Multi-Agent Reinforcement Learning based Online Distributed Server Selection for Cloud-enhanced Crowdsourced Video Generation and Streaming

Christopher Griffith *(Link to Interactive Jupyter Notebook)*

***Abstract*—A server selection schema and user association optimality policy must be established to properly migrate traffic from content generation centers to requested viewers while maintaining quality of service (QoS) and optimizing the transmission of such services. Local or Multi-access Edge Computing (MEC) cloud distributed networks invested and exploited to minimize transmission times within network traffic; however, current industry schemas and optimization policies are either computationally taxing on resources, complex in model implementation without satisfactory performance or do not achieve appropriate optimality for multiple content generators requested by multiple viewers in real-time. This paper explores a generalized system model of the overall data transmission time for video generation and streaming cases between multiple content generation and viewership via a model-free reinforcement learning schema derived optimization policy. In order to achieve minimization convergence for a multitude of content simultaneously in transit within a MEC network, a Multi-Agent Reinforcement Learning (MARL) algorithmic solution is proposed to obtain a system makespan minimization and leverages the state and action spaces with respect to the time-dependencies of data processing for content generation, interrelation and delivery optimizations. Results show that the MARL schema produces optimal solutions similar to the objective functions of linear programming models while decreasing the computational time significantly for higher order multi-agent systems. However, more advanced neural network approaches and policy gradient models may improve performance for more complex multi-agent systems.**

***Keywords*—MEC, MARL, Streaming, Viewership, Computational Resources**

I. INTRODUCTION

Mobile or Multi-access Edge Computing (MEC) is at the forefront of revolutionizing how computational tasks are distributed, managed, offloaded, and optimized amongst cloud-server data centers and user-end local devices, e.g. mobile devices, wireless cellular networks and operational servers connected to various types of base station networks. Model-free reinforcement learning (RL) algorithms are commonly formulated and implemented to properly engage with the stochastic parameters and features of such system models, i.e. algorithms that do not depend on generated transition probabilities or predictions of policies, action-value functions, value functions and environment models. The standard objective of similar system models is to maximize a user's quality of experience (QoE) preferences by actively making dynamic decisions to achieve conditions favorable for a viewer. Optimal policies from the model-free RL algorithms can achieve this by minimizing offloading computation times, device or server energy consumption, computational tasks queues, server-side computational idle times, general transmission latency, or in our specific investigative problem, to minimize the overall data transmission time for video streaming cases between content generators and viewers via local cloud server selection and user association.

Considering the server placement and user association scheme described in the principal literature is a NP-hard minimization problem, the approach explored in the problem (1) relies heavily on the utilization of traditional linear optimization modeling, (2) does not account for the continuous time dependency parameters of quality of service (QoS) which constantly reflect dynamic server network conditions and (3) exposes its limitations to one snapshot of the system at a given time [1] where in the real world servers can be overloaded by content traffic and viewership requests. Although similar makespan optimization problems in other industry domains have transitioned to deep reinforcement learning (DRL) techniques for solutions, the adoption of such methodological improvements for cloud-distributed resource management is still in its infancy [2], [3], [4], [5], [6], [7]. Deep Reinforcement Learning (DRL) methods, federated learning based algorithms and Multi-agent reinforcement learning (MARL) schemas can be extended to remove the constraints in the principal literature while significantly enhancing the current optimization results. The exploration for alternative approaches will provide an opportunity to acquire knowledge in a new domain, further develop required iterative methodologies and principles while improving the effectiveness of reinforcement learning and neural network implementations.

Hundreds of thousands of users' requests and concurrent broadcasters at a given time evolves into numerous optimization problems that can become heavily computationally taxing and resource demanding. Linear programming (LP) approaches have historically been used to investigate problems discussed in this paper; however, as the complexity of a system increases the limitations of LP become increasingly clear. Using LP as an approach to reach optimality is not realistic in any real-time real world MEC network problem due to resource allocation and impractical computational times. Therefore, reinforcement learning models and other advanced neural networks are implemented to circumvent such restrictions and their performance metrics are compared to LP approaches as a benchmark. Hence, a more computationally liberating model is preferred and many resources are being poured into research to develop algorithms to reach such standards.

Our proposed model attempts to utilize a simplistic reinforcement learning technique to achieve optimal policies for a similar makespan optimization problem as the principal paper [1]. Multiple agents within the proposed multi-agent reinforcement learning (MARL) schema would act as the content transmitting to requested viewers across a MEC network. Similar to the linear programming benchmark model, the MARL approach achieved optimality for the minimization of the transmission time through the network system. More interestingly, the model drastically outperformed the linear programming benchmark computationally. In more complex network simulations, the computational time and resource demands for the MARL model were commendatory due to how swiftly optimal convergence was achieved and how little resources the MARL model required to perform favorably.

II. RELATED WORKS

Wei et al. [8] proposed a deep reinforcement learning-based computation offloading scheme to reduce the computational time and energy consumption of a computation-intensive and latency-sensitive task. They implemented a deep q-learning network (DQN) by iteratively minimizing the loss function using the gradient to approximate the action-value function within the DQN. Ho et al. [9] ambitiously implemented a deep Q-network (DQN) for a MEC dynamic environment to simultaneously decide the optimal server selection, cooperative offloading decision and handover scheme for users' computational tasks, i.e. computed locally, on the edge and/or on cloud data servers. Interestingly, the highlight of these studies for our proposed schema was the added focus on the handover strategies and the DQN development. They utilized the energy consumption or resource allocation metric to appropriately "handover" computational tasks from an overburdened selected server to an idled or less burden server. Ho et al. also modified the conventional DQN to handle multi-dimensional continuous action spaces and to learn multiple Markov Decision Processes (MDP) simultaneously. Based on these approaches, we proposed a modification to the traditional Q-learning algorithm to handle multi-agents and update the action values similarly to a conventional deep Q-network. Instead of using a neural network, a decentralized simplistic schema could be modified to maintain constraints such as energy consumption and resource allocation. Implementation of a traditional reinforcement learning algorithm utilizes the objective functions as the rewards for each agent's respective actions and approaches. Simply negating the objective function for a single agent can be used to maximize the accumulated rewards for all agents within any system model.

III. SYSTEM MODEL & PROBLEM DEFINITION

Fig. 1: Reference state system model for the server selection schema

As an introductory review and analysis of the principal paper [1], a referenced system model was developed to simulate the static and dynamic conditions a MEC assisted server selection model could resemble. Data transmission times, power demands from transmitting the data and server/node conditions such as power consumption, current data processing and limitations were both analyzed and integrated into the proposed system model to closely parallel real world and real-time MEC network environments. Considering the transitional states in Fig. 1, content generator(s) or agents $X = \{0, \cdots, x\}$ are tasked with transmitting data with a power load from their initial position (i.e. state 0) to the closest requested viewer's base station (i.e. state 4 or terminal state). The base stations, upload nodes and download nodes represent the MEC servers along the pathway to the viewer's location. In this elementary system, we only have five transitional states, $N = \{0, 1, 2, 3, 4, \cdots, n\}$ which are scalable based on the system model.

As illustrated in Fig. 2, each state has a set of servers or actions $M = \{0, 1, 2, 3, 4, \cdots, m\}$ that an agent can choose from and select as a transitional state pathway towards the requested viewer to receive data from the content generator. Depending on the action/server selected, an associated transitional transmission time T_{nm} is assigned based on the conditions of the selected action/server m for state n transitioning to state $n + 1$. Since only one action/server can be selected per transitional state per agent, a binary action classifier $a_{nm} = \{0, 1\}$ is applied to each transmission time to denote if the affiliated action/server was selected for each transitional state.

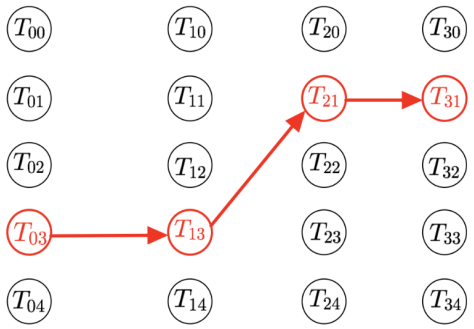


Fig. 2: Example server selection policy for the transitional states

The sum of the transitional transmission times is acquired and the objective function for each agent converges to the following:

$$T = \sum_{n \in N} a_{nm} T_{nm} \tag{1}$$

The total transmission time T in Eq (1) will scale based on each agent/content generator transmitting data to a requested number of viewers. Appropriately considering the maximum transmission time per state for a given number of agents is essential to properly model the decentralized actions/server selections of the agents and their corresponding effects on the total transmission time. Therefore, the maximum transmission time per state will be aggregated from the set of agents. The generalized objective function for a set of agents/ content generators X is as follows:

$$T = \sum_{n \in N} max\left[\tau_n\right] = \sum_{n \in N} max\left[\left\{a_{0_{nm}} T_{0_{nm}}, \cdots, a_{x_{nm}} T_{x_{nm}}\right\}_n\right] \tag{2}$$

The updated objective function (Eqn. 2) is now reflective of the choices made by all agents. The aggregated transmission times of all agents for each state τ_n is based on the associated actions/server selections by all agents. The motivation of this project is to minimize the total transmission time of this replica system while maintaining that the selected actions/servers' constraints are not violated. Such constraints

are based on the power consumption p_{nm} and quantity of data transmissions k_{nm} of a selected action/server m for each transitional state n by each agent. The finalize objective function (Eqn. 3) with the appropriate constraints are as follows:

$$\textbf{obj: } min\, T \;=\; min \sum_{n \in N} max\left[\tau_n\right] \tag{3}$$

$$\textbf{s.t.} \sum_{n \in N} \sum_{m \in M} a_{x_{nm}} = argmax(n)\colon x \in X \tag{4}$$

$$\sum_{x \in X} p_{x_{nm}} \leq P_{nm} \tag{5}$$

$$\sum_{x \in X} k_{x_{nm}} \leq K_{nm} \tag{6}$$

Eqns 4, 5 and 6 ensure that only one action/server is selected for each transitional state by every agent while maintaining the power consumption limits P_{nm} and the data transmission limits K_{nm} for each selectable action/server.

IV. PROBLEM ANALYSIS & ALGORITHMS

In order to establish a benchmark for each variation or schema of the proposed multi-agent reinforcement learning model, a linear programing algorithm (PuLP) [10] was first implemented to determine the optimal solution of the system model and compared with the solution generated by the MARL model. The conditions of each server were randomly generated to include the number of current data transmission tasks, the current power consumption load, the maximum number of data transmission tasks the server can maintain, the maximum collective power consumption the server can maintain and the server's transmission time to send data to the next transitional state. These conditions are dynamically updated based on which action/server each agent (for single and multi-agent models) selects and uses to transmit data from the content generator.

Based on the objective function, constraints and system model, a reinforcement learning algorithm can be developed to simulate the decisions or pathway data from content generators can take to reach requested viewers. The reward function for a Q-Learning algorithm, which is used for this project, can emulate the objective function with small variations if the model is a single-agent schema or a multi-agent schema. Generically, the reward function r would equate to the sum of transmission times for each transitional state. Since the transmission times are < 1, the penalty rewards would either equal -1 for single-agent models or -3 for multi-agent models. Lastly, the power consumption value for all data transmission tasks defaulted to an unitless value of 100. This parameter along with the penalty rewards are indeed tunable and the default values were determined based on the performance of the models.

Each agent simultaneously will observe the environmental state, select an action within the environment, receive an appropriate reward based on their action and transition to the next state of the environment. To summarize, the description of the state, action and rewards for the environment is below.

1) State $n \in N$: The state of the environment consists of the real-time MEC servers' conditions located at each node along the pathway from the generator to the viewer. Each state contains an array of power load, data load, power consumption and data transmission limits for all MEC servers located at that state or node. The time required to transmit data between two adjacent states is also included within the states' parameters.

2) Action $m \in M$: The available action of each given state consists of the MEC servers accessible for each agent. The action is then "selected" and represented as the boolean value stored within the binary action classifier $a_{nm} = \{0, 1\}$. The value 1 denotes the action was selected while 0 reveals the action was not selected for a given state. This notation is extended when all agents $X = \{0, \cdots, x\}$ are recorded: $a_{x_{nm}}$.

3) Reward r_n: The reward function leverages the maximum transmission time between all agents or a given action across all states within the environment.

$$r_n = -\, max\left[\left\{a_{0_{nm}} T_{0_{nm}}, \cdots, a_{x_{nm}} T_{x_{nm}}\right\}_n\right] \colon x \in X \tag{7}$$

Parameter	Value	Parameter	Value
n	$N = \{0,\ 1,\ \dots\ ,\ 4\}$	ϵ	0.10
m	$M = \{0,\ 1,\ \dots\ ,\ 6\}$	α	0.60
x	$X = \{0,\ 1,\ \dots\ ,\ 4\}$	γ	0.50
T_p	0.16	r_p	$-\,3 \cdot T_p$

Table 1: Multi-Agent Reinforcement Learning Parameters

The MARL schema derives from a single agent Q-Learning model to maximize the total reward over successive episodic iterations. Each agent tabulates their own Q-table and updates their tables after each subsequent iteration. The recursive function to update the action values $Q_{n,m}$ for each agent based on a given discrete action m and specified state n is given below.

$$Q'_{n,m} \leftarrow Q_{n,m} + \alpha \cdot \left(r_{nm} + \gamma \cdot max\, Q_{n+1,\,m} - Q_{n,m}\right) \tag{8}$$

where $Q'_{n,m}$ is the updated action value after an action is committed. The reward $r_{nm} = -\,a_{nm} T_{nm}$ for each agent defaults to the transmission time corresponding to a selected action per state. This reward parameter will change based on the system evaluated: single agent or multi agent. The parameters for the MARL model in Table 1 leveraged a 10% initial epsilon exploration factor ϵ (i.e. epsilon-greedy policy) with a running decay rate to decrease exploration over every consecutive episode. The decay rate was implemented after the first episode and plateaued after half of the target episodes

were reached (i.e. 50 episodes for all models). For the multi-agent schemas, there was a 16% penalty reward T_p (i.e. 16% of the generalized penalty reward given for unfavorable conditions that cause constraints to be violated) applied for agents who selected an identical action/server as another agent. This penalty reward was executed only if the duplicated action caused the selected server to overload and violate the constraints in the system model. The learning rate α was set to 60% while the discount factor γ remained 50% for all implementations and variations of the multi-agent reinforcement learning model. Although the Q-learning iteration for the MARL schema is generic (i.e. model-free and Temporal difference derivable), the model is heavily nested with the multi-agents action dependency of the reward function. The proposed MARL algorithm used in all models implemented in this project (i.e. single agent and multi-agent reinforcement learning based Q-learning schema) is detailed as a pseudocode on the right.

V. RESULTS & PERFORMANCE EVALUATION

Emphasized in Fig. 3, various system models ranging from single-agent systems to higher order multi-agent systems were both developed and solved using the MARL algorithm and an LP algorithm for comparison. As the number of agents increased in the system, the possible multi-agent (ma) actions increased dramatically. The action space N followed an exponential trend with respect to the number of agents X introduced to the system model:

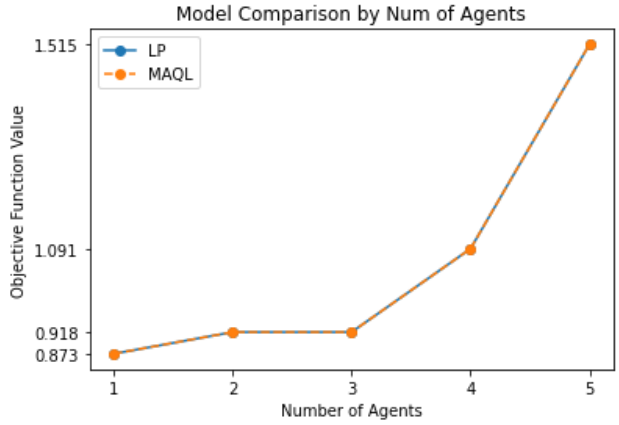
$$\#\text{ of ma-actions per state} = N^X$$

The system model generated for this project evaluation established 5 states and 6 possible actions per state available to select from. Therefore, a 2-agent system would have $36 = 6^2$ possible ma-actions available to select from since each agent could select the same action/server as each other. Following this trend, system models with large amounts of multi-agents evolved into heavily complex and computationally heavy linear programming (LP) optimization problems. Compared to the LP algorithmic challenges, the MARL algorithm easily converged to either the same optimal policy as the LP algorithm or a different policy with the identical optimal solution.

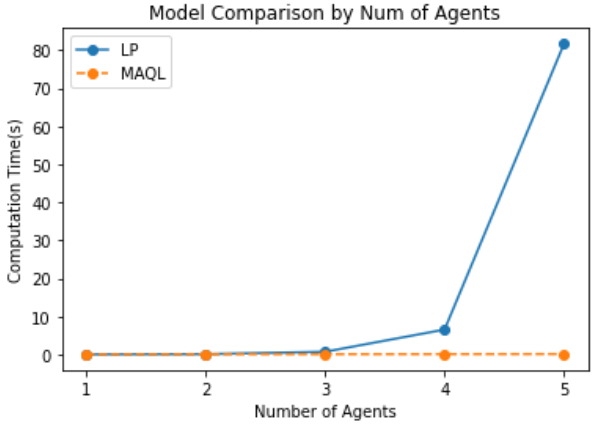
Algorithm 1: Proposed MARL algorithm

```
Initialize Q-Table for each agent and parameters
Initialize exploration decay rate and initial action space policy
for first episode to last episode do
    for state n = 0 to N do
        while all agents chose valid actions
            for agent x = 0 to X do
                choose one action from action space  m ∈ M
                store action
            compare agents actions
            update server conditions based on action space
            obtain rewards
            for agent x = 0 to X do
                update Q-Table
        store policy
    update exploration rate
    update policy
```

Although the LP algorithms used from the python package PuLP was able to solve and converge to an optimal solution for each system model, the proposed MARL algorithm greatly outperformed the LP methodologies once the action space of the multi-agent systems grew exponentially. In Fig. 3a, both the LP and MARL approaches converged to the same objective function value for every multi-agent schema while in Fig. 3b, the curves illustrate just how computationally taxing the LP method is compared to the MARL method. The LP problems can develop millions of LP binary variables which at times demanded a huge amount of computational resources and slowed the time to converge to a solution. For example, a 4-agent system model was developed with 5 states and 25^4 = 390,625 ma-actions per state. This resulted in 1.5 million variables to solve for and required over 76 minutes to produce an optimal solution from the LP algorithm while the MARL algorithm required only a fraction of a second to reach the optimal solution. Similar results were observed for other complex multi-agent system models which are not highlighted in this paper.



(a) Objective Function Values for each multi-agent schema



(b) Computation Times for each multi-agent schema

Fig. 3: Comparison curves for various multi-agent schemas with 5 states and 6 selectable actions per agent per state

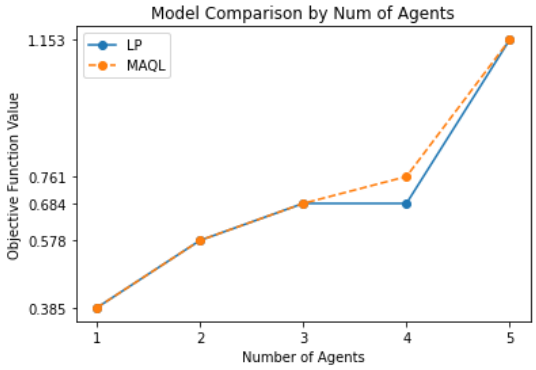


Fig. 4: Example of a system model evaluation that preferred the LP model

Despite the dramatic success of the MARL algorithm, the limitations of this approach pointed to the tunable parameters of the programmatic model. Due to the explorative nature of reinforcement learning, the MARL algorithm may cause the agents to explore too much and converge to a solution that is not optimal. The introduction of the exploration decay rate was implemented to compensate for this along with increasing the number of episodes and increasing the penalty reward for unfavorable conditions with the ma-actions. However, there are observations with server conditions, as in Fig. 4, where such adjustments were not enough to reach optimality and. The LP model in that example reached the optimal solution for every multi-agent case while the MARL model converged to a solution with increased episodes yet that solution was not optimal. Other approaches used in the references such as Actor and Critic models, Federated Learning, Neural Networks and Deep Deterministic Policy Gradient techniques may provide an optimal solution without the same limitations of the MARL algorithm. Further investigation is required to compare the performance of the MARL approach with the other models.

VI. CONCLUSION

In this project paper, a decentralized multi-agent reinforcement learning (MARL) algorithm was proposed to minimize the total transmission time required to transmit data using a mobile edge computing server network in real-time. Through the development of an objective function and associated constraints from a generalized system model, the proposed MARL algorithm was applied to numerous single-agent and multi-agent variations. The benchmark model from the linear programming approach underperformed compared to the MARL model. Optimality was achieved for all schemas through MARL and converged with the solutions of the legacy linear programming (LP) model. Although the performance of the MARL model was significant, other advanced reinforcement learning algorithms and modified neural networks may exceed the metrics of both MARL and LP schemas.

REFERENCES

[1] Huang, Siqi, et al. "Budget-aware Video Crowdsourcing at the Cloud-enhanced Mobile Edge." IEEE Transactions on Network and Service Management (2021).

[2] Zhu, Jialin, et al. "A Deep Reinforcement Learning Approach to the Flexible Flowshop Scheduling Problem with Makespan Minimization." In 2020 IEEE 9th Data Driven Control and Learning Systems Conference (DDCLS), pp. 1220-1225. IEEE, 2020.

[3] Caviglione, Gaggero, et al. "Deep reinforcement learning for multi-objective placement of virtual machines in cloud datacenters." Soft Computing (2020): 1-20.

[4] Chou, Po-Yu, et al. "Deep Reinforcement Learning for MEC Streaming with Joint User Association and Resource Management." In ICC 2020-2020 IEEE International Conference on Communications (ICC), pp. 1-7. IEEE, 2020.

[5] Mao, Hongzi, et al. "Resource management with deep reinforcement learning." In Proceedings of the 15th ACM workshop on hot topics in networks, pp. 50-56. 2016.

[6] Peng, Zhiping, et al. "A multi-objective trade-off framework for cloud resource scheduling based on the deep Q-network algorithm." Cluster Computing (2020): 1-15.

[7] Kwon, Dohyun, et al. "Multiagent DDPG-Based Deep Learning for Smart Ocean Federated Learning IoT Networks." IEEE Internet of Things Journal 7, no. 10 (2020): 9895-9903.

[8] Wei, Yifei, et al. "Deep Q-learning based computation offloading strategy for mobile edge computing." Computers, Materials and Continua 59.1 (2019): 89-104.

[9] Ho, Tai Manh, and Kim-Khoa Nguyen. "Joint server selection, cooperative offloading and handover in multi-access edge computing wireless network: A deep reinforcement learning approach." IEEE Transactions on Mobile Computing (2020).

[10] Optimization with PuLP: python package - https://coin-or.github.io/pulp/