Multi-Agent Reinforcement Learning Based Coded Computation for Mobile Ad Hoc Computing

Baoqian Wang[1] Junfei Xie[2] Kejie Lu[3] Yan Wan[4] Shengli Fu[5]

ICC 2021 - IEEE International Conference on Communications | 978-1-7281-7122-7/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICC42927.2021.9500600

***Abstract*—Mobile ad hoc computing (MAHC), which allows mobile devices to directly share their computing resources, is a promising solution to address the growing demands for computing resources required by mobile devices. However, offloading a computation task from a mobile device to other mobile devices is a challenging task due to frequent topology changes and link failures because of node mobility, unstable and unknown communication environments, and the heterogeneous nature of these devices. To address these challenges, in this paper, we introduce a novel coded computation scheme based on multi-agent reinforcement learning (MARL), which has many promising features such as adaptability to network changes, high efficiency and robustness to uncertain system disturbances, consideration of node heterogeneity, and decentralized load allocation. Comprehensive simulation studies demonstrate that the proposed approach can outperform state-of-the-art distributed computing schemes.**

I. Introduction

In recent years, we have witnessed the significant growth of mobile applications that are based on advanced artificial intelligence and deep-learning models, such as face recognition, object detection and natural language processing. These applications stimulate the demands for greater computing resources, which may be difficult to support in one device. To address this need, one promising solution is to offload computation-intensive tasks from resource-limited mobile devices to remote clouds or nearby edge computing facilities [1]. Nevertheless, this solution can suffer from high transmission delays and becomes infeasible when there are no communication infrastructures nearby, e.g., in rural areas or emergencies. Local computing is thus essential for delay-sensitive applications, or when there is no or weak wireless Internet connection to remote clouds and when nearby edge computing devices are not available.

Mobile ad hoc computing (MAHC) [2], [3] is coined recently to provide local computing services, by enabling resource sharing among mobile devices. However, the research on MAHC is still in its infancy. Existing studies have mainly considered smartphones [4] or ground vehicles [5] as the main resources. A few recent studies attempted to move computing resources up to the unmanned aerial vehicles (UAVs) and create a networked airborne computing system [6], [7] that allows UAVs to share resources between each other through direct flight-to-flight communications. This can be considered as one type of MAHC, but studies along this direction are still very limited.

Offloading a computation task from a mobile device to other mobile devices nearby is challenging, due to the mobility and heterogeneity of these devices. Particularly, devices' movement can cause frequent topology changes and communication instability/failures. Furthermore, mobile devices may not have an explicit knowledge of its communication environment and the computing powers of other devices. In this paper, we aim to address these challenges, by exploring coded computation [8] and multi-agent reinforcement learning (MARL) [9].

Coded computation is a promising technique for improving the resilience of distributed computing systems to uncertain system disturbances, such as communication bottlenecks, node/link failures, slow-downs of computing nodes, etc. Different coded computation tasks have been explored for systems with homogeneous computing nodes, such as matrix-vector multiplication [10], gradient descent [11], multivariate polynomials [12], etc. Coded computation for heterogeneous systems has also been recently investigated. For example, [13] introduced a *Heterogeneous Coded Matrix Multiplication* (HCMM) scheme that maximizes the expected computing results received by the master node. Our research group developed a batch processing based coded computation (BPCC) scheme [14], [15], which allows partial results to be returned and is highly efficient and robust to uncertain system disturbances. However, all these works assume static networks and adopt simplified communication and computing models. Recently, a coded cooperative computation protocol (C3P) [16] was proposed that considers mobile computing systems, and allocates tasks in a dynamic and adaptive manner based on the time lag between two consecutive tasks. However, this method requires explicit knowledge of the delay model.

MARL has been widely used for applications that involve interactions among multiple agents, such as multi-robot control, multiplayer games, Internet of Things, to name a few. Although multiple studies apply RL/MARL to solve computation offloading or resource allocation problems, e.g., [17], [18], the use of MARL for MAHC, especially with coded computation, has not been investigated.

In this paper, we consider distributed computing over a MAHC system that consists of multiple mobile and heterogeneous computing devices, with unknown computing power and communication environment. As a preliminary

Baoqian Wang is with the Department of Electrical and Computer Engineering, University of California, San Diego, and San Diego State University, San Diego, CA, 92182 (e-mail: bawang@ucsd.edu).

Junfei Xie is with the Department of Electrical and Computer Engineering, San Diego State University, San Diego, CA, 92182 (e-mail: jxie4@sdsu.edu). Corresponding author.

Kejie Lu is with the Department of Computer Science and Engineering, University of Puerto Rico at Mayagüez, Mayagüez, Puerto Rico, 00681, e-mail: (kejie.lu@upr.edu).

Yan Wan is with the Department of Electrical Engineering, University of Texas at Arlington, Arlington, Texas, 76019 (e-mail: yan.wan@uta.edu).

Shengli Fu is with the Department of Electrical Engineering, University of North Texas, Denton, Texas, 76201 (e-mail: Shengli.Fu@unt.edu).

investigation, we focus on a classical distributed computing task, the matrix-vector multiplication problem, which is a building block of many computation tasks and machine learning algorithms. Based on this task, our *main contribution* is an innovative MARL-based coded computation framework with the following key features: 1) adaptability to time-varying communication changes caused by node mobility; 2) capability to address node heterogeneity; 3) applicability for any MAHC or mobile edge/fog computing (MEC) systems without having to know their communication or computing characteristics; 4) high computational efficiency by applying the multi-agent deep deterministic policy gradient (MADDPG) [19], the state-of-the-art MARL algorithm, and adopting the batch processing procedure proposed in our previous studies [14], [15]; and 5) decentralized load allocation that relieves the computation burden at the master node. To demonstrate these promising features, we conduct simulation studies on a networked airborne computing system.

In the rest of this paper, Sec. II describes the computing system, distributed computing schemes, and the optimization problem. Sec. III transforms the original problem into a MARL problem and then presents a MADDPG- and batch processing based solution. Simulation results are presented in Sec. IV, followed by the conclusion in Sec. V.

II. Problem Description

In this section, we first describe the computing system and the computation tasks considered in this study, and then present the uncoded and coded distributed computing schemes. Finally, we formulate the optimization problem.

A. Computing System

Consider a MAHC system that consists of multiple mobile devices with different computing powers. These devices can talk to each other through wireless communication. Suppose one of the devices receives/has a sequence of K matrix-vector multiplication tasks to complete, denoted as $\{\mathbf{A}\mathbf{x}_1, \mathbf{A}\mathbf{x}_2, \ldots, \mathbf{A}\mathbf{x}_K\}$, where $\mathbf{A} \in \mathbb{R}^{p\times m}$ is a pre-stored matrix and $\mathbf{x}_j \in \mathbb{R}^{m\times 1}$, $j \in [K] := \{1, 2, \ldots, K\}$ are input vectors. Due to limited computing power, this device, often called the *master node*, offloads the computation tasks to its N neighbors, known as the *worker nodes*, in a sequential order, with task j being offloaded only after the previous task $j-1$ has been completed.

Offloading a computation task j is achieved by first partitioning the task into N subtasks. Each worker node then executes one subtask and returns the result to the master node after completion. The master recovers and outputs the value of task j, i.e., $\mathbf{A}\mathbf{x}_j$, after receiving sufficient results and then moves on to the next task. In the following subsection, we describe how the traditional uncoded and coded computation schemes partition and allocate a task.

B. Distributed Computing Schemes

1) Uncoded Scheme: The traditional uncoded scheme partitions a matrix-vector multiplication task $\mathbf{A}\mathbf{x}_j$ by decomposing matrix $\mathbf{A}$ row-wise into N non-overlapping submatrices $\{\mathbf{A}_{1,j}, \mathbf{A}_{2,j}, \ldots, \mathbf{A}_{N,j}\}$, where $\mathbf{A}_{i,j} \in \mathbb{R}^{\ell_{i,j}\times m}$. Each worker node i computes subtask $\mathbf{A}_{i,j}\mathbf{x}_j$, $i \in [N]$. The master node is able to recover $\mathbf{A}\mathbf{x}_j$ after receiving results from all the worker nodes, by simply concatenating the results, i.e., $\mathbf{A}\mathbf{x}_j = [\mathbf{A}_{1,j}\mathbf{x}_j; \mathbf{A}_{2,j}\mathbf{x}_j; \ldots; \mathbf{A}_{N,j}\mathbf{x}_j]$.

One major drawback of this scheme is that the master node cannot recover $\mathbf{A}\mathbf{x}_j$ until all results from the worker nodes have been received. Because of this, the efficiency of the uncoded scheme is bounded by the slowest worker node, making it inefficient when uncertain system disturbances are prominent, e.g., in a MAHC system.

2) Coded Scheme: The coded scheme overcomes the drawback of the uncoded scheme by introducing redundant computations using the coding theory. In particular, for each task j, the coded scheme first encodes $\mathbf{A}$ into a larger matrix $\hat{\mathbf{A}}_j \in \mathbb{R}^{q_j\times m}$ with additional rows, i.e., $q_j > p$, by applying the equation below

$$\hat{\mathbf{A}}_j = \mathbf{G}_j\mathbf{A}. \tag{1}$$

where $\mathbf{G}_j \in \mathbb{R}^{q_j\times p}$ is an encoding matrix with the property that any matrix formed by any p rows of $\mathbf{G}_j$ has full rank. With $\hat{\mathbf{A}}_j$, the coded scheme then follows a similar procedure to partition and allocate the task. Particularly, worker node i executes subtask $\hat{\mathbf{A}}_{i,j}\mathbf{x}_j$, where $\hat{\mathbf{A}}_{i,j} \in \mathbb{R}^{\ell_{i,j}\times m}$, $i \in N$ is a submatrix of $\hat{\mathbf{A}}_j$, and $\sum_{i=1}^{N} \ell_{i,j} = q_j$.

To recover the final result $\mathbf{A}\mathbf{x}_j$, the master node waits until sufficient, but not all, results are received. Specifically, let $\hat{\mathbf{y}}_j$ be the results received by the master node by a certain time, which satisfies $\hat{\mathbf{y}}_j = \hat{\mathbf{G}}_j\mathbf{A}\mathbf{x}_j$, where $\hat{\mathbf{G}}_j$ is a submatrix of $\mathbf{G}_j$. $\mathbf{A}\mathbf{x}_j$ can then be recovered when $|\hat{\mathbf{y}}_j| \geq p$ via:

$$\mathbf{A}\mathbf{x}_j = (\hat{\mathbf{G}}_j^\top \hat{\mathbf{G}}_j)^{-1}\hat{\mathbf{G}}_j^\top \hat{\mathbf{y}}_j, \tag{2}$$

C. Optimization Problem

In distributed computing schemes, the load numbers $\ell_{i,j}$ determine the amount of workloads assigned to each worker node. As worker nodes have different computing powers and the time required by each node to transmit the result is also different and time-varying, it is crucial to choose a proper load number $\ell_{i,j}$ for each worker node $i \in [N]$ and each task $j \in [K]$. To determine the optimal load numbers $\ell_{i,j}^*$, we formulate an optimization problem to minimize the expected task completion time.

Denote the time required by worker node i to complete the j-th task, i.e., $\hat{\mathbf{A}}_{i,j}\mathbf{x}_j$ with $\hat{\mathbf{A}}_{i,j} \in \mathbb{R}^{\ell_{i,j}\times m}$, as $T_{i,j}$. Then $T_{i,j}$ can be captured by the following equation:

$$T_{i,j} \approx T_{i,comm}(\mathbf{x}_j) + T_{i,comp}(\hat{\mathbf{A}}_{i,j}, \mathbf{x}_j) + T_{i,comm}(\hat{\mathbf{A}}_{i,j}\mathbf{x}_j) \tag{3}$$

where $T_{i,comm}(\mathbf{x}_j)$ is the time spent to send $\mathbf{x}_j$ from the master node to the worker node i, $T_{i,comp}(\hat{\mathbf{A}}_{i,j}, \mathbf{x}_j)$ is the time spent to compute subtask $\hat{\mathbf{A}}_{i,j}\mathbf{x}_j$, and $T_{i,comm}(\hat{\mathbf{A}}_{i,j}\mathbf{x}_j)$ is the time spent to send the computation result back to the master node. Note that $T_{i,j}$ is a random variable reflective of various environmental and system uncertainties. Explicit communication and computation models are not available to the nodes.

With $T_{i,j}$, we can then describe the task completion time for each task $j \in [K]$ as follows:

$$T_j = \min_t \ \{t \ \mid \ R_j(t) \geq p\} \tag{4}$$

where $R_j(t) = \sum_{i=1}^{N} \ell_{i,j} \mathbb{1}_{T_{i,j} \leq t}$ is the total number of rows of inner product results for task j that the master node has received by time t. $\mathbb{1}$ is the indicator function [20]. Finally, the optimization problem to solve can be formulated as follows,

$$\underset{\ell_{i,j}, \forall i \in [N], \forall j \in [K]}{\text{minimize}} \quad \mathbb{E}\left[\sum_{j=1}^{K} T_j\right] \tag{5}$$

$$\text{subject to} \quad \ell_{i,j} \in \mathbb{Z}^+, \forall i \in [N], \forall j \in [K]$$

$$\sum_{i=1}^{N} \ell_{i,j} \geq p, \forall i \in [N], \forall j \in [K] \tag{6}$$

III. Main Results

In this section, we solve the optimization problem in (5), by first transforming it into a MARL problem and then developing a MADDPG- and batch processing based solution.

A. MARL-based Formulation

To solve the optimization problem in (5), we provide a MARL-based formulation. In particular, the N worker nodes are regarded as the *agents* that learn to achieve the optimization goal in (5) by interacting with the environment. The *state* of each agent i at time t is denoted as $\mathbf{s}_i(t) = [d_i(t), \mathbf{d}_{-i}(t), \mathbf{v}_i(t), \mathbf{v}_{-i}(t), \mathbf{v}_m(t)]^\top \in \mathcal{S}_i$, where $d_i(t)$ and $\mathbf{d}_{-i}(t)$ represent the distance of agent i to the master node and the distances of all agents except agent i to the master node at time t, respectively. $\mathbf{v}_i(t)$, $\mathbf{v}_{-i}(t)$, and $\mathbf{v}_m(t)$ represent the velocity of agent i, velocities of all agents except agent i, and velocity of the master node at time t, respectively. $\mathcal{S}_i$ is the corresponding state space.

Upon start of a new task j, each agent i decides the computation load it will execute for this task. This differs from the traditional load allocation mechanisms where the master node decides the computation load for each worker node (agent). Now denote the time when the master node sends task j to the agents as t_j, we then define the *action* of each agent i taken at time t_j as $a_i(t_j) = \ell_{i,j} \in \mathcal{A}_i$, where $\mathcal{A}_i$ is the corresponding action space. It is reasonable to set $\mathcal{A}_i = [0, p]$, as the total computation load required for completing task j is p. Note that, during the execution of task j, agent i does not take any actions, and the next action is taken when the next task $j+1$ starts. The transition of the agent's state from $\mathbf{s}_i(t_j)$ to $\mathbf{s}_i(t_{j+1})$ is determined by the mobility model of the agent described by $\mathbf{s}_i(t_{j+1}) = f_i(\mathbf{s}_i(t_j))$, and $t_{j+1} = t_j + T_j$, where T_j is the time spent to complete task j. Here we assume the velocities of all agents and the master node do not change during the execution each task j.

After each transition, agent i receives a reward $r_i(\mathbf{s}(t_j), \mathbf{a}(t_j)) : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$, where $\mathbf{s}(t) = (\mathbf{s}_1(t), \mathbf{s}_2(t), \ldots, \mathbf{s}_N(t)) \in \mathcal{S} := \prod_{i \in [N]} \mathcal{S}_i$ and $\mathbf{a}(t) = (a_1(t), a_2(t), \ldots, a_N(t)) \in \mathcal{A} := \prod_{i \in [N]} \mathcal{A}_i$ are the joint state and action for all agents, respectively, and $\mathcal{S}$, $\mathcal{A}$ are the corresponding joint state and action spaces. Here the reward function is defined as $r_i(\mathbf{s}(t_j), \mathbf{a}(t_j)) = -T_j - c\mathbb{1}_{\sum_i^N \ell_{i,j} \leq p}$, which takes the constraint (6) into consideration and c is the weight for the penalty term. Given state $\mathbf{s}_i$, each agent then aims to choose a deterministic policy, specified by function $\pi_i(\mathbf{s}_i) : \mathcal{S}_i \mapsto \mathcal{A}_i$, such that the expected cumulative discount reward given by

$$V_i^{\boldsymbol{\pi}}(\mathbf{s}) := \mathbb{E}_{\substack{\mathbf{s}(t_j) \sim \mathbf{f} \\ \mathbf{a}(t_j) \sim \boldsymbol{\pi}}}\left[\sum_{j=1}^{K} \gamma^{j-1} r_i(\mathbf{s}(t_j), \mathbf{a}(t_j)) | \mathbf{s}(t_1) = \mathbf{s}\right]$$

is maximized. In the above equation, γ is a discount factor, $\boldsymbol{\pi} = (\pi_1, \pi_2, \ldots, \pi_N)$ is the concatenated policy of all agents, and $\mathbf{f} = (f_1, f_2, \ldots, f_N)$ is the concatenated mobility models of all agents. $V_i^{\boldsymbol{\pi}}(\mathbf{s})$ is also known as the value function. Alternatively, an optimal policy π_i^* for agent i can be obtained by maximizing the action-value function:

$$Q_i^{\boldsymbol{\pi}}(\mathbf{s}, \mathbf{a}) := \mathbb{E}_{\substack{\mathbf{s}(t_j) \sim \mathbf{f} \\ \mathbf{a}(t_j) \sim \boldsymbol{\pi}}}\left[\sum_{j=1}^{K} \gamma^{j-1} r_i(\mathbf{s}(t_j), \mathbf{a}(t_j)) | \mathbf{s}(t_1) = \mathbf{s}, \mathbf{a}(t_1) = \mathbf{a}\right] \tag{7}$$

and setting $\pi_i^*(a_i|\mathbf{s}_i) \in \arg\max_{a_i} \max_{\mathbf{a}_{-i}} Q_i^*(\mathbf{s}, \mathbf{a})$, where $Q_i^*(\mathbf{s}, \mathbf{a}) := \max_{\boldsymbol{\pi}} Q_i^{\boldsymbol{\pi}}(\mathbf{s}, \mathbf{a})$ and $\mathbf{a}_{-i}$ denotes the actions of all agents except i.

B. MADDPG- and Batch-Processing based Solution

To solve the above MARL problem, we apply an advanced MARL algorithm proposed recently, called multi-agent deep deterministic policy gradient (MADDPG) [19]. The MADDPG is an off-policy algorithm that extends the deep deterministic policy gradient (DDPG) method to multiple agents. For each agent i, it uses four neural networks to approximate its policy $\pi_i(\mathbf{s}_i; \theta_{\pi_i})$, action-value function $Q_i^{\boldsymbol{\pi}}(\mathbf{s}, \mathbf{a}; \theta_{Q_i})$, target policy $\pi_i'(\mathbf{s}_i; \theta'_{\pi_i})$, and target action-value function $Q_i^{\boldsymbol{\pi}'}(\mathbf{s}, \mathbf{a}; \theta'_{Q_i})$, respectively, where $\boldsymbol{\pi}' = (\pi_1', \pi_2', \ldots, \pi_N')$ is the concatenated target policy of all agents. A replay memory denoted by $\mathcal{D}$ is used to store the transitions $\{(\mathbf{s}, \mathbf{a}, \mathbf{r}, \mathbf{s}')\}$, where $\mathbf{r} = [r_1(\mathbf{s}, \mathbf{a}), r_2(\mathbf{s}, \mathbf{a}), \ldots, r_N(\mathbf{s}, \mathbf{a})]$ and $\mathbf{s}' = \mathbf{f}(\mathbf{s})$. In each training iteration, a mini-batch $\mathcal{B}_i$ is sampled from the replay memory $\mathcal{D}$ and used to learn the parameters for agent i. In particular, the parameters θ_{Q_i} for the action-value function are updated by minimizing the temporal-difference error given as follows:

$$J(\theta_{Q_i}) = \frac{1}{|\mathcal{B}_i|} \sum_{(\mathbf{s}, \mathbf{a}, \mathbf{s}', \mathbf{r}) \in \mathcal{B}_i} \left(L_i^{\boldsymbol{\pi}'}(\mathbf{s}', r_i) - Q_i^{\boldsymbol{\pi}}(\mathbf{s}, \mathbf{a}; \theta_{Q_i})\right)^2 \tag{8}$$

where $L_i^{\boldsymbol{\pi}'}(\mathbf{s}', r_i) = r_i(\mathbf{s}, \mathbf{a}) + \gamma Q_i^{\boldsymbol{\pi}'}(\mathbf{s}', \boldsymbol{\pi}'(\mathbf{s}'))$. The policy parameters θ_{π_i} are updated using gradient ascent based on the policy gradient theorem, with the gradient provided as follows [21]:

$$\nabla_{\theta_{\pi_i}} J(\theta_{\pi_i}) \approx \frac{1}{|\mathcal{B}_i|} \sum_{(\mathbf{s}, \mathbf{a}, \mathbf{s}', \mathbf{r}) \in \mathcal{B}_i} \nabla_{\theta_{\pi_i}} \pi_i(\mathbf{s}_i; \theta_{\pi_i}) \nabla_{a_i} Q_i^{\boldsymbol{\pi}}(\mathbf{s}, \mathbf{a}). \tag{9}$$

The parameters for the target policy and the target action-value function are updated via Polyak averaging using the following equations:

$$\begin{aligned} \theta'_{\pi_i} &\leftarrow \tau \theta'_{\pi_i} + (1 - \tau)\theta_{\pi_i} \\ \theta'_{Q_i} &\leftarrow \tau \theta'_{Q_i} + (1 - \tau)\theta_{Q_i}, \end{aligned} \tag{10}$$

where $\tau \in (0,1)$ is a hyperparameter. The procedure of MADDPG is summarized in Algorithm 1.

Algorithm 1: MADDPG Algorithm

```
// Initialize parameters
Initialize θ_{π_i}, θ_{Q_i}, θ'_{π_i}, θ'_{Q_i}, ∀i ∈ [N]
for iteration = 1 : max_iteration do
    // Collect transitions
    for k = 1 : max_episode_number do
        for j = 1 : K do
            For each agent i, select
              a_i(t_j) = π_i(s_i(t_j); θ_{π_i}).
            Execute joint action a(t_j) and receive new
              state s'(t_{j+1}) and reward r(t_{j+1}).
            Store (s(t_j), a(t_j), r(s(t_j), a(t_j)), s'(t_{j+1}))
              in replay buffer D.
    Sample a random mini-batch B_i for each agent
      i, ∀i ∈ [N].
    // Update parameters
    for i = 1 : N do
        Update θ_{Q_i} by minimizing the
          temporal-difference error in (8).
        Update θ_{π_i} using gradient ascent, with the
          gradient provided in (9).
        Update θ'_{π_i} and θ'_{Q_i} using (10).
```

To further improve the computational efficiency, we apply the batch processing procedure developed in our previous studies [14], [15]. In particular, we let each worker node i further divide matrix $\hat{\mathbf{A}}_{i,j}$ into a set of submatrices $\{\hat{\mathbf{A}}_{i,j,k}\}_{k=1}^{w_{i,j}}$ row-wise, namely batches, where each submatrix $\hat{\mathbf{A}}_{i,j,k}$ has $b_{i,j}$ number of rows and $w_{i,j} = \lceil \frac{\ell_{i,j}}{b_{i,j}} \rceil$ is the total number of batches. Note that the last batch has a size of $\ell_{i,j} - (w_{i,j}-1)b_{i,j}$. The worker node then multiples the input vector $\mathbf{x}_j$ with each batch one by one and sends the partial result back to the master once available. In [14], [15], we have shown nice properties of this batch processing procedure in improving system's efficiency and the resilience to uncertain system disturbances through both theoretical and experimental studies. Algorithm 2 summarizes the procedures followed by the master and worker nodes during the execution of computation tasks.

Algorithm 2: Batch Processing Based Coded Scheme

Input: $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_K$
Output: $\mathbf{A}\mathbf{x}_1, \mathbf{A}\mathbf{x}_2, \ldots, \mathbf{A}\mathbf{x}_K$

```
// Master node:
for j = 1 : K do
    ŷ_j ← [ ]
    Broadcast x_j to all worker nodes.
    do
        Listen to the channel and collect results
          Â_{i,j,k} x_j from the worker nodes.
        ŷ_j ← [ŷ_j; Â_{i,j,k} x_j]
    while |ŷ_j| < p
    Send acknowledgements to all worker nodes.
    Ax_j ← (Ĝ_j^⊤ Ĝ_j)^{-1} Ĝ_j^⊤ ŷ_j
    return Ax_j
// Worker node i:
for j = 1 : K do
    Determine ℓ_{i,j} by applying the policy trained
      using Algorithm 1.
    Upon receiving x_j:
    Select ℓ_{i,j} rows from Â to construct Â_{i,j}.
    Divide Â_{i,j} into batches Â_{i,j,k}, k ∈ [w_{i,j}]
    for k = 1 : w_{i,j} do
        if Acknowledgement received then
            break
        Compute Â_{i,j,k} x_j.
        Send Â_{i,j,k} x_j back to the master node.
```

IV. Simulation Studies

In this section, we conduct simulation studies to evaluate the performance of the proposed MADDPG- and batch-processing based algorithm. As an illustration, we consider a MAHC system formed by multiple UAVs. All experiments are run on an Alienware Desktop with 32GB memory, 16-cores CPU with 3.6GHz. In the rest of the section, we first describe the simulation settings and then show the results.

A. Simulation Settings

1) Environment Description: We consider a MAHC system that consists of multiple UAVs flying at the same altitude [22]. Each UAV is equipped with a micro-computer with different computing powers. The communication between two UAVs is achieved through directional antennas for extended communication range and higher bandwidth. Assume the directional antennas are always aligned during the movement, which can be achieved by the control algorithms developed in [22]. We can then describe the time to transmit a matrix $\mathbf{X} \in \mathbb{R}^{m_1 \times n_1}$ between two UAVs as $T_{comm}(\mathbf{X}) = \frac{m_1 \times n_1 \times u}{C}$, where u (bits) is the average size of the elements in matrix $\mathbf{X}$. $C = Wlog_2(1 + \frac{S}{Noise})$ (bits/sec) is the data rate, according to the Shannon's theory, where W (Hz) is the communication bandwidth between the two UAVs, $Noise$ (Watts) is the noise power, which is assumed to be constant. S (Watts) is the signal power that can be modeled by $S = 10^{\frac{(S_d - 30)}{10}}$, with $S_d = P_t + 20\log_{10}(\lambda) - 20\log_{10}(4\pi) - 20\log_{10}(d) + G_{l|dBi} + \omega$. Here, P_t (dBm) is the transmitting power, λ is the wave length, $G_{l|dBi}$ is the sum of the transmitting and receiving gains, ω is Gaussian noise with zero mean and variance σ, and d is the distance between the two UAVs [22].

To capture the computation time, we adopt the modeling technique used in many recent studies [13], [23]. Particularly, we assume the time $T_{comp}(\mathbf{A}, \mathbf{x})$ to compute a matrix-vector multiplication task $\mathbf{A}\mathbf{x}$ with $\mathbf{A} \in \mathbb{R}^{\ell \times m}$ and $\mathbf{x} \in \mathbb{R}^{m \times 1}$ follows an exponential distribution defined by:

$$\mathrm{P}\left[T_{comp}(\mathbf{A}, \mathbf{x}) \leq t\right] = 1 - e^{-\frac{\beta}{\ell}(t - \alpha\ell)} \tag{11}$$

where $t \geq \alpha\ell$. β and α are straggling and shift parameters, respectively, and both β and α are positive constants.

To describe the mobility of the UAVs, we here adopt a simple model: $\mathbf{p}(t') = \mathbf{p}(t) + \mathbf{v}(t)(t' - t)$, where $\mathbf{p}(t)$ represents the position of a UAV at time t, and $t' > t$. In simulations, we assume the velocity $\mathbf{v}(t)$ of each UAV is constant, i.e., $\mathbf{v}(t) = \mathbf{v}$.

2) Distributed Computing Schemes: For comparison, we consider the following three distributed computing schemes as the benchmarks:

- **Uniform Uncoded**: This is an uncoded computation scheme which divides the computation loads equally, i.e., $\ell_i = \frac{p}{N}$, $\forall i \in [N]$.
- **Load-Balanced Uncoded** [24]: This is an uncoded computation scheme which divides the computation loads according to the computing capabilities of the worker nodes. In particular, the computation load assigned to each worker node i is inversely proportional to the expected time for this node to compute an inner product, i.e., $\ell_i \propto (\frac{\beta_i}{\alpha_i \beta_i + 1})$ and $\sum_{i=1}^{N} \ell_i = p$.
- **HCMM** [24]: This is a state-of-the-art coded scheme that computes the load number by $\ell_i = \frac{p}{h\lambda_i}$, where λ_i is the positive solution to $e^{\beta_i \lambda_i} = e^{\alpha_i \beta_i}(\beta_i \lambda_i + 1)$, $h = \sum_{i=1}^{N} \frac{\beta_i}{1+\beta_i \lambda_i}$, and β_i, α_i are the straggling and shift parameters for worker node i, respectively.

Note that both load-balanced uncoded and HCMM schemes require explicit knowledge of the nodes' computing capabilities, which is not required by our method.

3) Computation Scenarios: We consider three computation scenarios described as follows:

- **Scenario 1:** $N = 3$, $p = 6000$.
- **Scenario 2:** $N = 4$, $p = 8000$.
- **Scenario 3:** $N = 5$, $p = 10000$.

In all scenarios, the total number of tasks is set to $K = 30$, and the size of each task $\mathbf{x}_j$ is $m = 10000$. The initial position of each UAV is sampled uniformly from the range $[-100, 100]^2$, and its velocity is randomly selected from the range $[-10m/s, 10m/s]^2$ at each episode. The computing parameter β_i of each worker node i is randomly sampled from the range $[10^4, 10^5]$ and α_i is set to $\alpha_i = \frac{1}{\beta_i}$. The communication model is configured by setting $W = 10^4$, $Noise = 1.1 \times 10^{-12}$, $S_d = 6 - 20log_{10}(d)$, and $\sigma = 1$.

In our MADDPG- and batch processing based algorithm, the learning rates are set to $\alpha = 0.01$ and $\gamma = 0.95$. To approximate the policies and action-value functions, we use neural networks with four fully connected layers. Each hidden layer has 64 ReLU units. At the output layer, Sigmoid units are used by the policy networks, and linear units are used by the action-value networks. The weight for the penalty term in each reward function is set to $c = 200$.

B. Experimental Results

In this subsection, we first show the effectiveness of our algorithm and then investigate the impact of an important parameter, batch size $b_{i,j}$, on the computing performance. Comparison results with the benchmark schemes are presented at the end.

1) Effect of Training: Fig. 1 shows the training reward of our algorithm in the three scenarios. Each value is obtained by averaging the total rewards of all agents for every 250 iterations over 1000 episodes. We can see that in all scenarios, the average total reward increases and finally converges with the increasing numbers of training iterations.

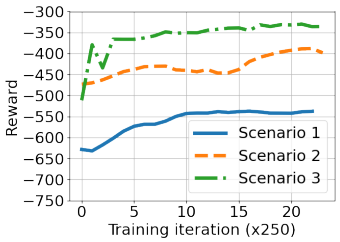

Figure 1. Training reward of our algorithm in different scenarios.

2) Impact of Batch Size: We investigate the impact of batch size $b_{i,j}$ on the performance of our algorithm, by varying $b_{i,j}$. For each value of $b_{i,j}$, we run Algorithm 2 to measure the task completion times, with load numbers $\ell_{i,j}$ determined by the policies trained using Algorithm 1. To also understand the resilience of our algorithm to uncertain system disturbances, we further consider the scenario with one uncertain straggler and the scenario without any stragglers. The straggler is introduced by randomly selecting a worker node at each iteration to sleep for 10 times of the computation time of that node using the $time.sleep()$ function.

Fig. 2 shows the total completion time of all K tasks averaged over 20 episodes in different scenarios. As we can see, with the increase of batch size, the performance degrades, which is consistent with our findings [14]. In the following comparative studies, we set the batch size to $b_{i,j} = 1$, $\forall i \in [N]$ and $j \in [K]$.

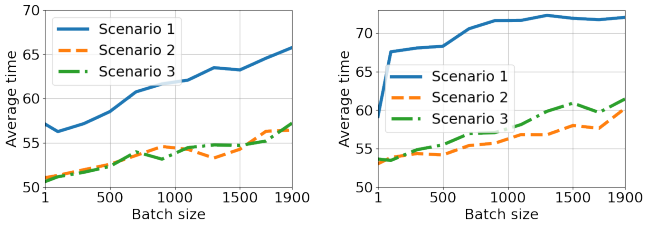

Figure 2. The average total task completion time for different batch sizes when there is (a) no straggler and (b) one uncertain straggler in different scenarios.

3) Comparative Studies: We compare our MADDPG- and batch processing based algorithm with the three benchmark schemes in different scenarios. For the load-balanced uncoded and HCMM schemes, computing parameters β_i and α_i are assumed to be known.

The average total task completion time of each scheme in different scenarios is shown in Fig. 3. As we can see, our algorithm achieves the best performance in all scenarios regardless of whether there are any uncertain stragglers. This verifies the effectiveness of our algorithm in addressing node mobility, and also demonstrates its high computational

efficiency and resilience to uncertain system disturbances. Of interest, when no stragglers are present (see Fig. 3(a)), the two uncoded schemes outperform HCMM. This is because HCMM introduces redundant computations, causing additional computation overhead. However, when stragglers are present (see Fig. 3(b)), HCMM achieves better performance than the two uncoded schemes, due to the use of the coding technique. Additionally, the uniform uncoded scheme shows the worst performance in most scenarios as it ignores the heterogeneity nature of the computing nodes.

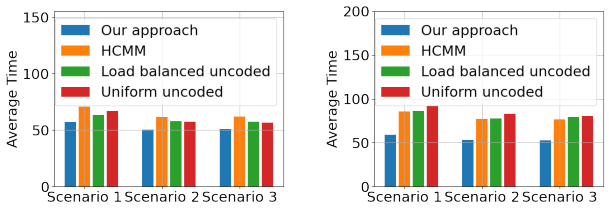

Figure 3. The average task completion time of each scheme when there is (a) no straggler and (b) one uncertain straggler in different scenarios.

V. Conclusion

In this paper, we present a MARL-based coded computation framework for the MAHC system that consists of multiple mobile computing devices sharing resources among themselves. This framework addresses both node mobility and the heterogeneity nature of mobile devices, and can be applied to any MAHC systems without having to know their communication or computing characteristics. Moreover, our framework, which applies the MADDPG, a state-of-the-art MARL algorithm, enables the load allocation to be determined in a decentralized manner, thus helping to relieve the computing burden for the master node. It also adopts a batch processing procedure developed in our previous studies to further improve the performance in the aspects of both efficiency and resilience to uncertain system disturbances. The comprehensive simulation studies show that our MADDPG- and batch processing based algorithm achieves the best performance compared with existing uncoded and coded schemes, including the uniform uncoded, load balanced uncoded, and HCMM. In the future work, we will extend this study to consider more general scenarios, such as varying node velocities and task sizes.

Acknowledgement

We would like to thank the National Science Foundation (NSF) under Grants CI-1953048/1730675/1730570/1730325, ECCS-1953049/1839804, CAREER-2048266 and CAREER-1714519 for the support of this work.

References

[1] S. Wang, X. Zhang, Y. Zhang, L. Wang, J. Yang, and W. Wang, "A survey on mobile edge networks: Convergence of computing, caching and communications," *Ieee Access*, vol. 5, pp. 6757–6779, 2017.

[2] X. Cao, L. Liu, Y. Cheng, and X. Shen, "Towards energy-efficient wireless networking in the big data era: A survey," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 1, pp. 303–332, 2017.

[3] I. Yaqoob, E. Ahmed, A. Gani, S. Mokhtar, M. Imran, and S. Guizani, "Mobile ad hoc cloud: A survey," *Wireless Communications and Mobile Computing*, vol. 16, no. 16, pp. 2572–2589, 2016.

[4] 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.

[5] L. Liu, C. Chen, Q. Pei, S. Maharjan, and Y. Zhang, "Vehicular edge computing and networking: A survey," *Mobile Networks and Applications*, pp. 1–24, 2020.

[6] B. Wang, J. Xie, S. Li, Y. Wan, Y. Gu, S. Fu, and K. Lu, "Computing in the air: An open airborne computing platform," *IET Communications*, vol. 14, no. 15, pp. 2410–2419, 2020.

[7] B. Wang, J. Xie, S. Li, Y. Wan, S. Fu, and K. Lu, "Enabling high-performance onboard computing with virtualization for unmanned aerial systems," in *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, 2018, pp. 202–211.

[8] K. Lee, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Coded computation for multicore setups," in *Proc. of IEEE ISIT 2017*, 2017, pp. 2413–2417.

[9] K. Zhang, Z. Yang, and T. Başar, "Multi-agent reinforcement learning: A selective overview of theories and algorithms," *arXiv preprint arXiv:1911.10635*, 2019.

[10] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding Up Distributed Machine Learning Using Codes," *IEEE Transactions on Information Theory*, vol. 64, no. 3, pp. 1514–1529, mar 2018.

[11] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient Coding: Avoiding Stragglers in Distributed Learning," in *Proc. of the 34th International Conference on Machine Learning*. PMLR, 2017, pp. 3368–3376.

[12] M. Rudow, K. Rashmi, and V. Guruswami, "A locality-based approach for coded computation," *arXiv preprint arXiv:2002.02440*, 2020.

[13] A. Reisizadeh, S. Prakash, R. Pedarsani, and S. Avestimehr, "Coded computation over heterogeneous clusters," in *Proc. of IEEE ISIT 2017*, 2017.

[14] B. Wang, J. Xie, K. Lu, Y. Wan, and S. Fu, "On batch-processing based coded computing for heterogeneous distributed computing systems," *arXiv preprint arXiv:1912.12559*, 2019.

[15] ——, "Coding for heterogeneous uav-based networked airborne computing," in *2019 IEEE Globecom Workshops (GC Wkshps)*. IEEE, 2019, pp. 1–6.

[16] Y. Keshtkarjahromi, Y. Xing, and H. Seferoglu, "Dynamic heterogeneity-aware coded cooperative computation at the edge," in *Proc. of ICNP 2018*. IEEE, sep 2018, pp. 23–33.

[17] H. Lu, C. Gu, F. Luo, W. Ding, and X. Liu, "Optimization of lightweight task offloading strategy for mobile edge computing based on deep reinforcement learning," *Future Generation Computer Systems*, vol. 102, pp. 847–861, 2020.

[18] J. Chen, Z. Wei, S. Li, and B. Cao, "Artificial intelligence aided joint bit rate selection and radio resource allocation for adaptive video streaming over f-rans," *IEEE Wireless Communications*, vol. 27, no. 2, pp. 36–43, 2020.

[19] R. Lowe, Y. I. Wu, A. Tamar, J. Harb, O. P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *Advances in neural information processing systems*, 2017, pp. 6379–6390.

[20] Q. Y. Kenny *et al.*, "Indicator function and its application in two-level factorial designs," *The Annals of Statistics*, vol. 31, no. 3, pp. 984–994, 2003.

[21] R. S. Sutton and A. G. Barto, "An introduction to reinforcement learning, chapter 3," 2018.

[22] M. Liu, Y. Wan, S. Li, F. L. Lewis, and S. Fu, "Learning and uncertainty-exploited directional antenna control for robust long-distance and broad-band aerial communication," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 1, pp. 593–606, 2019.

[23] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," in *Proc. of IEEE ISIT 2016*. IEEE, aug 2016, pp. 1143–1147.

[24] A. Reisizadeh, S. Prakash, R. Pedarsani, and A. S. Avestimehr, "Coded computation over heterogeneous clusters," *IEEE Transactions on Information Theory*, vol. 65, no. 7, pp. 4227–4242, 2019.