=1}^{\tau}\sum_{i=1}^{N} \max\{CT_{it}^{MD}, (CT_{it}^{ES} + TT_{it}^{ES}), (CT_{it}^{CC} + TT_{it}^{CC})\} \tag{12}$$

$$E = \sum_{t=1}^{\tau}\sum_{i=1}^{N} \{CE_{it}^{MD} + CE_{it}^{ES} + TE_{it}^{ES} + CE_{it}^{CC} + TE_{it}^{CC}\} \tag{13}$$

C. PROBLEM FORMULATION

Our problem is to find offloading ratios in collaborated edge and cloud computing systems to minimize delay and energy consumption of all MDs and ESs. We formulate this optimization as,

$$\begin{aligned} \min_{\alpha,\beta,\gamma,\delta} \Big(& \sum_{i=1}^{N}\sum_{t=1}^{\tau} \alpha_{it}(CT_{it}^{MD} + CE_{it}^{MD}) + \\ & \beta_{it}(TT_{it}^{ES1} + TE_{it}^{ES1} + CT_{it}^{ES1} + CE_{it}^{ES1}) + \\ & \gamma_{it}(TT_{it}^{ES2} + TE_{it}^{ES2} + CT_{it}^{ES2} + CE_{it}^{ES2}) + \\ & \delta_{it}(TT_{it}^{CC} + TE_{it}^{CC} + CT_{it}^{CC} + CE_{it}^{CC})\Big). \end{aligned} \tag{14}$$

s.t.

$$\sum_{t=1}^{\tau} \alpha_{nt} + \beta_{nt} + \gamma_{nt} + \delta_t = 1. \tag{15}$$

$$T_n \leq D_n, \tag{16}$$

$$\sum_{t=1}^{\tau}(CE_{nt}^{MD} + \sum_{i=1}^{M} TE_{nt}^{ES_i} + TE_{nt}^{CC}) < E_{MD_n}^{battery}. \tag{17}$$

for $\forall$ n $\in$ N.

$$\sum_{t=1}^{\tau}\sum_{i=1}^{N} CE_{it}^{ES} \leq E_{ES}^{battery}. \tag{18}$$

$$\max(CT_{it}^{MD}, (TT_{it}^{ES} + CT_{it}^{ES}), (TT_{it}^{CC} + CT_{it}^{CC})), \tag{19}$$

for $\forall t \in \tau \leq \vec{\Delta}T$, where $\vec{\Delta}T$ is the duration of one-time slot.

The equation in (14) gives a minimization equation for energy and time consumed in offloading for two edge servers, one cloud server, and computation at local mobile devices where β and γ are offloading ratios of two edge servers. For scaling our solution to add more edge servers, we can increase offloading ratios such that offloading ratios separately identify each edge server. Constraint in (15) suggests that each mobile device n distributes the entire task to all computation sites over the time period τ. Constraint in (16) guarantees that the task is completed within a specified deadline. Constraints in (17), (18) ensure the battery of MD ain't exhausted and energy consumed by edge servers should be limited. We have set a limit of energy consumption by edge servers and for each understanding, this limit is numerically set using its battery capacity. Constraint in (19) presents that offloading decision is taken per time slot. We design optimization of offloading strategy as an MP-MDP which is a

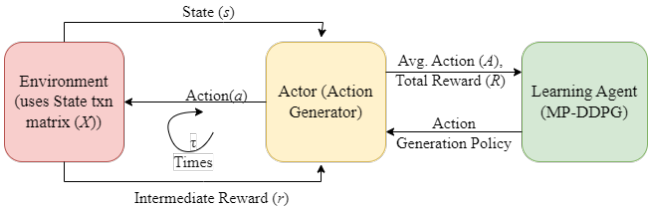

FIGURE 2. Offloading strategy optimization using MP-DDPG, formulated as MP-MDP problem.

5-tuple problem (S, A, R, τ, X) as follows (illustrated in Fig. 2),

- **States (S):** At beginning of each time slot t, the system observes a state s_t of the network, defined as,

$$s_t = \{B(t), D(t), E_x(t), L_n(t)\}, \tag{20}$$

where $B(t) = \{B_1(t), B_2(t)...B_N(t)\}$ is a set of task sizes of all N in MDs, $D(t) = \{D_1(t), D_2(t)...D_N(t)\}$ is a set of their acceptable delays, $E_x(t)$ is a set of batteries left at computation sites x, and S is sequence of states s_t.

- **Actions (A):** Action a_t at time t represents a set of offloading ratios in which tasks are offloaded to computation sites. Actions A is average of a_t taken over period τ, where the action a_t is defined as,

$$a_t = \{\alpha_i(t), \beta_i(t), \gamma_i(t), \delta_i(t)\}, \forall_i \in N, \tag{21}$$

and $\alpha, \beta, \gamma, \delta$ are offloading ratios for different computation sites.

- **Reward (R):** Every action a_t taken to drive system from s_t to s_{t+1} should maximize the reward r_t. From the start state (task submitted for computation) to the final state (task completion) a series of intermediate actions are taken to generate intermediate rewards. Total reward R is the total of all immediate rewards. Since we aim to reduce the overall latency and energy, the reward is calculated as

$$R = -(T + E), \tag{22}$$

where T and E can be found from (12) and (13).

- **State Transition Matrix Providing Next State (X):** Performing action takes the system to the next state which is given as, $\{B(t+1),\ D_i(t) - \vec{\Delta}T,\ \{E_x(t) - E_{xs}\},\ L_n(t+1)\}$ where E_{xs} is site x's battery consumed in state s_t. X denotes a mapping function which calculates s_{t+1} and aids selection of action a_t.
- **Period (τ):** A total number of time-slots taken to complete tasks of mobile device n for $\forall n \in N$ MDs.

IV. PROPOSED METHOD

In this section, we introduce our reinforcement learning-based algorithm called "multi-period deep deterministic policy gradient (MP-DDPG) algorithm" for dynamic resource allocation through offloading strategy optimization spanned over multiple time slots based on Multi-Period Markov Decision Process (MP-MDP) [4].

A. MULTI-PERIOD MARKOV DECISION PROCESS

Let a D be the maximum acceptable delay for a task of size B, where $D \geq \tau \times \vec{\Delta}T$, $\vec{\Delta}T$ is the duration of the one-time slot. MP-MDP process consists of states S and actions A to complete task B. For generating the next state s_{t+1}, an optimal action a_t is selected using a deterministic policy $\mu(s_t)$. The optimal policy is obtained such that the cost corresponding to action is maximum. For a multi-period process, the cost can be calculated as,

$$\begin{aligned} C_\tau(s_\tau, a_\tau) &= C_t(s_t, a_t) + \sum_{i=t+1}^{\tau} c_i(s_i, a_i),\ \forall t < \tau\,,\ \tau\ >\ 1 \\ &= C_\tau(s_\tau, a_\tau),\ \forall t = \tau\,,\ \tau\ >\ 1 \end{aligned} \tag{23}$$

Algorithm 1 Algorithm for DDPG

Require: Randomly initialize the critic network Q(s,a|θ_Q) and actor-network by μ(s|θ_μ) using weights θ_Q and θ_μ.
Ensure: Initialize Replay buffer R_buff
for `episode = 1..M` **do**
  Initialize random noise N_0 for action exploration
  Receive initial observation state s1
  for `t = 1..T` **do**
    $a_t = \mu(s_t|\theta_\mu) + N_0$
    calculate reward r_t by executing a_t on state s_t
    Store transition in buffer R_buff.add(s_t, a_t, r_t, s_{t+1})
    Sample minibatch of K transitions from R_buff
    Set $y' = r(s_t, a_t) + \gamma \times Q'(s'_{t+1}, \mu'(s'_{t+1}))$
    Update critic by minimizing loss L_Q
    Update actor policy using policy gradient $\nabla_{\theta_\mu} L_{\theta_\mu}$
    Update target networks $\theta_{\mu'}$ and $\theta_{Q'}$
  end for
end for

B. DEEP DETERMINISTIC POLICY GRADIENT ALGORITHM

Originally proposed in [32], Deep Deterministic Policy Gradient (DDPG) (Algorithm 1) is an algorithm that concurrently learns a Q-function and a policy. It is efficient in solving problems with continuous action spaces. DDPG has two main networks, one is the critic and another is the actor-network. They further contain subnetworks, online neural networks, and target neural networks which have the same design. These four networks combine their efforts to generate an optimized solution by training their corresponding parameters (θ). These parameters define a policy function $\mu_\theta(s)$. The actor-network evaluates this policy function to choose action a which generates maximum reward. Critic network uses the value function to calculate the Q-value, $Q_\theta(s, a)$ of the selected action. As discussed in [32], the critic network computes the loss as the Mean-Squared Error between the TD-Target and the Q-value estimate of s and a. The TD-Target y' utilizes the target networks. The next state s' and the

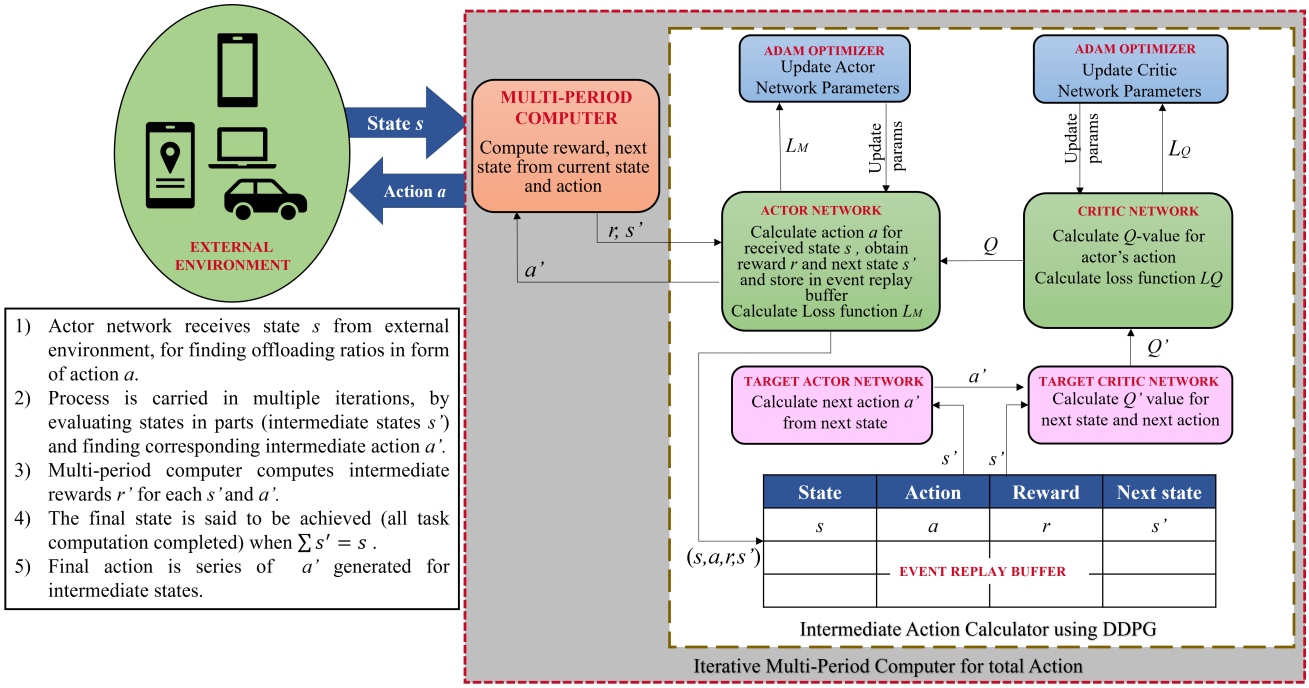

FIGURE 3. Offloading strategy optimization using MP-DDPG, formulated as MP-MDP problem.

Algorithm 2 Algorithm for MP-DDPG

Input: Randomly initialize the critic network Q(s,a|θ_Q) and actor network by μ(s|θ_μ) using weights θ_Q and θ_μ.
Output: Initialize Replay buffer R_buff

for `episode = 1...M` **do**
  Initialize random noise N_0 for action exploration
  Receive initial observation state s1 with task size B
  while B < 0 **do**
    $a_{it} = \mu(s_t|\theta_\mu) + N_0$
    calculate reward r_{it} by executing a_{it} on state s_{it}
  end while
  $A = \sum a_{it}$
  $R = \sum r_{it}$
  $S_{t+1} = s_{it}$
  $S_t = s1$
  Store transition(S_t, A, R_t, S_{t+1}) in buffer R_buff
  Sample minibatch of K transitions from R_buff
  Set $y' = r(s_t, a_t) + \gamma \times Q'(s'_{t+1}, \mu'(s'_{t+1}))$
  Update critic by minimizing loss L_Q
  Update actor policy using policy gradient $\nabla_{\theta_\mu} L_{\theta_\mu}$
  Update target networks $\theta_{\mu'}$ and $\theta_{Q'}$
end for

associated action predicted by the target actor μ' are provided as input to the target critic Q', can be formulated as:

$$y' = r(s,a) + \gamma \times Q'(s', \mu'(s')), \tag{24}$$

where $r(s,a)$ is reward obtained when actor network chose action a in state s, γ is discount factor and $Q'(s', \mu'(s'))$ is maximum future return of the next state. DDPG uses a buffer to sample minibatch of size $\dot{m}$, for calculating the loss function of critic and then learning by minimizing the loss function given by,

$$L_Q = \frac{1}{\dot{m}} \sum_{i=1}^{\dot{m}} (y_i - Q(s, \mu_\theta(s)))^2, \tag{25}$$

where $Q(s, \mu_\theta(s))$ is Q-value estimate of the current state and corresponding action at time t. For Actor-network, we need to maximize the Q value. Since the action space is continuous, we cannot calculate the max of Q-value for a set of actions. For continuous values, maximizing a function is the same as minimizing the negative value of that function. Hence loss function of actor-network is as

$$L_\mu = -Q(s,a). \tag{26}$$

Here loss function is in terms of Q and not in terms of θ, hence we apply the chain rule to optimize it.

$$\nabla_\theta L_\mu = \frac{1}{\dot{m}} \sum_{i=1}^{\dot{m}} (\nabla_{\mu(s)} Q_\mu(s, \mu(s)) \nabla_\theta \mu_\theta(s)). \tag{27}$$

The parameters of the actor and critic network are updated using optimized loss functions in (30) and (32). Finally, the DDPG agent uses a small constant ζ to update the critic target network and actor target network softly:

$$\begin{aligned} \theta_{\mu'} &= \theta_\mu + (1-\zeta)\theta_{\mu'} \\ \theta_{Q'} &= \theta_Q + (1-\zeta)\theta_{Q'}. \end{aligned} \tag{28}$$

C. MULTI-PERIOD DEEP DETERMINISTIC POLICY GRADIENT ALGORITHM

MP-DDPG algorithm (Algorithm 2) is designed to reevaluate the resources of our network at the beginning of each time slot and take offloading decisions based on the current state. At the beginning of time slot t, the actor-network evaluates the current state containing the remaining task $B(t)$ and uses deterministic policy function $\mu_\theta(s)$ to choose action a_t. Critic network uses the value function to calculate the Q-value, $Q_\theta(s_t, a_t)$ of the selected action. After computation of task $B(t)$, i.e. after $D(t)$ time, network parameters as learned. Fig. 3 shows the working of the MP-DDPG algorithm. It has two modules, the outer module is a multi-period iterative computer and the inner module is an intermediate action calculator. For every intermediate state s_t, which is executed in consecutive time slots, the actor in the inner module generates intermediate actions a_t using the policy function. These actions define the offloading decisions taken at that time slot. This action is forwarded to the outer module to calculate the reward for the action. Next remaining state and calculated intermediate reward are again given as input to the inner module. The inner module evaluates the rewards and adjusts the parameters of policy to generate the next reward. This process continues until it completes the given task of B bytes by executing a part of it B_t at each time t. At each intermediate step, parameters are learned by actor and critic networks. Actor and Critic are two Temporal-Difference (TD) versions of policy gradient. The actor takes action and the critic tells how good the action was and how to adjust it or we can say, how to learn the parameters to adjust the action. The learning of the actor is based on the policy gradient approach. The TD-Target y' utilizes the target networks. The next state s'_{t+1} and the associated action predicted by the target actor μ' are provided as input to the target critic Q', can be formulated as:

$$y' = r(s_t, a_t) + \gamma \times Q'(s'_{t+1}, \mu'(s'_{t+1})), \tag{29}$$

where $r(s_t, a_t)$ is reward obtained when actor network chose action a_t in state s_t, γ is discount factor and $Q'(s'_{t+1}, \mu'(s'_{t+1}))$ is maximum future return of the next state. Loss function at critic is given by,

$$L_Q = \frac{1}{\dot{m}} \sum_{i=1}^{\dot{m}} (y_i - Q(s_t, \mu_\theta(s_t)))^2, \tag{30}$$

where $Q(s_t, \mu_\theta(s_t))$ is Q-value estimate of the current state and corresponding action at time t. For Actor-network, we need to maximize the Q value. Since the action space is continuous, we cannot calculate the max of Q-value for a set of actions. For continuous values, maximizing a function is the same as minimizing the negative value of that function. Hence loss function of actor-network is as

$$L_\mu = -Q(s_t, a_t). \tag{31}$$

Here loss function is in terms of Q and not in terms of θ, hence we apply the chain rule to optimize it.

$$\nabla_\theta L_\mu = \frac{1}{\dot{m}} \sum_{i=1}^{\dot{m}} (\nabla_{\mu(s)} Q_\mu(s_t, \mu(s)) \nabla_\theta \mu_\theta(s)). \tag{32}$$

The parameters of the actor and critic network are updated using optimized loss functions in (30) and (32). The critic-target-network and actor-target network are updated softly as discussed in (28).

V. RESULTS AND ANALYSIS

We carried out reinforcement learning with parameters and optimized hyperparameters as discussed in [24]. We use learning rate $\alpha_{LR_{actor}} = 0.0001$ and $\alpha_{LR_{critic}} = 0.002$. The summary of our simulation parameters is shown in Table 2. For comparison with the DDPG algorithm proposed in [24], we have used 3 variations in architecture with the MP-DDPG

TABLE 2. Simulation Parameters

Symbol	Description	Value
N	Number of mobile devices	4
M	Number of edge servers	2
C	Number of Centralized Cloud	1
$R_m[]$	Range in Miles for $M[i]$	50, 75 miles
$\vec{\Delta}T$	Time slot	5 ms
$C[]$	Computation cycles for per byte	0.0001 (local), 0.00001 (EC)
$B[]$	Task size list	[2,140] MB
$D[]$	Accepted delay list	Random (5 ms, 11 ms)
$Q[]$	Cache at each user	1GB
W_{ij}	Channel bandwidth b/w for user i and EC j	From UE to EC - 10^6 MBps
P_{mn}	Transmission power for user n to ECm	0.1 W
N_0	Noise	-100 dBm
h_{mn}	Channel gain for channel between user n to EC m	6×10^{-10}
f_d	Frequency of local processor	0.5 Ghz
f_m	Frequency of EC processor	5 Ghz
f_c	Frequency of cloud processor	10 Ghz
z	Energy consumption per cycle	4/3 $\times$ P^{idle}
x_L	Energy consumption per cycle by local when idle	15 mJ
x_M	Energy consumption per cycle by edge server when idle	75 mJ
x_C	Energy consumption per cycle by cloud server when idle	0.3 J
$E[]$	Battery capacity list	5000 J
$L_d[]$	Location of User devices	random ((0,0) (101,101))
ns_area	Network simulation area	500 $\times$ 500
$L_M[]$	Location of EC servers	[50, 50], [100, 100]

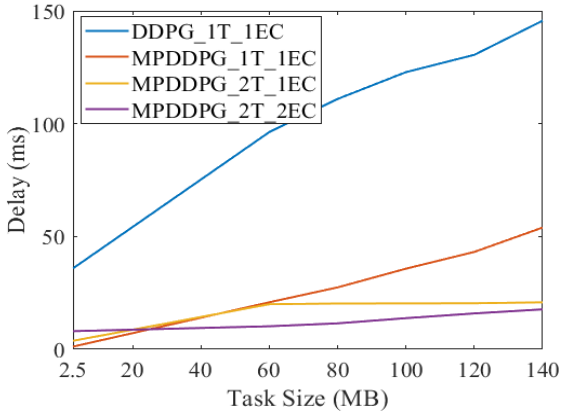

FIGURE 4. Comparisons of computation latency for varying task sizes based on our testing models shown in Table 3.

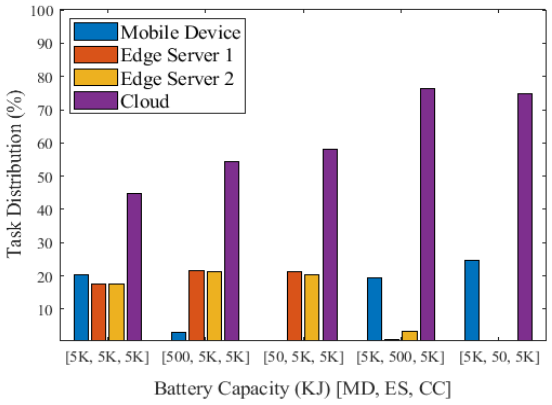

FIGURE 6. Comparisons of task distributions for varying battery capacity.

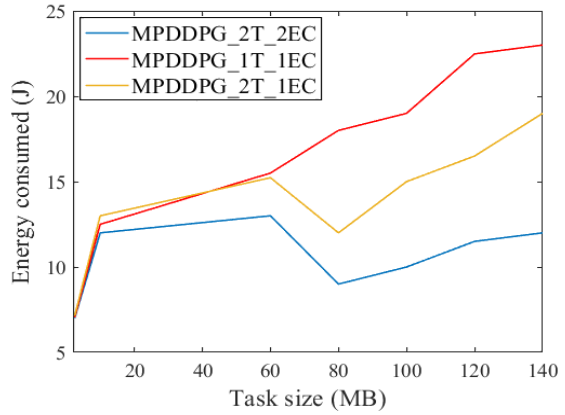

FIGURE 5. Comparisons of computation energy consumption for varying task sizes based on our testing models shown in Table 3.

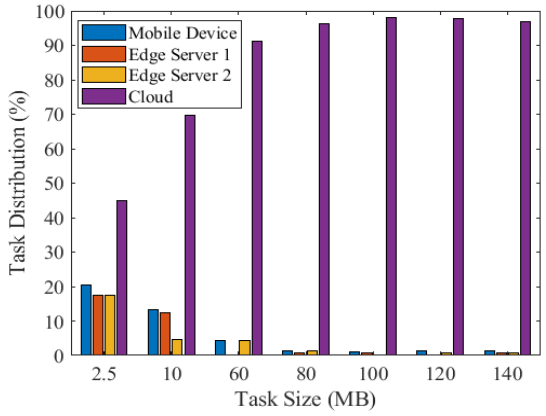

FIGURE 7. Comparisons of task distributions for varying task size.

algorithm deployed on the controller. They are described in Table 3.

Fig. 4 shows the latency as a function of the task size of each model discussed in Table 3. MP-DDPG outperforms [24] by finding offloading strategy which exhibits lower latency. Using a multi-period strategy to offload has lower latency as compared to single-slot offloading. When we use *2-tier* model (cloud + edge computing), performance is further improved as compared to *1-tier* model (edge computing).

TABLE 3. Model description

Model	Algorithm	Number of ES	Number of Cloud
1_TIER_DDPG_1EC	DDPG [24]	1	0
1_TIER_MP-DDPG_1EC	MP-DDPG	1	0
2_TIER_MP-DDPG_1EC	MP-DDPG	1	1
2_TIER_MP-DDPG_2EC	MP-DDPG	2	1

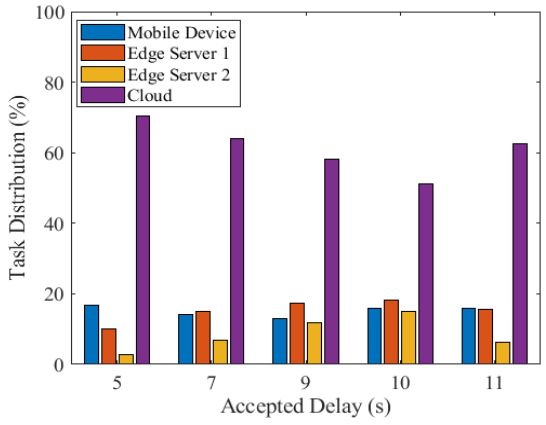

FIGURE 8. Comparisons of task distributions for varying deadlines.

An increasing number of ESs improve performance but that improvement pertains to computation from MDs which lie in overlapping areas of the range of adjacent base stations. Fig. 5 shows a comparison of drainage levels of MD batteries

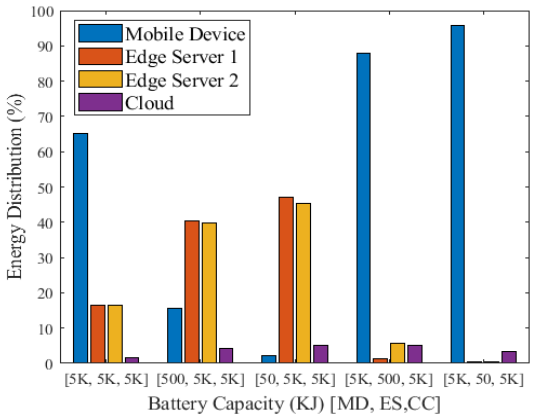

FIGURE 9. Comparisons of energy consumption distributions for varying battery capacity.

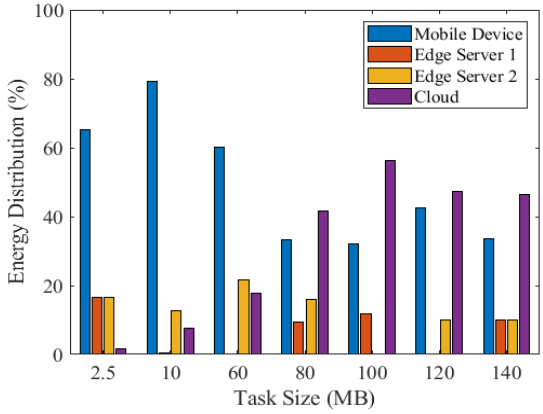

FIGURE 10. Comparisons of energy consumption distributions for varying task size.

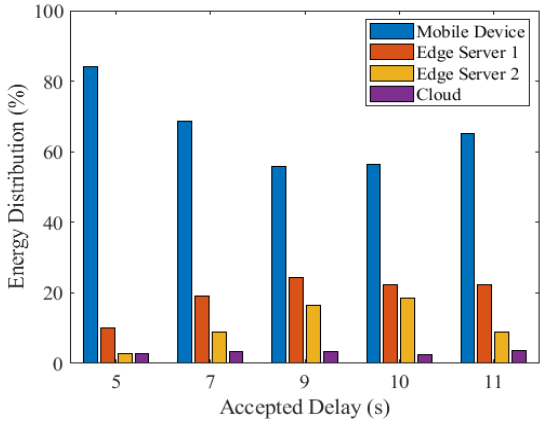

FIGURE 11. Comparisons of energy consumption distributions for varying deadlines.

(energy consumption) for varying task sizes on different models using MP-DDPG-based algorithms. Initially, when task sizes are small, the task is executed locally on MD, hence

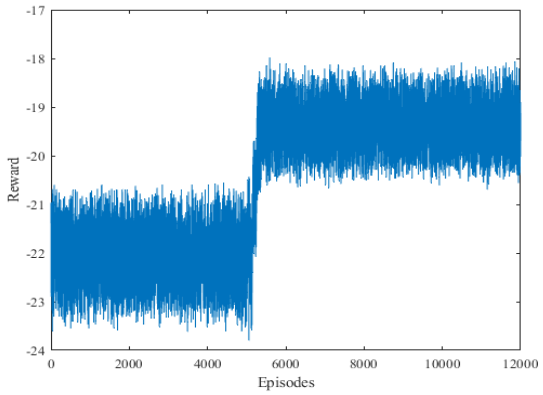

FIGURE 12. Learning Iterations taken by MP-DDPG algorithm to optimize the results.

energy consumed in all three cases is the same. But as the task size increases, MD battery consumption increase and there arises a need to offload the tasks. The first and best option is always ESs. Hence we see a drop in battery consumption between 60-80 MB task sizes. Moreover, *2_TIER_MP-DDPG_2EC* provides two ESs, and hence drop is more and battery consumption remains low for higher task sizes as well. Further increase in task sizes involves offloading to CC. This increases to-cloud transmission overhead in addition to increased local computation at MD. Hence we observe that the rate of battery usage decreases when we resort to offloading strategy as compared to local computation. Though the rate of energy consumption increases for offloading to the cloud, it remains less as compared to the non-offloading case.

We have further shown task distribution and energy distribution for *2_TIER_MP-DDPG_2EC*. Task Distributions in Figs. 6, 7, and 8 give a detailed view of what percentage of tasks are offloaded to ESs and CC, and what part of the task is computed locally in various scenarios. Similarly Energy distributions in Figs. 9, 10, and 11 explain how much energy is consumed by each computation site for transmission and computation. Figs. 6 and 9 are about offloading strategy results when the batteries of the computation site change. The Y-axis gives the battery levels as (MD, ES, and CC) in KJ. In Fig. 6, we observe that ESs are utilized to the fullest. When the device battery is reduced, the excess computation which is not sufficed by MD is offloaded to the cloud. Whereas, when the battery capacity of ESs is reduced, excess computation is distributed to MD and then to CC. It is observed that the system always tries to lessen the offloading to CC since it consumes more time for transmission due to its distance. In Fig. 9 we observe that total energy consumed at both ESs and energy consumed at MDs is more compared to the cloud since first priority for task offloading is ES and then local or the CC. As per our parameters, (delay + energy) for computation and transmission to CC is more as compared to computation in MD. But as the battery capacity of the device decreases, the computation is switched to CC.

Figs. 7 and 10 give task and energy distribution for varying task sizes. Fig. 7 shows how the tasks are distributed when task size increases. We observe that once the capacity of ESs and MD is reached, the task is offloaded to CC. It is observed that for larger task sizes, even with a multi-period offloading strategy, the percentage of computation on the cloud is higher as compared to local MDs or ESs. Even though we see the percentage of task distributed is less for larger task sizes, Fig. 10 confirms that the cloud offloading takes place only after energy requirements are satisfied by ESs and devices. We observe that as the task size increases, energy utilization of local MD and ES increases but after the limit, as attained there comes a need to offload the task to CC.

Each task has some tolerable delay specified by its application. The task is required to complete within that delay limit. Figs. 8 and 11 give us details of how our system behaves when accepted delay for each task changes. We observe from Fig. 8 that MD can suffice tasks faster so the first choice of task computation is always MD then ES and CC due to its transmission cost. But when the delay tolerance value increases, the CC resources are utilized so that low tolerance tasks could utilize local and ESs. Fig. 11 gives insight that local energy is mostly used when an acceptable delay is at stake. As the tolerance threshold is increased, ES and CC resources are used.

Fig.12 shows learning in our proposed MP-DDPG algorithm using parameters shown in Table 2. We used 12000 iterations to optimize our parameters. We see that around 6000 iterations were required to increase the reward which reduced energy and time shown in (22).

VI. CONCLUSION

Our paper aims to design an algorithm for finding optimal offloading strategies in collaborative edge and cloud computing networks. We have designed a multi-period deep deterministic reinforcement learning (MP-DDPG) algorithm which gives an optimal offloading policy and utilizes maximum network resources so as to minimize latency and energy in the network system. We have compared our proposed MP-DDPG algorithm to the existing DDPG algorithm [24] in terms of latency for offloading. MP-DDPG algorithm achieves about 71% improvement as compared to the DDPG algorithm for 1-tier architecture. We further show improvement in latency and energy consumption for higher architectures of MP-DDPG described in table 3. Furthermore, we have shown the distribution of tasks over ESs, CC, and MD for computation. We observe that larger task sizes offload their major share to the cloud and conserve the energy of MD. But real-time computation prefers computation on MD than offloading to the cloud. This makes our algorithm more useful in a real-time environment in high-speed next-generation collaborative networks. Our algorithm also conserves the energy of both MD and ESs.

ACKNOWLEDGEMENTS

This research was supported in part by US NSF grants CNS-2103405 and AMPS-2229073.

REFERENCES

[1] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE pervasive Computing*, vol. 8, no. 4, pp. 14–23, Oct 2009.

[2] Z. Liao, W. Hu, J. Huang, and J. Wang, "Joint multi-user dnn partitioning and task offloading in mobile edge computing," *Ad Hoc Networks*, vol. 144, p. 103156, 2023.

[3] H. Wu, W. Knottenbelt, K. Wolter, and Y. Sun, "An optimal offloading partitioning algorithm in mobile cloud computing," in *Quantitative Evaluation of Systems: 13th International Conference, QEST 2016, Quebec City, QC, Canada, August 23-25, 2016, Proceedings 13*. Springer, 2016, pp. 311–328.

[4] W. L. Cooper and B. Rangarajan, "Performance guarantees for empirical markov decision processes with applications to multiperiod inventory models," *Operations Research*, vol. 60, no. 5, pp. 1267–1281, 2012.

[5] M. Jia, J. Cao, and L. Yang, "Heuristic offloading of concurrent tasks for computation-intensive applications in mobile cloud computing," in *2014 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2014, pp. 352–357.

[6] B. Li, Y. Pei, H. Wu, and B. Shen, "Heuristics to allocate high-performance cloudlets for computation offloading in mobile ad hoc clouds," *The Journal of Supercomputing*, vol. 71, no. 8, pp. 3009–3036, 2015.

[7] T. Verbelen, T. Stevens, F. De Turck, and B. Dhoedt, "Graph partitioning algorithms for optimizing software deployment in mobile cloud computing," *Future Generation Computer Systems*, vol. 29, no. 2, pp. 451–459, 2013.

[8] J. Ren, G. Yu, Y. Cai, and Y. He, "Latency optimization for resource allocation in mobile-edge computation offloading," *IEEE Transactions on Wireless Communications*, vol. 17, no. 8, pp. 5506–5519, 2018.

[9] Y. Liu, C. Liu, J. Liu, Y. Hu, K. Li, and K. Li, "Mobility-aware and code-oriented partitioning computation offloading in multi-access edge computing," *Journal of Grid Computing*, vol. 20, no. 2, pp. 1–15, 2022.

[10] R. Yadav, W. Zhang, I. A. Elgendy, G. Dong, M. Shafiq, A. A. Laghari, and S. Prakash, "Smart healthcare: Rl-based task offloading scheme for edge-enable sensor networks," *IEEE Sensors Journal*, vol. 21, no. 22, pp. 24 910–24 918, 2021.

[11] A. A. A. Ateya, A. Muthanna, R. Kirichek, M. Hammoudeh, and A. Koucheryavy, "Energy-and latency-aware hybrid offloading algorithm for uavs," *IEEE Access*, vol. 7, pp. 37 587–37 600, 2019.

[12] Z. Cheng, M. Min, M. Liwang, L. Huang, and Z. Gao, "Multiagent ddpg-based joint task partitioning and power control in fog computing networks," *IEEE Internet of Things Journal*, vol. 9, no. 1, pp. 104–116, 2021.

[13] W. Fan, J. Han, L. Yao, F. Wu, and Y. Liu, "Latency-energy optimization for joint wifi and cellular offloading in mobile edge computing networks," *Computer Networks*, vol. 181, p. 107570, 2020.

[14] P. Lagabka, A. Lee, K. Suo, and D. Kim, "Seamless communication techniques in mobile cloud computing: A survey," *ITU Journal on Future and Evolving Technologies*, 2021. [Online]. Available: https://www.itu.int/pub/S-JNL-VOL2.ISSUE2-2021-A01

[15] T. Q. Dinh, J. Tang, Q. D. La, and T. Q. Quek, "Offloading in mobile edge computing: Task allocation and computational frequency scaling," *IEEE Transactions on Communications*, vol. 65, no. 8, pp. 3571–3584, 2017.

[16] L. Huang, X. Feng, C. Zhang, L. Qian, and Y. Wu, "Deep reinforcement learning-based joint task offloading and bandwidth allocation for multi-user mobile edge computing," *Digital Communications and Networks*, vol. 5, no. 1, pp. 10–17, 2019.

[17] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji, and M. Bennis, "Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4005–4018, 2018.

[18] D. Li, S. Xu, and P. Li, "Deep reinforcement learning-empowered resource allocation for mobile edge computing in cellular v2x networks," *Sensors*, vol. 21, no. 2, p. 372, 2021.

[19] X. Chen, H. Ge, L. Liu, S. Li, J. Han, and H. Gong, "Computing offloading decision based on ddpg algorithm in mobile edge computing," in *2021 IEEE 6th International Conference on Cloud Computing and Big Data Analytics (ICCCBDA)*. IEEE, 2021, pp. 391–399.

[20] Z. Liao, J. Peng, B. Xiong, and J. Huang, "Adaptive offloading in mobile-edge computing for ultra-dense cellular networks based on genetic algorithm," *Journal of Cloud Computing*, vol. 10, no. 1, pp. 1–16, 2021.
[21] H. Zhang, Y. Yang, X. Huang, C. Fang, and P. Zhang, "Ultra-low latency multi-task offloading in mobile edge computing," *IEEE Access*, vol. 9, pp. 32 569–32 581, 2021.
[22] H.-J. Jeong, C. H. Shin, K. Y. Shin, H.-J. Lee, and S.-M. Moon, "Seamless offloading of web app computations from mobile device to edge clouds via html5 web worker migration," in *Proceedings of the ACM Symposium on Cloud Computing*, 2019, pp. 38–49.
[23] H. Gao, X. Wang, X. Ma, W. Wei, and S. Mumtaz, "Com-ddpg: A multiagent reinforcement learning-based offloading strategy for mobile edge computing," *arXiv preprint arXiv:2012.05105*, 2020.
[24] Y. Wang, W. Fang, Y. Ding, and N. Xiong, "Computation offloading optimization for uav-assisted mobile edge computing: a deep deterministic policy gradient approach," *Wireless Networks*, vol. 27, no. 4, pp. 2991–3006, 2021.
[25] Y. Xie, Z. Xu, Y. Zhong, J. Xu, S. Gong, and Y. Wang, "Backscatter-assisted computation offloading for energy harvesting iot devices via policy-based deep reinforcement learning," in *2019 IEEE/CIC International Conference on Communications Workshops in China (ICCC Workshops)*. IEEE, 2019, pp. 65–70.
[26] Z. Xue, C. Liu, C. Liao, G. Han, and Z. Sheng, "Joint service caching and computation offloading scheme based on deep reinforcement learning in vehicular edge computing systems," *IEEE Transactions on Vehicular Technology*, 2023.
[27] L. Liao, Y. Lai, F. Yang, and W. Zeng, "Online computation offloading with double reinforcement learning algorithm in mobile edge computing," *Journal of Parallel and Distributed Computing*, vol. 171, pp. 28–39, 2023.
[28] X. Yao, N. Chen, X. Yuan, and P. Ou, "Performance optimization of serverless edge computing function offloading based on deep reinforcement learning," *Future Generation Computer Systems*, vol. 139, pp. 74–86, 2023.
[29] A. Qadeer and M. J. Lee, "Hrl-edge-cloud: Multi-resource allocation in edge-cloud based smart-streetscape system using heuristic reinforcement learning," *Information Systems Frontiers*, pp. 1–17, 2023.
[30] B. Gong, X. Jiang *et al.*, "Dependent task-offloading strategy based on deep reinforcement learning in mobile edge computing," *Wireless Communications and Mobile Computing*, vol. 2023, 2023.
[31] X. Huang, S. Leng, S. Maharjan, and Y. Zhang, "Multi-agent deep reinforcement learning for computation offloading and interference coordination in small cell networks," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 9, pp. 9282–9293, 2021.
[32] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *International conference on machine learning*. PMLR, 2014, pp. 387–395.

JUI MHATRE (Member, IEEE) (M'76–SM'81–F'87) is currently pursuing her second Master's degree in Computer Science with Thesis from Kennesaw State University, Marietta, Georgia, USA. The First Masters's degree was obtained from VJTI, Mumbai, Maharashtra, India, 2017. Since 2021, she is currently a Research Assistant with the Kennesaw State University under the supervision of Dr. Ahyoung Lee in the Intelligent Computing & Networking (ICN) Laboratory and a part of the Vertically Integrated Project (VIP) team in the L3BN: Low-Power Low-Cost Long-Range Broadband Networking Lab. Currently working in IoT, AI, and Wireless network optimization domains, she has industry experience as a Software Engineer in the banking domain. She has also done research in the field of data mining in geoinformation Systems for spatiotemporal data.

AHYOUNG LEE (Senior Member, IEEE) received her M.S., and Ph.D. degrees in computer science and engineering from the University of Colorado, Denver, in 2006 and 2011, respectively. she was a Postdoctoral Fellow at Georgia Institute of Technology in the Broadband Wireless Networking Laboratory (BWN Lab) under the supervision of Prof. Ian F. Akyildiz with a research project focused on Software Defined Networking (SDN). Currently, she is an Assistant Professor with the Department of Computer Science at Kennesaw State University; the Director of the Intelligent Computing & Networking (ICN) Laboratory and the VIP team in the L3BN: Low-Power Low-Cost Long-Range Broadband Networking Lab. Her main research interests include algorithm design and analysis in SDN, mobile wireless networks, sensor networks, cyber-physical systems, and LoRa networks; optimal computing and networking in edge/cloud/quantum computing; applied data science with machine learning in network communications.

TU N. NGUYEN (Senior Member, IEEE) is currently an assistant professor of Computer Science at Kennesaw State University, USA. Prior to joining the KSU, he was an assistant professor of computer science at Purdue University Fort Wayne. He earned a Ph.D. degree in electronic engineering from the National Kaohsiung University of Science and Technology (formerly, National Kaohsiung University of Applied Sciences) in 2016. He was a Postdoctoral Associate in the Department of Computer Science Engineering, University of Minnesota - Twin Cities in 2017. Prior to joining the University of Minnesota, he worked at the Missouri University of Science and Technology as a Postdoctoral Researcher in the Intelligent Systems Center in 2016. His research focuses on developing fundamental mathematical tools and principles to design and develop smart, secure, and self-organizing systems, with applications to network systems, cyber-physical systems, quantum networks, and quantum computing. His research has resulted in more than 100 publications in leading academic journals as well as conferences. Dr. Nguyen has engaged in many professional activities, including serving as Editor/Guest Editor for academic journals such as an Associate Editor of IEEE Systems Journal, Associate Editor of Journal of Combinatorial Optimization, Associate Editor of IEEE Access, leading guest editor of IEEE Transaction on Computational Social Systems, IEEE Internet of Things Magazine, and IEEE Journal of Biomedical and Health Informatics, and he is also a Technical Editor of Computer Communication. He is the Co-Editor-in-Chief of the book series: IET Advances in Distributed Computing and Block-chain Technologies. He has been in different Organizing Committees such as being TPC-chairs and general chairs for several IEEE/ACM/Springer flagship conferences. He has also served as a technical program committee (TPC) member for several international conferences. He is an US NSF CRII Award recipient of 2021, a member of ACM, and a senior member of IEEE.