Coordinated Load Balancing in Mobile Edge Computing Network: a Multi-Agent DRL Approach

Manyou Ma*, Di Wu*, Yi Tian Xu*, Jimmy Li*, Seowoo Jang†, Xue Liu*, Gregory Dudek*
*Samsung Electronics, Canada, †Samsung Electronics, Korea (South)
manyou.ma@partner.samsung.com, {di.wu1, yitian.xu, jimmy.li, seowoo.jang, steve.liu, greg.dudek}@samsung.com

ICC 2022 - IEEE International Conference on Communications | 978-1-5386-8347-7/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICC45855.2022.9838615

***Abstract*—Mobile edge computing (MEC) networks have been recently adopted to accommodate the fast-growing number of mobile devices performing complicated tasks with limited hardware capability. Recently, edge nodes with communication, computation, and caching capacities are starting to be deployed in MEC networks. Due to the physical separation of these resources, efficient coordination and scheduling are important for efficient resource utilization and optimal network performance. In this paper, we study mobility load balancing for communication, computation, and caching-enabled heterogeneous MEC networks. Specifically, we propose to tackle this problem via a multi-agent deep reinforcement learning-based framework. Users served by overloaded edge nodes are handed over to less loaded ones, to minimize the load in the most loaded base station in the network. In this framework, the handover decision for each user is made based on the user's own observation which comprises the user's task at hand and the load status of the MEC network. Simulation results show that our proposed multi-agent deep reinforcement learning-based approach can reduce the time-average maximum load by up to 30% and the end-to-end delay by 50% compared to baseline algorithms.**

I. Introduction and Related Work

Mobile edge computing (MEC) [1] has been proposed as one of the key enabling technologies for the fifth generation (5G) and beyond communications networks. Under the MEC framework, Internet of Things (IoT) devices with limited communication, computing, and caching (3C) capabilities are deployed to perform various tasks with stringent quality of services (QoS) requirements such as latency and throughput. To this end, edge nodes with 3C capabilities, *i.e.,* small cell base stations with local central processing units (CPUs), fronthaul connection, and file storage systems, have been deployed for the IoT devices to offload tasks and fetch popular contents. For example, in a virtual reality (VR)-based application, VR users may submit computational tasks, such as video processing, or content downloading tasks, such as movie streaming, to a MEC network. The CPUs, fronthaul links, and wireless links in the MEC network need to work in concert to handle the computational, fronthaul, and transmission loads in the network. Due to the physical separation of the resources and the coupling between the 3C components for each task, the efficient coordination and resource allocation is crucial for efficient resource utilization and satisfactory system performance in 3C-enabled MEC systems [2, 3].

In conventional multi-cell mobile wireless networks, mobility load balancing (MLB) [4] algorithms have been designed to evenly distribute user traffic across base stations. In MLB, the traffic load can be controlled by a parameter called cell individual offset (CIO). Users make handover decisions based on the relative magnitude of their channel state information (CSI) and CIO with respect to two neighboring cells, *i.e.*, identifying A3 events as detailed in 3GPP (TR36.133). Earlier work focused on using rule-based [5] methods to perform handover, while recent research has also started to employ deep reinforcement learning (DRL) to determine the CIO matrix [4]. Hierarchical [6] and transfer learning [7]-based DRL methods have also been developed and showed improved performance in terms of data throughput and load variation reduction. Fig. 1(a) shows the data flow for DRL-based MLB algorithm, where the output of the neural network corresponds to the CIO matrix.

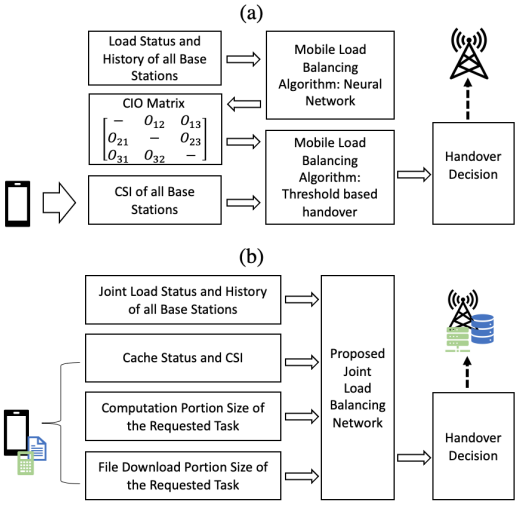

Fig. 1: Comparison of (a) CIO-based MLB approach, and (b) proposed joint load balancing approach.

Reinforcement learning has shown to be effective for different real-world applications recently [6, 7, 8, 9]. Leveraging these successes on applying DRL to MLB, in this work, we explore applying DRL to the load balancing problem in 3C-enabled MEC networks [3]. The goal is to minimize the number of backlogged jobs in the most overloaded base station, hence reducing the average end-to-end delay experienced by users in the network. This joint 3C load balancing task requires a new algorithm, since, in addition to CSI, the user association decision also depends on the caching and computational requirements of each user, making solely CIO-based algorithms restrictive.

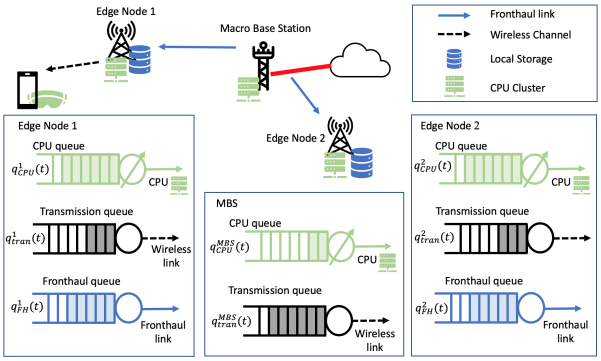

Fig. 2: Illustration of the proposed 3C-enabled MEC network with two edge nodes and one user. The user requests for a computational task for VR video processing.

To this end, in this paper, we propose to use a DRL-based algorithm that directly assigns the associated edge nodes for all users, as shown in Fig. 1(b). To accommodate the large and variable number of users in the network, we adopt a multi-agent DRL (MARL) [10]-based training approach. Separate policy networks are used to determine the base station association decision for each user request based on the 3C load components of the request and joint load status in the network. To overcome the non-stationarity problem of MARL, we adopt a parameter sharing-based scheme [11] for training. Simulation results show that the proposed MARL-based load balancing algorithm can effectively reduce the load in the most loaded base station in the network, as well as end-to-end delay in the system compared to heuristics baseline algorithms. The main contributions of our work are as follows:

- We formulate the join fronthaul, transmission, and computational load balancing problem as a Markov Decision Problem (MDP). The optimization objective is to minimize the time average maximum load in the most overloaded base station in the network.
- We propose a distributive MARL solution for the formulated problem based on parameter sharing and the proximal policy optimization (PPO) [12] algorithm.
- We perform preliminary simulation and the results show that the proposed algorithm consistently outperforms existing heuristics baselines.

II. System Model and Problem Formulation

We consider a time-slotted system with a set of T time slots, denoted by set $\mathcal{T} = \{0, 1, 2, \dots T\}$. Each time slot lasts for a duration of T_{slot}, which corresponds to multiple transmission time intervals in a standardized wireless network. User association decisions are made at the beginning of each time slot. We consider the downlink transmission in an MEC network with one macro base station (MBS) and N edge nodes, which are small cell base stations equipped with local cache and CPUs. Let us use the set $\mathcal{N} = \{1, \dots, N\}$ to denote the set of edge nodes and $\mathcal{M} = \{\text{MBS}\} \bigcup \mathcal{N}$ denote the set of all the base stations. We consider an ultra-dense network scenario, where a set of K active MEC users, denoted by $\mathcal{K} = \{1, \dots, K\}$, can be served by the MBS or any of the edge nodes in the MEC network. We consider the case where efficient frequency reuse is deployed, hence the inter-base station interference is limited. An illustration of the proposed system topology is shown in Fig. 2.

A. Channel Model

Let us define vector $\mathbf{h}_k(t) = (h_k^1(t), \dots, h_k^M(t))$, where $h_k^m(t) \in \mathbb{R}^+$ denote the channel gain between user k and base station m at time slot t, $m \in \mathcal{M}$, $k \in \mathcal{K}$, $t \in \mathcal{T}$. Given the fixed transmission power P_n, the received noise power $\sigma_{m,k}$, and system bandwidth W, the instantaneous transmission rate between base station m and user k can be expressed as

$$f_{m,k}^{\text{tran}}(t) = W \log_2 \left(1 + \frac{|h_k^m(t)|^2 P_n}{\sigma_{m,k}^2}\right), \quad \forall m, k, t. \tag{1}$$

We assume the noise power $\sigma_{m,k}^2$ is fixed and the channel gain $h_k^m(t)$ follow a random process, with the probability distribution $\mathbb{P}(h_k^m(t))$.

B. User Request Model

Let the random variable $r_k^{\text{stat}}(t) \in \{0, 1, 2\}$ denote the *request status* from user k at time slot $t \in \mathcal{T}$. At time slot t, let $r_k^{\text{stat}}(t) = 1$ denote the case where user k requests for a *file downloading task*, $r_k^{\text{stat}}(t) = 2$ denote the case where user k requests for *computational task*, and $r_k^{\text{stat}}(t) = 0$ denote the case where user k does not have any request. We assume that $r_k^{\text{stat}}(t)$ follows a stochastic process, with the probability distribution[1]

$$\mathbb{P}(r_k^{\text{stat}}(t)) = \lambda^{\text{file}} \mathbf{I}(r_k^{\text{stat}}(t) = 1) + \lambda^{\text{comp}} \mathbf{I}(r_k^{\text{stat}}(t) = 2), \forall k, t,$$

where $\mathbf{I}(\cdot)$ stands for the indicator function, and λ^{file} and λ^{comp} denote the task arrival rates for file downloading tasks and computational tasks.

Let the random vector $\mathbf{r}_k^{\text{size}}(t) = (r_k^{\text{file}}(t), r_k^{\text{comp}}(t))$ denote the size of the request made by user $k \in \mathcal{K}$ at time slot $t \in \mathcal{T}$. For a file downloading task, let $r_k^{\text{file}}(t)$ denote the size of the requested file; while for a computational task, let $r_k^{\text{file}}(t)$ denote the size of the solution to the computational task. Furthermore, let $r_k^{\text{comp}}(t)$ denote the number CPU cycles required for completing the computational task. We assume $\mathbf{r}_k^{\text{size}}(t)$ follow a random process, with probability distribution

$$\mathbb{P}(r_k^{\text{file}}(t) \mid r_k^{\text{stat}}(t)) = \frac{\mathbf{I}\left(r_k^{\text{file}}(t) \in [r_{\min}^{\text{file}}, r_{\max}^{\text{file}}]\right) \mathbf{I}(r_k^{\text{stat}}(t) = 1)}{r_{\max}^{\text{file}} - r_{\min}^{\text{file}}} + \frac{\mathbf{I}\left(r_k^{\text{file}}(t) \in [r_{\min}^{\text{sol}}, r_{\max}^{\text{sol}}]\right) \mathbf{I}(r_k^{\text{stat}}(t) = 2)}{r_{\max}^{\text{sol}} - r_{\min}^{\text{sol}}},$$

$$\mathbb{P}(r_k^{\text{comp}}(t) \mid r_k^{\text{stat}}(t)) = \frac{\mathbf{I}\left(r_k^{\text{comp}}(t) \in [r_{\min}^{\text{comp}}, r_{\max}^{\text{comp}}]\right) \mathbf{I}(r_k^{\text{stat}}(t) = 2)}{r_{\max}^{\text{comp}} - r_{\min}^{\text{comp}}}.$$

That is, $r_k^{\text{file}}(t)$ and $r_k^{\text{comp}}(t)$ follow uniform distributions within bounds defined by $r_{\min}^{\text{file}}$, $r_{\max}^{\text{file}}$, $r_{\min}^{\text{sol}}$, $r_{\max}^{\text{sol}}$, $r_{\min}^{\text{comp}}$, and $r_{\max}^{\text{comp}}$.

[1]In this section, we detail the system model that we tested in Sec. IV. In practice, since the MARL algorithm we propose is model-free, it can also be deployed in system models with other probability distributions.

C. User Association Decision

At time slot $t \in \mathcal{T}$, each active users $k \in \mathcal{K}_{\text{active}} = \{k \in \mathcal{K} \mid r_k^{\text{stat}}(t) > 0\}$ needs to be served by one of the base stations $m \in \mathcal{M}$. Let $u_k(t) \in \mathcal{M}$ indicate the user association decision for $k \in \mathcal{K}$ at time slot $t \in \mathcal{T}$.

D. MBS and Edge Node Model

1) HetNet Model: The MBS is connected to the cloud via high-speed fibre connection and can fetch contents requested by the users. Each edge nodes is equipped with a local storage with finite capacity, where a subset of the contents that might be requested by users being cached beforehand. A microwave fronthaul [13] between the edge nodes and the MBS can be used to fetch requested files that are not being cached in the edge nodes. Let f_{FH}^n denote the fronthaul capacity, in terms of transmission rate, of edge node $n \in \mathcal{N}$. Let f_{comp}^m denote the computing capacity of base station m, in terms of CPU cycle per time slot, $m \in \mathcal{M}$. To accommodate bursty traffic and overloaded system scenarios, buffers are installed in the base stations, where incoming tasks for the fronthaul, CPU, and wireless channel are first placed in the *fronthaul queue*, *CPU queue*, and *transmission queue*, respectively, and later being executed in order.

2) Cache Model: At time slot $t \in \mathcal{T}$, let us use a binary *cache status* vector $\boldsymbol{\delta}_k(t) = (\delta_k^{\text{MBS}}(t), \delta_k^1(t), \ldots, \delta_k^N(t))$ to indicate whether the content requested by user k is being cached in the edge nodes, where $\delta_k^m(t) = 1$ when content requested by user k is being cached in base station m, and $\delta_k^m(t) = 0$ denote the case otherwise, $k \in \mathcal{K}$, $m \in \mathcal{M}$, and $t \in \mathcal{T}$. We assume $\boldsymbol{\delta}_k(t)$ follow a stochastic process, with distribution

$$\mathbb{P}(\delta_k^m(t)) = \delta_{\text{hit}}^m \mathbf{I}(\delta_k^m(t) = 1), \forall k, m, t, \tag{2}$$

where δ_{hit}^m corresponds to the *cache hit rate* at edge node m, $m \in \mathcal{M}$. Since the MBS can access all the contents in the cloud, we have $\delta_{\text{hit}}^{\text{MBS}} = 1$.

3) Example Data Flow: Here, we introduce the data flow for the task requested by user $k \in \mathcal{K}_{\text{active}}$ associated with base station $m \in \mathcal{M}$ at time slot $t \in \mathcal{T}$, depending on the request status $r_k^{\text{stat}}(t)$ and the cache status $\delta_k^m(t)$. Three scenarios are possible in this paper:

- Case 1 ($r_k^{\text{stat}}(t) = 1$ and $\delta_k^n(t) = 1$): In this case, since the file being requested is being cached in the serving edge node n, the file will only be placed in the transmission queue.
- Case 2 ($r_k^{\text{stat}}(t) = 1$ and $\delta_k^n(t) = 0$): In this case, the file requested is not being cached in the serving edge node n, it needs to be fetched from the MBS through the fronthaul. The task is placed in both the fronthaul queue and the transmission queue. After the requested file is being fetched, transmission to user k will start.
- Case 3 ($r_k^{\text{stat}}(t) = 2$): In this case, the computational portion of the task will be placed in the fronthaul queue and a placeholder for the solution will be placed in the transmission queue. After the computational portion of the task is computed the solution will be transmitted to user k.

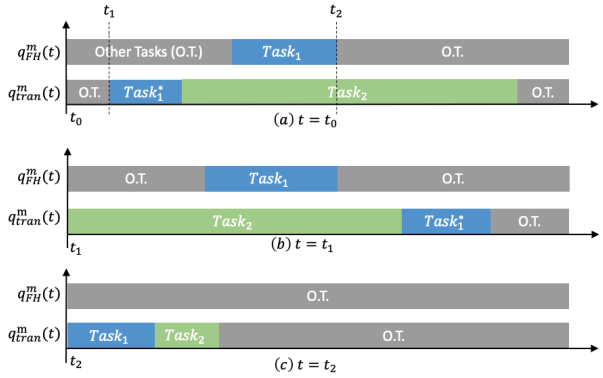

Fig. 3: Illustration of the preemptive queuing order adopted for the transmission queue.

4) Queues and Load: Let the *fronthaul load* at base station m at time slot t, $q_{\text{FH}}^n(t) \in \mathbb{R}^+$ denote the time it will take for base station $m \in \mathcal{M}$ to fetch all the queued content requests at time slot $t \in \mathcal{T}$. To simplify notations, we also define a fronthaul load $q_{\text{FH}}^{\text{MBS}}(t)$ at the MBS, where $q_{\text{FH}}^{\text{MBS}}(t) = 0, t \in \mathcal{T}$. Let the *CPU load* at base stations m, $q_{\text{CPU}}^m(t) \in \mathbb{R}^+$ to denote the time it takes for the CPU to finish all the backlogged tasks at time slot t. Since future CSI in the system is uncertain, let the *transmission load* of base station m at time slot t, $q_{\text{tran}}^m(t) \in \mathbb{R}^+$ denote the time estimate for a base station m to finish transmitting all the pending computation solutions and contents to the users at time slot t, $m \in \mathcal{M}$, $t \in \mathcal{T}$. The fronthaul, CPU, and transmission load at base station m can also be thought of as the *queue length* of the fronthaul, CPU, and transmission queues, respectively. We will use these notations interchangeably.

We used $L^m(t) = \max(q_{\text{FH}}^m(t), q_{\text{CPU}}^m(t), q_{\text{tran}}^m(t))$ to estimate the amount of time that a base station m needs to complete all its backlogged tasks at time slot t. We denote $L^m(t)$ as the *load* of base station m at time slot t, $m \in \mathcal{M}, t \in \mathcal{T}$.

5) Queue Dynamics: We assume that the tasks located in the fronthaul and CPU queues are executed in a first-come first-served (FCFS) manner. Given the fronthaul queue length at t, $q_{\text{FH}}^m(t)$, and the amount of new data that edge node m needs to fetch content for all users $\mathcal{K}$, $\Delta^{\text{FH}}(t) = \sum_{k \in \mathcal{K}} r_k^{\text{file}}(t) \mathbf{I}(u_k(t) = n) \mathbf{I}(r_k^{\text{stat}}(t) = 1)(1 - \delta_n^k(t))$, $q_{\text{FH}}^m(t+1)$ is a deterministic value, where

$$q_{\text{FH}}^m(t+1) = \max(q_{\text{FH}}^m(t) - 1, 0) + \frac{\Delta_k^{\text{FH}}(t)}{f_{\text{FH}}^m}, n \in \mathcal{N}. \tag{3}$$

The dynamics of the queue lengths of the CPU queues can be expressed in a similar manner, while the dynamics of the queue lengths of the transmission queues cannot be expressed as a deterministic expression. Given the queue length of the transmission queue at base station m at time slot $q_{\text{tran}}^m(t)$, and the newly arrived tasks, we assume $q_{\text{tran}}^m(t+1)$ to be a random process, following the probability distribution $\mathbb{P}(q_{\text{tran}}^m(t+1) \mid q_{\text{tran}}^m(t), \mathbf{H})$, where $\mathbf{H}$ corresponds to the combination of the historical and current values of the afore-

mentioned random variables. Due to the inter-dependencies between the fronthaul queue and transmission queue, user requests are not necessarily executed FCFS in the transmission queue. Consider two tasks, Task_1 and Task_2 that arrive in order. Task_1 requires both fetching data from fronthaul and data transmission, while Task_2 only requires data transmission. At time t_1, as shown in Fig. 3(a), when other tasks in the transmission queue are completed, the task at the head-of-line is Task_1. However, since Task_1's fetching data from fronthaul portion is not completed yet, its transmission portion cannot start immediately. In this case, Task_2's data transmission portion will start first, as shown in Fig. 3(b). However, once Task_1's required content is fetched from the MBS, the execution of Task_2 will pause to first serve Task_1 preemptively, as shown in Fig. 3(c).

At the beginning of time slot $t \in \mathcal{T}$, each base station $m \in \mathcal{M}$ will share, through broadcasting on the control channel, the load of all its fronthaul, CPU, and transmission queues. We summarize the load status of all the base stations in a vector

$$\begin{aligned}\mathbf{q}(t) =&(q_{\text{FH}}^{\text{MBS}}, q_{\text{CPU}}^{\text{MBS}}, q_{\text{FH}}^{\text{tran}}, q_{\text{FH}}^{1}, q_{\text{CPU}}^{1}, q_{\text{tran}}^{1}, \ldots,\\ &q_{\text{FH}}^{\text{N}}, q_{\text{CPU}}^{\text{N}}, q_{\text{tran}}^{\text{N}}), t \in \mathcal{T}.\end{aligned} \tag{4}$$

E. Load Balancing Objective

The objective of our joint load balancing problem is to distribute the 3C load in the MEC networks evenly among the base stations. We study a tractable variant of this problem by minimizing the load in the base station that is the most loaded. We define the maximum load among all the fronthauls, CPUs, and wireless link, as the maximum load $L(t)$ in the network. That is,

$$L(t) = \max_{m \in \mathcal{M}} L^m(t). \tag{5}$$

In the following, we will formulate the problem of minimizing the time-averaged maximum load in an MEC network as a MDP and present our proposed solution based on MARL.

F. Dec-POMDP Problem Formulation

To alleviate the signaling overhead and large state and action space of centralized scheduling algorithms. In this paper, we consider a decentralized user association framework. At the beginning of the each time slot $t \in \mathcal{T}$, the association decision for user $k \in \mathcal{K}$ is made based on the current load status of the MEC network, $\mathbf{q}(t)$, and the user's request $\mathbf{r}_k(t) = (\mathbf{h}_k(t), r_k^{\text{stat}}(t), \mathbf{r}_k^{\text{size}}(t), \boldsymbol{\delta}_k(t))$, $k \in \mathcal{K}$, $t \in \mathcal{T}$. The decision-making module for each user is defined as an *agent*, which can either locate on the users' device, or in the decision-making module of the MEC network. Hence, in the decentralized user scheduling framework we consider, a set of $\mathcal{N} = \{1, \ldots, N\}$ agents cooperatively attempt to minimize a global cost, which corresponds to the time-averaged maximum load in the system. In this subsection, we formulate the joint load balancing problem as a decentralized partially observable Markov decision process (Dec-POMDP) [11] problem, which is a framework for cooperative sequential decision making under uncertainty.

1) Decision Epoch and Discount Factor: We use each discrete time slot as a decision epoch for the formulated Dec-POMDP problem, hence the set of decision epochs can be represented as $\mathcal{T}$. We consider a discount cost scenario with discount factor γ.

2) States: In decision epoch $t \in \mathcal{T}$, the state of the Dec-POMDP is the concatenation of all the queues in the network and information about the new requests from users

$$\mathbf{s}(t) = (\mathbf{q}(t), \mathbf{r}_1(t), \ldots, \mathbf{r}_K(t)) \in \mathcal{S}. \tag{6}$$

3) Observations: In decision epoch $t \in \mathcal{T}$, the observation of agent $k \in \mathcal{K}$ is chosen as

$$\mathbf{o}_k(t) = (\mathbf{q}(t), \mathbf{r}_k(t)) \in \mathcal{O}, \quad \forall k, t. \tag{7}$$

Let $Z_k(\cdot) : \mathcal{S} \mapsto \mathcal{O}$ denote the function that maps state of the network $\mathbf{s}(t)$ to the observation $\mathbf{o}_k(t)$ of agent $k \in \mathcal{K}$.

4) Actions: In decision epoch $t \in \mathcal{T}$, agent k chooses the association action $u_k(t) \in \mathcal{M}$ for user k. The joint action in decision epoch t can represented as $\mathbf{u}(t) = (u_1(t), \ldots, u_K(t)) \in \mathcal{M}^K$, $t \in \mathcal{T}$.

5) Cost: In decision epoch t, the cost corresponds to the maximum load in the MEC network, where

$$c\left(\mathbf{s}(t), \mathbf{u}(t)\right) = L(t). \tag{8}$$

6) Policy: A control policy for agent $k \in \mathcal{K}$, $\pi_k(\cdot) : \mathcal{O} \mapsto \mathcal{M}$, maps the observation $\mathbf{o}_k(t)$ of agent k to an association action $u_k(t)$.

7) State Transition Probability: The joint state transition probability of the Dec-POMDP problem depends on the aforementioned probability distributions of the random variables in the system $\mathbb{P} = \mathbb{P}(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{u}(t))$.

In summary, the Dec-POMDP problem is described as the 8-tuple $(\mathcal{K}, \mathcal{S}, \mathcal{M}, \mathbb{P}, c(\cdot), \gamma, \mathcal{O}, Z_1(\cdot) \times \ldots \times Z_K(\cdot))$

G. Optimal Decentralized Policy

In our paper, the joint load balancing problem aims to find the optimal stationary decentralized policy $\pi^* = (\pi_1^*, \ldots, \pi_K^*)$, where

$$\pi^* = \arg\min_{\pi} \sum_{t=1}^{T} \gamma^t \mathbb{E}\left[c(\mathbf{s}(t), \pi_1(\mathbf{o}_1(t)), \ldots, \pi_K(\mathbf{o}_K(t))\right].$$

In the next section, we will introduce our proposed MARL algorithm on finding the optimal policy π^*.

III. Parameter Sharing-based MARL-based Solution

In this paper, a parameter sharing-based MARL [11] framework is adopted, under which all the agents share the same policy parameters $\boldsymbol{\theta}$ and value function parameters $\boldsymbol{\phi}$. It has been reported to have good training efficiency when the agents in the system are homogenous, which is the case for the agents in our formulated Dec-POMDP problem. We assume the readers are familiar with the basic actor-critic DRL training framework [14]. Let $\pi_{\boldsymbol{\theta}}(u_k(t)|\hat{\mathbf{o}}_k(t))$ denote the policy parameterize by parameters $\boldsymbol{\theta}$ and $v_{\boldsymbol{\phi}}(\hat{\mathbf{o}}_k(t))$ denote

Algorithm 1 PS-PPO-based Joint Load Balancing Algorithm (please refer to [11, 12] for more implementation details)

Input: Initial policy network parameters $\boldsymbol{\theta}_0$ and value network parameters $\boldsymbol{\phi}_0$
for $i = 0, \ldots, N_{\text{iter}}$ **do**
  for $k \in \mathcal{K}$ **do**
    Collect set of J trajectories using $\pi_{\boldsymbol{\theta}_i}(u_k(t)|\hat{\mathbf{o}}_k(t))$
    Estimate the advantage function $A_{\boldsymbol{\theta}_i}(\hat{\mathbf{o}}_k^j(t), u_k^j(t))$
  end for
  Update policy network parameter $\boldsymbol{\theta}$ by (9)
  Update value network parameter $\boldsymbol{\phi}$ by (11)
end for

the value function parameterized by parameters $\boldsymbol{\phi}$. The index of each agent is appended into the observation space, where $\hat{\mathbf{o}}_k(t) = (\mathbf{o}_k(t), k)$, $k \in \mathcal{K}$, to ensure that different agents may adopt different actions under the same observation.

In parameter sharing-based MARL scheme, centralized training and decentralized execution are adopted, and many single agent DRL methods can be chosen to update the policy network. In this paper, we adopt the PPO [12] method, due to its robustness and simplicity. In Algorithm 1, we describe the MARL algorithm we designed that combines parameter sharing and PPO (PS-PPO). The algorithm first initializes the policy network parameters $\boldsymbol{\theta}_0$ and value network parameters $\boldsymbol{\phi}_0$. Afterwards, at iteration i, all agents jointly roll out J trajectories, $\{\tau_k^1, \ldots, \tau_k^J\}$, where $\tau_k^j = \{\mathbf{s}^j(1), \mathbf{u}^j(1), \ldots, \mathbf{s}^j(T), \mathbf{u}^j(T)\}$ for T time steps using policy $\pi_{\boldsymbol{\theta}_i}(u_k(t)|\hat{\mathbf{o}}_k(t))$. Then, the advantage function for each time step $A_{\boldsymbol{\theta}}(\hat{\mathbf{o}}_k(t), u_k(t))$ is estimated by taking the difference of the cost-to-go function $\hat{C}_k^j(t) = \frac{1}{K}\sum_{m=t+1}^{T} c(\mathbf{s}^j(m), \mathbf{u}^j(m))$ and the value function $v_{\boldsymbol{\phi}_i}(\hat{\mathbf{o}}_k(t))$. Then, the policy network parameters are updated by jointly optimizing the PPO-Clip objective for all agents

$$\boldsymbol{\theta}_{i+1} = \arg\min_{\boldsymbol{\theta}} \sum_{k=1}^{K}\sum_{j=1}^{J}\sum_{t=1}^{T} \min\left(\frac{\pi_{\boldsymbol{\theta}}(u_k^j(t)|\hat{\mathbf{o}}_k^j(t))}{\pi_{\boldsymbol{\theta}_i}(u_k^j(t)|\hat{\mathbf{o}}_k^j(t))} A_{\boldsymbol{\theta}_i}(\hat{\mathbf{o}}_k^j(t), u_k^j(t)), g(\epsilon, A_{\boldsymbol{\theta}_i}(\hat{\mathbf{o}}_k^j(t), u_k^j(t)))\right)^2, \tag{9}$$

where

$$g(\epsilon, A) = [\mathbf{I}(A \geq 0)(1+\epsilon) + \mathbf{I}(A < 0)(1-\epsilon)]A. \tag{10}$$

The value network parameters are updated by

$$\boldsymbol{\phi}_{i+1} = \arg\max_{\boldsymbol{\phi}} \sum_{k=1}^{K}\sum_{j=1}^{J}\sum_{t=1}^{T} \left(v_{\boldsymbol{\phi}}(\hat{\mathbf{o}}_k^j(t)) - \hat{C}_k^j(t)\right)^2 \tag{11}$$

We adopt fully connected neural networks for both the policy and value networks. Gradient descent is implemented using the Adam optimizer. The learning rate is set to α.

IV. Simulation Results

In this paper, the network parameter and scenario described in 3GPP standard (TR36.839) is used. Specifically, we consider a network topology that is a square with the diameter 100 m, with one MBS and $N = 4$ edge nodes located in the centre of the network. The edge nodes are spaced apart by 20 m. The carrier frequency of the network considered is $f_c = 2.0$ GHz, while the bandwidth considered is 10 MHz. The transmission power of the base station is set to 46 dBm and the pathloss is set as $L_k^{\text{MBS}}(t) = 128.1 + 37.6\log(R_k^{\text{MBS}}(t))$. The transmission power of the edge nodes are set to 30 dBm and the pathloss is set to $L_k^n(t) = 140.7 + 36.7\log(R_k^n(t))$, where $R_k^m(t)$ is the distance between the user k and the base station $m \in \mathcal{M}$ in kilometers. Unless otherwise specified, we consider a $N = 16$ potential mobile users, each user uniformly moves in fixed random directions with wrap-around in the area considered with the speed $v = 3$ km/h. In Table I, we summarize the default parameters we adopted in this paper. Deep learning programming framework PyTorch was used to implement the programming code, and the simulations were conducted on an Nvidia GTX2060 GPU (Santa Clara, CA). We compare the performance of the proposed PS-PPO-based **MARL** algorithm with three heuristic baselines:

- Random Association (**RA**): Each user is associated randomly to one of the base station.
- Greedy Scheduling (**Greedy**): Each user is associated to the base station with the shortest queue.
- CSI-based Scheduling (**MLB**): Each user is associated to the base station that has the best CSI. Since we do not consider handover cost in this paper, it is equivalent to the CSI-based MLB algorithm with a zero CIO matrix.

In Fig. 4, we show the episode cost of the proposed PS-PPO algorithm for different number of users. We also include the aggregate episode cost for the random association baseline for

TABLE I: Parameters Adopted in this Paper [15].

Parameter	Value
T_{iter}, T, N, K	1 [s], 100, 4, 16
λ^{file}, λ^{comp}	0.4, same as λ^{file}
$r_{\min}^{\text{file}}$, $r_{\max}^{\text{file}}$, $r_{\min}^{\text{sol}}$, $r_{\max}^{\text{sol}}$	20, 50, 20, 50 [Mbits]
$r_{\min}^{\text{comp}}$, $r_{\max}^{\text{comp}}$	10, 30 [megacycles]
f_{FH}^n, f_{comp}^n,	14 [Mbps], 41.8 [GHz]
δ_{hit}^m, ϵ, γ, α,	0.3, 0.05, 0.995, 0.00001

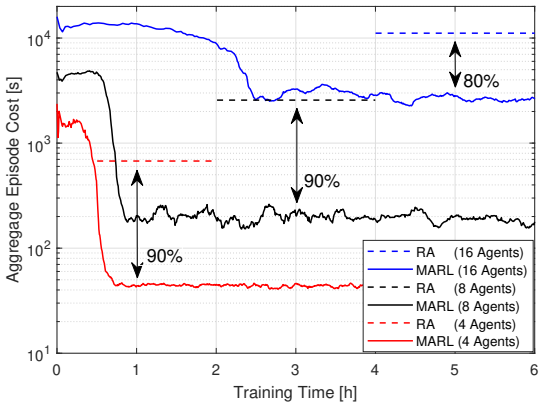

Fig. 4: Episode reward during training, when 4, 8, and 16 users are present in the system. Performance of the RA baseline is shown for comparison.

comparison. We observe that training converged in all cases, and the training time is longer for networks with larger number of agents. In all cases, the MARL algorithm outperformed the RA baseline by reducing the maximum load in the network up to 80%. This shows the effectiveness of the training scheme.

In Fig. 5, we compare the the aggregate load in the most overloaded base station of the proposed MARL algorithm with respect to the baselines, as a function of the sum task arrival rate ($\lambda^{\text{file}} + \lambda^{\text{comp}}$). We can observe that the proposed algorithm outperforms the baselines in all cases, where the improvement is more prominent when the task arrival rate is higher. When the task arrival rate is low, the proposed MARL scheme achieved comparable performance with the MLB heuristic. When the sum task arrival rate is high, up to 30% reduction in the load can be achieved. In Fig. 6, we compare the end-to-end delay experienced by users in the MEC network achieved by the proposed MARL algorithm with respect to the baseline. 100 data samples are collected. The boxplot of the number of user requests being served using the proposed algorithm and baselines is presented in Fig. 6(a). The proposed MARL and Greedy methods outperform the RA and MLB baselines by a large margin. In Fig. 6(b), we show the average end-to-end delay experienced by all user requests being served. The MARL algorithm achieved around 50% reduction in average end-to-end delay compared to the Greedy scheduling heuristic[2].

V. Conclusion

In this paper, we studied the joint optimization for communication, computation, and caching load balancing for 3C-enabled MEC networks. Specifically, we proposed a distributive user association and handover algorithm, where handover decisions are made by jointly considering the user request and the system load status. First, we formulated the problem as a Dec-POMDP problem, and used the time-averaged load in the most overloaded base station in the MEC network as the objective function. Second, we adopted a PS-PPO-based MARL algorithm to obtain a suboptimal solution to the Dec-POMDP problem. Preliminary results show that the proposed algorithm can reduce the load in the most overloaded base station, compared to three heuristic baselines. Moreover, we showed that the average end-to-end delay experienced by all users using the proposed MARL approach is also reduced. In the future, we plan to extend this study to consider changing the number of active users and variable task arrival rates.

References

[1] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2322–2358, 2017.

[2] M. Tang, L. Gao, and J. Huang, "Communication, computation, and caching resource sharing for the internet of things," *IEEE Commun. Mag.*, vol. 58, no. 4, pp. 75–80, 2020.

[2]In the literature, when end-to-end delay is used as the optimization objective, unfinished tasks at T are considered *dropped* [15], and a penalty is introduced to discourage dropping packets. Since this penalty is not considered in our work, we only compare the end-to-end delay of MARL and Greedy algorithms, which achieved comparable number of finished tasks.

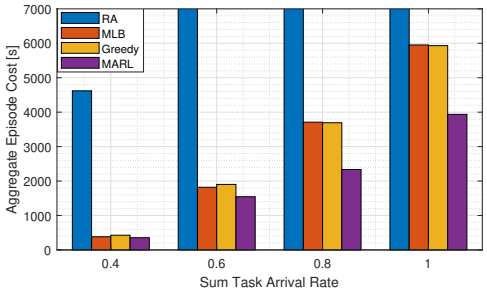

Fig. 5: Comparison of the aggregate episode cost achieved by the proposed MARL algorithm with respect to the RA, MLB, and Greedy baselines

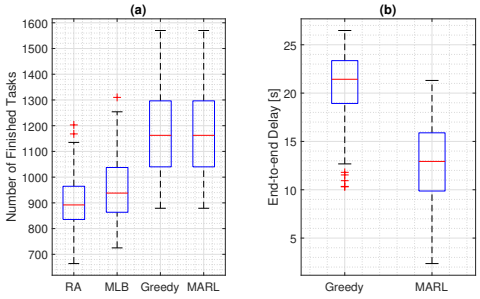

Fig. 6: Comparison of (a) the aggregate number of tasks completed, and (b) average end-to-end delay achieved by the proposed MARL algorithm with respect to the RA, MLB, and Greedy baselines.

[3] A. Ndikumana, N. H. Tran, T. M. Ho, Z. Han, W. Saad, D. Niyato, and C. S. Hong, "Joint communication, computation, caching, and control in big data multi-access edge computing," *IEEE Trans. Mobile Comput.*, vol. 19, no. 6, pp. 1359–1374, 2020.

[4] Y. Xu, W. Xu, Z. Wang, J. Lin, and S. Cui, "Load balancing for ultradense networks: A deep reinforcement learning-based approach," *IEEE Internet of Things Journal*, vol. 6, no. 6, pp. 9399–9412, 2019.

[5] R. Kwan, R. Arnott, R. Paterson, R. Trivisonno, and M. Kubota, "On mobility load balancing for LTE systems," in *Proc. IEEE Vehicular Technology Conference (VTC-Fall)*, Ottawa, Canada, Sep. 2010.

[6] J. Kang, X. Chen, D. Wu, Y. T. Xu, X. Liu, G. Dudek, T. Lee, and I. Park, "Hierarchical policy learning for hybrid communication load balancing," in *Proc. of IEEE Int'l Conf. on Commun. (ICC)*, Montreal, Canada, Jun. 2021.

[7] D. Wu, J. Kang, Y. T. Xu, H. Li, J. Li, X. Chen, D. Rivkin, M. Jenkin, T. Lee, I. Park, X. Liu, and G. Dudek, "Load balancing for communication networks via data-efficient deep reinforcement learning," in *Proc. of IEEE Global Commun. Conf.*, Madrid, Spain, Dec. 2021.

[8] Q. Dang, D. Wu, and B. Boulet, "Ev charging management with ann-based electricity price forecasting," in *2020 IEEE Transportation Electrification Conference & Expo (ITEC)*. IEEE, 2020, pp. 626–630.

[9] D. Wu, *Machine learning algorithms and applications for sustainable smart grid*. McGill University (Canada), 2018.

[10] T. Li, K. Zhu, N. C. Luong, D. Niyato, Q. Wu, Y. Zhang, and B. Chen, "Applications of multi-agent reinforcement learning in future internet: A comprehensive survey," 2021.

[11] J. K. Gupta, M. Egorov, and M. Kochenderfer, "Cooperative multi-agent control using deep reinforcement learning," in *Proc. Int'l Conf. on Autonomous Agents and Multiagent Systems*, San Paulo, Brazil, May 2017, pp. 66–83.

[12] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017.

[13] "Microwave: perfect for fixed wireless access backhaul," Ericsson, Tech. Rep., Oct. 2021. [Online]. Available: ericsson.com/microwave-outlook

[14] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. Cambridge, MA: MIT press, 2018.

[15] M. Tang and V. W. Wong, "Deep reinforcement learning for task offloading in mobile edge computing systems," *IEEE Trans. Mobile Comput.*, 2020 (Early Access).