A Continuum Approach for Collaborative Task Processing in UAV MEC Networks

Lorson Blair, Carlos A. Varela, Stacy Patterson
Department of Computer Science
Rensselaer Polytechnic Institute
Troy, NY, USA
blairl@rpi.edu, cvarela@cs.rpi.edu, sep@cs.rpi.edu

***Abstract*—Unmanned aerial vehicles (UAVs) are becoming a viable platform for sensing and estimation in a wide variety of applications including disaster response, search and rescue, and security monitoring. These sensing UAVs have limited battery and computational capabilities, and thus must offload their data so it can be processed to provide actionable intelligence. We consider a compute platform consisting of a limited number of highly-resourced UAVs that act as mobile edge computing (MEC) servers to process the workload on premises. We propose a novel distributed solution to the collaborative processing problem that adaptively positions the MEC UAVs in response to the changing workload that arises both from the sensing UAVs' mobility and the task generation. Our solution consists of two key building blocks: (1) an efficient workload estimation process by which the UAVs estimate the task field—a continuous approximation of the number of tasks to be processed at each location in the airspace, and (2) a distributed optimization method by which the UAVs partition the task field so as to maximize the system throughput. We evaluate our proposed solution using realistic models of surveillance UAV mobility and show that our method achieves up to 28% improvement in throughput over a non-adaptive baseline approach.**

***Index Terms*—Mobile edge computing, task offloading, Voronoi partitioning.**

I. Introduction

In its 2021 climate change report, the United Nations' International Panel on Climate Change (IPCC) gave a grim forecast regarding the frequency, severity, and devastating consequences of natural disasters—hurricanes, wild fires, floods, etc.—as a result of climate change [1]. With more catastrophic disasters, there is an urgent need for more sophisticated prediction models and enhanced disaster management and search and rescue techniques. Due to their flexible mobility, relatively cheap cost, and ease of deployment, unmanned aerial vehicles (UAVs) are becoming increasingly popular for use in these applications [2], [3]. UAVs can be quickly and efficiently deployed in disaster areas to assess the level of damage, monitor the progression of wildfires and floods, draw digital maps of the disaster area, and identify rescue candidates [4].

These applications require the collection of large amounts of data, for example, video, infrared imaging, or air quality data, which must be processed to generate actionable intelligence. UAVs have limited physical space, which is reserved for data collection hardware, as well as a limited battery capacity that must be dedicated to this data collection. Thus, the UAVs, or *mobile sensing agents* (MSAs), must offload data processing tasks to external resources. While a traditional solution is to offload this computation to the cloud, the urgency of the situation and potentially compromised infrastructure require offloading to nearby compute resources for immediate processing. A promising approach is to utilize a compute platform consisting of additional vehicles equipped with larger batteries and dedicated Mobile Edge Computing (MEC) servers [5]–[10]. These *mobile compute agents* (MCAs) can be UAVs with larger batteries and specialized computing hardware, or they can be ground vehicles with their own power sources and servers. The MCAs have the freedom and flexibility to move in close proximity to the MSAs and to change locations in response to demand that changes both in location and quantity.

The successful deployment of such a platform requires addressing several research challenges. First, both the locations and quantities of computing demand are unpredictable and dynamic; however, this information is crucial for determining the best placement of the MCAs. Second, even if the demand is known, the problem of optimally adapting the locations of the MCAs is intractable on its face due to the complexity of wireless communication and the binary nature of associating tasks to MCAs [11], [12]. Third, unlike offloading to the cloud, MCAs have limited processing capacity, and so the assignment of tasks must also take these capacities into account to ensure balanced task processing. Finally, in disaster scenarios, communication infrastructure may be severely limited due to damage or the remoteness of the location; thus, the MCAs must be able to autonomously coordinate to provide the task processing services, reserving external communication for important results.

Several early works proposed solutions for an MEC platform consisting of a single UAV [5], [7], [13]–[16]; however, a single UAV may not be sufficient to meet the computing demands of a large number of MSAs. More recent works have considered the deployment of multi-UAV MEC systems and address the problem of optimizing the MEC UAV trajectories for performance measures like latency and energy. Many of these works assume that the demand is stationary [8], [11], [12], [17]. A few works support dynamic demand [9], [10], [18]. However, all of these works propose centralized solutions, making them unsuitable for our setting.

We propose a novel distributed solution to maximize pro-

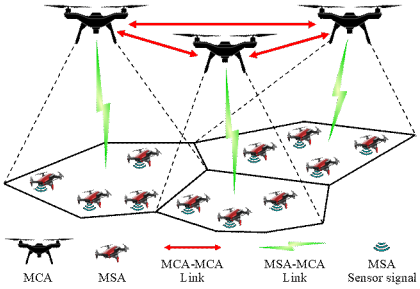


Fig. 1: System model.

cessing throughput in a multi-UAV MEC system. Our solution adaptively positions a limited number of MCAs in response to the changing workload that arises both from the mobility and task generation of the MSAs. We adopt a *continuum* approach to model the distribution of demand over the surveillance region. This unique approach enables us to formulate a continuous optimization problem that admits a distributed solution. Our solution consists of two key building blocks: (1) an efficient demand estimation algorithm in which the MCAs collaborate to generate an estimate of the demand as a continuous time-varying field, and (2) a theoretically-verified two-phase distributed optimization algorithm by which the MCAs optimize their locations to maximize the task transmission rate while ensuring that the assignments of tasks is proportional to the compute resources at each agent. We illustrate the effectiveness of our method through realistic numerical simulations, which show that our method provides up to 28% improvement processing in throughput over a non-adaptive baseline method.

The rest of the paper is organized as follows. In Section II, we describe the system model and problem formulation. Our proposed method is presented in Section III. Simulation results are presented in Section IV. We summarize related work in Section V. Finally, we conclude in Section VI.

II. System Model and Problem Formulation

As shown in Fig. 1, our system consists of a set of MSAs that are responsible for surveilling a region of interest. The MSAs do not have the compute resources or battery power for onboard processing, so they must offload the computation tasks to more highly-resourced MCAs. The goal is to offload the MSA tasks to the MCAs and process them as quickly as possible to inform system operators and emergency personnel.

To achieve this goal, the MCAs should be positioned close to MSAs so that the transmission rate is maximized. Since the transmission rate is inversely proportional to the transmit time, maximizing the rate also minimizes the energy expended to transmit each unit of data, which extends the surveillance lifetime of the MSAs. We must also ensure that the amount of work assigned to an MCA at any time does not exceed its computing capacity to avoid queuing delays at the MCAs. Thus, the challenge is to dynamically update the positions and task assignments of the MCAs in response to the changing demands of the MSAs (both in terms of quantity and location), while respecting the capacities of the MCAs.

In the remainder of this section, we formalize our system model and the distributed task processing problem.

A. System Model

There are N MSAs that fly at a fixed mean sea-level (MSL) altitude of H_s within a 3-dimensional region Ω. Using a fixed altitude, rather than frequently ascending and descending, allows the MSAs to save energy [6], [14]. We denote the position of MSA n at time t by $\mathbf{w}_n(t) = [x_n(t), y_n(t), H_s]^T$.

The M MCAs operate in the same region Ω at a fixed MSL altitude of $H_c \neq H_s$ to avoid collisions with the MSAs. The MCAs update their locations collaboratively to provide processing services to the MSAs. We denote the position of MCA m at time t by $\mathbf{u}_m(t) = [x_m(t), y_m(t), H_c]^T$. We drop the t from $\mathbf{w}_n(t)$ and $\mathbf{u}_m(t)$ when the context is clear. We assume that the MCAs do not have control over, nor a priori knowledge of, the MSA trajectories.

1) Communication Model: We make the common assumptions that the MCAs and MSAs are equipped with omni-directional antennas and that they communicate using an Orthogonal Frequency-Division Multiple Access (OFDMA) transmission mechanism (as in [19], [20]). The communication channel between MCA m and MSA n is dominated by the line-of-sight link transmission with a distance dependent path-loss model. The maximum transmission rate between MCA m and MSA n in bits per second (bps) at time t depends on the distance between them and is given by

$$r(\mathbf{u}_m, \mathbf{w}_n) = B \log_2 \left(1 + \frac{\beta P}{\sigma^2 \|\mathbf{u}_m - \mathbf{w}_n\|^2}\right) \qquad (1)$$

where B is the bandwidth allocated to each MSA, β is the channel gain at a reference distance of 1m, σ^2 is the power of the white Gaussian noise, and P is the MSA's transmit power.

A small portion of the OFDMA subcarriers are used as control channels for lightweight communication among MSAs and MCAs. As is common, we assume that the effects of this control channel communication on the maximum transmission rate is negligible.

2) Workload and Offloading Model: Each MSA n stores a queue of its unprocessed tasks. Tasks are continuously added to the queue as the MSA collects data and are removed from the queue when the MSA offloads them to an MCA. Let $d_n(t)$ denote the quantity of unprocessed tasks at MSA n at time t. As in previous works (cf. [5], [7], [13], [14], [16], [18]), we assume that the tasks are arbitrarily divisible. At each time t, each MSA n is assigned to offload to a single MCA m. Each MCA m has a processing capacity c_m, which represents the amount of tasks it can process per second (bps), i.e., its maximum throughput. The MSA offloads to its assigned MCA up to the rate defined in (1), provided the MCA has remaining capacity. If the MCA does not have remaining capacity, the MSA stores the tasks until it is assigned to an MCA that has capacity to process them. We note that multiple MSAs can be assigned to the same MCA simultaneously. In this case, all

MSAs offload in parallel until the MCA's capacity is saturated, at which point, the MSAs queue any remaining tasks.

B. Problem Formulation

Our goal is to develop a distributed solution by which the MCAs collaboratively maximize the system throughput by maximizing the total task transmission rate, subject to the constraint that the amount of tasks assigned to each MCA m does not exceed its processing capacity c_m. To achieve this, at each time t, the MCAs must identify both their optimal locations as well as the assignment of MSAs. The MCAs can query the MSAs to learn the current quantities of queued tasks $d_n(t)$ and the current MSA locations $\mathbf{w}_n(t)$.

We define the optimization variables $\tau(n,t)$, $n = 1 \dots N$; the variable $\tau(n,t)$ gives the rate at which MSA n transmits its data at time t. We also define the set of binary assignment variables $a_{m,n}(t)$, $n = 1 \dots N, m = 1 \dots M$, where $a_{m,n}(t) = 1$ if MSA n is assigned to MCA m at time t, and is 0 otherwise. Let Ω_c denote the two-dimensional plane in Ω at MSL height H_c. We formalize our problem as follows:

$$\underset{\substack{\mathbf{u}_m \in \Omega_c, m=1\dots M \\ a_{m,n}, m=1\dots M, n=1\dots N}}{\text{maximize}} \quad \sum_{m=1}^{M} \sum_{n=1}^{N} a_{m,n}(t)\tau(n,t) \tag{2}$$

subject to

$$\sum_{n=1}^{N} a_{m,n}(t)\tau(n,t) \leq c_m, \quad m = 1 \dots M \tag{3}$$

$$\tau(n,t) \leq d_n(t), \quad n = 1 \dots N \tag{4}$$

$$a_{m,n}(t)\tau(n,t) \leq r(\mathbf{u}_m, \mathbf{w}_n), \; m = 1 \dots M, n = 1 \dots N \tag{5}$$

$$\sum_{m=1}^{M} a_{m,n}(t) = 1, \quad n = 1 \dots N \tag{6}$$

$$a_{m,n}(t) \in \{0,1\}, \quad m = 1 \dots M, n = 1 \dots N. \tag{7}$$

Here, the objective (2) is to maximize the total transmission rate. The constraints (3) ensure that the total transmission rate to each MCA does not exceed its processing capacity. The constraints (4) require that an MSA cannot offload more data than it has in its queue, and the constraints (5) require that the transmission rate between an MSA and an MCA does not exceed the maximum transmission rate defined in (1). Finally, the constraints (6) and (7) ensure that each MSA is assigned to exactly one MCA.

The above optimization problem presents multiple challenges. First the trajectories and the workloads of the MSAs are not known a priori, nor can they be controlled by the MCAs. Second, even if these trajectories and locations were known, this optimization problem is a mixed-integer nonlinear programming problem, which is NP-Hard, in general. Further, both the objective and constraints are coupled across the MCA locations and assignments. Thus, even if the problem were tractable, it is not straightforward to devise a distributed solution for it.

To address these challenges, we model the workload as a continuous spatial field, a *task field*, rather than considering the discrete locations of the MSAs. This approach allows us to devise a distributed solution by which the MCAs predict the evolution of the task field and adaptively position themselves to maximize the transmission rates while maintaining their processing capacity constraints. We describe our continuum approach and proposed solution in the next section.

III. Continuum Approach

As a first step towards making the problem tractable, we develop a continuous representation of the tasks held by the MSAs. We define a continuous task field over the sensing airspace Ω_s, the two-dimensional plane in Ω at height H_s. This field is described by a continuous function ρ where $\rho(\mathbf{x}, t) \in \mathbb{R}_{\geq 0}$ quantifies the amount of tasks, in bits, that needs to be offloaded for processing from location $\mathbf{x} \in \Omega_s$ at time t.

Let $\mathbf{u}(t) = [\mathbf{u}_1(t)^T, \dots, \mathbf{u}_M(t)^T]^T$ denote the positions of the MCAs at time t. To maximize the total transmission rate, the tasks $\rho(\mathbf{x}, t)$ should be assigned to the MCA m with maximal transmission rate $r(\mathbf{u}_m, \mathbf{x})$. We assign a cost to offloading one bit from location $\mathbf{x}$ to $\mathbf{y}$, defined by

$$\omega(\mathbf{x}, \mathbf{y}) = \frac{1}{r(\mathbf{x}, \mathbf{y})} \tag{8}$$

and note that minimizing $\omega(\mathbf{x}, \mathbf{y})$ will maximize $r(\mathbf{x}, \mathbf{y})$.

We create a Voronoi partitioning of Ω_s based on the locations of the MCAs. The Voronoi region of MCA m is defined as

$$\mathcal{V}_m(\mathbf{u}(t)) = \{\mathbf{x} \in \Omega_s \mid \omega(\mathbf{u}_m(t), \mathbf{x}) \leq \omega(\mathbf{u}_k(t), \mathbf{x}) \quad k = 1 \dots M, k \neq m\}. \tag{9}$$

Note that the Voronoi region of MCA m depends on the positions of all MCAs. We denote the Voronoi partitioning of the entire region by:

$$\mathcal{V}(\mathbf{u}(t)) = \{\mathcal{V}_1(\mathbf{u}_1(t)), \dots, \mathcal{V}_M(\mathbf{u}_m(t))\}. \tag{10}$$

For a given Voronoi partitioning, the amount of tasks assigned to MCA m at time t is given by the volume of its Voronoi region:

$$|\mathcal{V}_m(\mathbf{u}_m(t))| = \int_{\mathcal{V}_m} \rho(\mathbf{x}, t) \, \mathrm{d}\mathbf{x}. \tag{11}$$

If $|\mathcal{V}_m| \leq c_m$, then the amount of tasks assigned to MCA m is less than or equal to its processing capacity. The total cost of the tasks assigned to MCA m is:

$$f_m(\mathbf{u}(t)) = \int_{\mathcal{V}_m} \rho(\mathbf{x}, t)\omega(\mathbf{u}_m(t), \mathbf{x}) \, \mathrm{d}\mathbf{x} \tag{12}$$

and the total system cost is:

$$F(\mathbf{u}) = \sum_{m=1}^{M} f_m(\mathbf{u}(t)). \tag{13}$$

We now reformulate problem (2) – (7) using this continuum task field model and the Voronoi partitioning.

$$\underset{\mathbf{u}_m \in \Omega_c, m=1\dots M}{\text{minimize}} \quad F(\mathbf{u}(t)) \tag{14}$$

$$\text{subject to } |\mathcal{V}_m| \leq c_m, \quad m = 1 \dots M. \tag{15}$$

The solution to this problem gives the locations of the MCAs as well as the assignment of tasks to MCAs. By minimizing F, we maximize the total transmission rate, while the constraints (15) ensure the capacity constraints of the MCAs are respected. Provided the Voronoi region volumes do not violate the capacity constraints, the tasks can all be offloaded and processed at their respective maximal transmission rates.

The objective (14) and the constraints (15) are non-convex, and as far as we are aware, there is no single distributed algorithm to solve such a problem. Thus, we decompose the problem into tractable components that each admit a distributed solution. We then combine these components to form our full solution. We describe our proposed solution and each of its components below.

A. Proposed Solution

We propose a distributed solution to problem (14) - (15). Our solution is divided into four phases:

1) **Task field estimation.** In this phase, the MCAs collect information about the current state of the task queues of the MSAs and generate an estimate of ρ.
2) **Transmission rate maximization.** Next, the MCAs use a distributed algorithm to determine the positions that minimize $F(\mathbf{u})$ over this estimated field ρ.
3) **Capacity balancing.** Then, the MCAs use a second distributed algorithm to collaboratively update these positions so that the resulting Voronoi region volumes are proportional to their processing capacities.
4) **Transmission and processing.** The MCAs move to the new positions identified in Phase 3, and the MSAs are assigned to MCAs using the Voronoi partitioning defined in (10). The transmission rates are determined in a round robin fashion. MCA m considers each MSA n one at a time and sets $\tau(n,t)$ to the maximum of $r(\mathbf{u}_m, \mathbf{w}_n)$ and its remaining capacity.

Rather than continuously executing the first three phases, the MCAs execute them every Δ seconds. A smaller value of Δ allows the MCAs to more accurately estimate ρ and adapt their positions as it changes over time. This costs the MCAs energy to update their positions, which decreases the energy available for task processing. A larger value of Δ leads to less accurate estimates of ρ and potentially lower transmission rates, but it saves in the motion energy. We explore different values of Δ in the experiments.

For the first Δ seconds, the MCAs hover in their initial positions. After Δ seconds, they query the MSAs, which respond with the number of tasks they each generated in the first Δ seconds. The MCAs use these values as $d_n(t)$ in phase 1. The MCAs execute phases 2 and 3 to determine their next locations, and then update their positions accordingly by flying along straight paths to their destinations. They hover in these positions until 2Δ seconds total have elapsed, and the entire process is repeated. Tasks are continuously assigned and processed as described in phase 4 using the Voronoi partitioning corresponding to the current MCA positions.

Next, we present the details of the first three phases.

B. Task Field Estimation

We consider $\rho(x)$ as a spatial Gaussian process, $GP(\mu, \mathcal{K}(\mathbf{x},\mathbf{y}))$ for $\mathbf{x},\mathbf{y} \in \Omega_s$. The constant μ is the mean of the Gaussian process, and $\mathcal{K}$ is the kernel function that gives the spatial correlation of ρ. We use the squared exponential Gaussian kernel

$$\mathcal{K}(\mathbf{x},\mathbf{y}) = \varphi^2 e^{\left(-\frac{\|\mathbf{x}-\mathbf{y}\|^2}{L^2}\right)} \tag{16}$$

where φ^2 is the variance of the Gaussian process and A_1 is the length scale. Gaussian processes are used in many applications to model spatial processes [21].

At the beginning of each window, at time t_0, each MCA collects the value $d_n(t_0)$ and location $\mathbf{w}_n(t_0)$ from each MSA in its Voronoi region. The MCAs share these values with each other over the control channels. Then, independently, each MCA uses the observed values of $d_n(t_0)$ and $\mathbf{w}_n(t_0)$, $n = 1 \dots N$, to construct an estimate of ρ using Gaussian process regression, where μ, φ^2, and L are estimated from the observed values.

C. Transmission Rate Maximization Algorithm

In this phase, the MCAs collaborate to solve the unconstrained optimization problem:

$$\underset{\mathbf{u}_m \in \Omega_c, m=1,\dots,M}{\text{minimize}} \quad F(\mathbf{u}) = \sum_{m=1}^{M} f_m(\mathbf{u}). \tag{17}$$

By doing so, the MCAs identify the positions and Voronoi partitioning that maximizes the total *maximal* transmission rate. However, this partitioning does not necessarily respect the capacities of the MCAs.

Each MCA communicates with nearby MCAs to solve this problem. We say that two MCAs are *neighbors* for window b if their Voronoi regions are adjacent in window $b-1$. We define the communication graph for window b as $\mathcal{G}_b = (V, E_b)$, where V is the set of MCAs, and $(m,k) \in E_b$ if and only if MCAs m and k are neighbors in window $b-1$. We denote the set of neighbors of MCA m in window b by $\mathcal{N}_b(m)$.

Each MCA m stores an estimate of the positions of all M MCAs. We let $\mathbf{u}_k^m$ denote MCA m's estimate of the position of MCA k, and $\mathbf{u}^m$ denotes the concatenation of all the estimates held by MCA m, i.e.,

$$\mathbf{u}^m = [(\mathbf{u}_1^m)^T, \dots, (\mathbf{u}_M^m)^T]^T.$$

We can then rewrite (17) in the following decomposable form:

$$\underset{\mathbf{u}^1,\dots,\mathbf{u}^m}{\text{minimize}} \ \hat{F}([(\mathbf{u}^1)^T, \dots, (\mathbf{u}^M)^T]^T) = \sum_{m=1}^{M} f_m(\mathbf{u}^m) \tag{18}$$

$$\text{subject to } \mathbf{u}^m = \mathbf{u}^k, \text{ for all } (m,k) \in E_b. \tag{19}$$

The MCAs use a distributed projected gradient algorithm to solve this problem [22], [23]. We now describe the details of this algorithm. The pseudocode is shown in Alg. 1. The algorithm executes in discrete iterations $\ell = 0, 1, 2, \dots, T_1$. Each MCA initializes its estimate $\mathbf{u}^m$ to be the current MCA positions, i.e., their positions at time t_0. In each iteration ℓ,

Algorithm 1 Transmission rate maximization algorithm.

Initialize: $\mathbf{u}^m(0)$ to current MCA positions
for $\ell = 0, 1, 2, \ldots, T_1$ **do**
  for each MCA m **do**
    $\mathbf{z}^m(\ell) \leftarrow \text{Proj}_{\Omega_c}(\mathbf{u}^m(\ell) - \eta_\ell \nabla f_m(\mathbf{u}^m(\ell)))$
    $\mathbf{u}^m(\ell+1) \leftarrow \xi \sum_{j \in \mathcal{N}_b(m)} \mathbf{z}^j(\ell) + (1 - \xi|\mathcal{N}_b(m)|)\mathbf{z}^m(\ell)$
  end for
end for

each MCA m first computes the derivative of f_m with respect to its estimate $\mathbf{u}^m$. It then performs a gradient step on this estimate with step size η_ℓ and projects the result of this gradient step onto Ω_c, where $\text{Proj}_{\Omega_c}(\mathbf{x})$ denotes the Euclidean projection onto the region Ω_c. To complete the iteration, the MCA updates its estimate by taking a weighted average of its own estimate and those of its neighbors; MCA m applies weight ξ to each of its neighbor's estimates and weight $(1 - \xi|\mathcal{N}_b(m)|)$ to its own estimate.

The MCAs repeat this iteration until a desired convergence is achieved (typically within a hundred iterations in our experiments). Intuitively, through the projected gradient steps of Alg. 1, each MCA updates its estimates of the positions to maximize the transmission rate in its own Voronoi region, while the averaging steps drive the estimates to agreement.

We now summarize the convergence behavior of Alg. 1. Define the set of stationary points on $(\Omega_c)^{M^2}$ as $\mathcal{L} = \{\mathbf{x} \in (\Omega_c)^{M^2} \mid \nabla \hat{F}(\mathbf{x}) \in \mathcal{C}(\mathbf{x})\}$, where $\mathcal{C}(\mathbf{x})$ is the normal cone, i.e., $\mathcal{C}(\mathbf{x}) = \{\mathbf{v} \in \mathbb{R}^{2M} \mid \forall \mathbf{u}' \in (\Omega_c)^{M^2}, \mathbf{v}^T(\mathbf{x} - \mathbf{u}')' \geq 0\}$.
With this definition, we present the following theorem.

Theorem 1: Let Ω_c be a closed, compact set, and assume that $\hat{F}(\mathcal{L})$ has an empty interior and that $\xi \leq \frac{1}{\delta}$, where δ is the maximum vertex degree of $\mathcal{G}_b$[1]. Let the step size $\{\eta_\ell\}_{\ell \geq 0}$ be such that $\sum_{\ell=0}^{\infty} \eta_\ell = \infty$ and $\sum_{\ell=0}^{\infty} \eta_\ell^2 < \infty$. Then, the sequence $\{[\mathbf{u}^1(\ell)^T, \ldots, \mathbf{u}^M(\ell)^T]^T\}_{\ell \geq 0}$ converges to the set $\{\mathbf{1} \otimes \mathbf{x} \mid \mathbf{x} \in \mathcal{L}\}$[2]. Moreover, $\frac{1}{M}\sum_{m=1}^{M} \mathbf{u}^m(\ell)$ converges to a connected component of $\mathcal{L}$.
The proof of this theorem follows from Theorem 1 in [22] and the fact that problem (18)-(19) satisfies certain technical conditions. The full proof is given in our technical report [24].

Theorem 1 shows that Alg. 1 converges to a stationary point of the transmission rate maximization problem, and further, that the MCAs position vectors $\mathbf{u}^m$, $m = 1 \ldots M$, converge to agreement at this fixed point.

D. Capacity Balancing Algorithm

While Alg. 1 aims to optimize the Voronoi partitioning for transmission rate, it may result in some Voronoi regions with workloads that exceed the capacities of their MCAs. To optimize the system throughput, the MCAs must have the capacity to process their assigned tasks. We next present a distributed algorithm to adjust the MCA positions and thus adjust the Voronoi partitioning, so that the region volumes are proportional to the processing capacities of the MCAs.

[1]If $\mathcal{G}_b$ is bipartite, we require $\xi < \frac{1}{\delta}$.

[2]The symbol $\otimes$ denotes the Kronecker product.

Algorithm 2 Capacity balancing algorithm.

Initialize: $\mathbf{u}_m(0)$ to result of Alg. 1
for $\ell = 0, 1, 2, \ldots, T_2$ **do**
  for each MCA m **do**
    $\mathbf{d}_m(\ell) \leftarrow \sum_{j \in \mathcal{N}_b(m)} \left(\left(\frac{\int_{\mathcal{V}_m} \rho(\mathbf{x}) d\mathbf{x}}{c_m} - \frac{\int_{V_j} \rho(\mathbf{x}) d\mathbf{x}}{c_j} \right) \times n_{mj} \int_{\mathcal{V}_m \cap \mathcal{V}_j} \rho(\gamma) d\gamma \right)$
    $\mathbf{u}_m(\ell+1) = \mathbf{u}_m(\ell) - \alpha \mathbf{d}_m(\ell)$
  end for
end for

We define the functions $g_m(\mathbf{u}) = |\mathcal{V}_m|^2$, $m = 1 \ldots M$, and the function G:

$$G(\mathbf{u}) = \sum_{m=1}^{M} \frac{1}{c_m} g_m(\mathbf{u}). \tag{20}$$

We define the optimization problem:

$$\underset{\mathbf{u}_m \in \Omega_c, m=1\ldots M}{\text{minimize}} \quad G(\mathbf{u}). \tag{21}$$

As shown in [25], the solution to problem (21) creates a Voronoi partitioning where the total workload assigned to each MCA is proportional to its capacity. This is formalized in the following lemma.

Lemma 1 (Lemma 1 in [25]): Let x_m, $m = 1 \ldots M$ be real variables, subject to the constraint that $\sum_{m=1}^{M} x_m = X$ where X is a constant. The function $\sum_{m=1}^{M}(x_m^2/c_m)$, with constants c_m, attains its minimum when $x_m/c_m = X/\sum_{m=1}^{M} c_m$, for $i = 1 \ldots M$.

We present a discrete-time distributed gradient algorithm, based on the continuous-time solution presented in [25], to solve problem (21). The pseudocode is shown in Alg. 2. The MCAs initialize their positions to the positions determined by the transmission rate maximization algorithm. In each iteration ℓ, each MCA communicates with its neighbors in the graph $\mathcal{G}_b$ using the control channels to exchange their values $\mathbf{u}_m(\ell)$. Each MCA m then updates $\mathbf{u}_m(\ell+1)$ to be a weighted average of its $\mathbf{u}_m(\ell)$ and that of its neighbor MCAs:

$$\mathbf{u}_m(\ell+1) = \mathbf{u}_m(\ell) - \alpha \mathbf{d}_m(\ell) \tag{22}$$

where α is the step size, and

$$\mathbf{d}_m(\ell) = \sum_{j \in \mathcal{N}_b(m)} \left(\left(\frac{\int_{\mathcal{V}_m} \rho(\mathbf{x}) d\mathbf{x}}{c_m} - \frac{\int_{V_j} \rho(\mathbf{x}) d\mathbf{x}}{c_j} \right) \times n_{mj} \int_{\mathcal{V}_m \cap \mathcal{V}_j} \rho(\gamma) d\gamma \right). \tag{23}$$

The weights used in the averaging depend on the volumes of the Voronoi regions. Here, n_{mj} denotes the unit normal of the shared boundary of the Voronoi regions of MCAs m and j. Each term in the sum in (23) exerts a "push" or

"pull" on MCA m's position proportional to how balanced m's capacity is with respect to its neighbor. The MCAs execute this algorithm until it converges to a balanced partitioning (typically within a hundred iterations in our experiments). The convergence behavior of Alg. 2 is given in the following theorem.

Theorem 2: For an appropriately chosen step size α, Alg. 2 converges asymptotically to an equilibrium where the MCA positions are such that the volume of each Voronoi region is proportional to the MCA's capacity.

Theorem 2 follows directly from Theorem 1 in [25].

By initializing Alg. 2 with positions that optimize for transmission rate, and then using Alg. 2 to adjust the positions to balance the workload to match the capacities, the resulting Voronoi partitioning incorporates both transmission rate maximization and processing capacity to optimize the total system throughput.

IV. Simulation Results

We evaluate our collaborative task processing method through numerical simulations. We first detail our simulation setup, followed by experimental results on the task field estimation and distributed task processing.

TABLE I: Simulation parameters.

Parameters	**Value**	**Parameter**	**Value**
Region size	$(5000\text{ m})^2$	Sub-region size	$(100\text{ m})^2$
H_c	100 m	H_s	50 m
MCA speed	25 m/s	MSA speed	$10-20$ m/s
Sub-region duration	$50-60$ s	Sub-region pause	$2-5$ s
B	0.2 Mbs	P	40 dBm
β	-50 dB	σ^2	-60 dBm
T_1	100	η_ℓ	$1e^{-4}$
T_2	200	α	$1e^{-2}$
ξ	$1/\delta$		

A. Simulation Setup

We implemented our simulations using Matlab. To simulate continuous time, we discretize it into 0.1s time steps. For Gaussian process regression, we use the Matlab Statistics and Machine Learning toolbox.

We consider a scenario with $N = 50$ MSAs that operate in a $(5000\text{ m})^2$ area at a fixed MSL altitude of $H_s = 50$ m. The MSAs surveil the region using a modified random waypoint model based on [26]. Each MSA starts at a randomly initialized location in the region and chooses a destination at random within the region. The MSA flies to this destination along the shortest path and then flies within a bounded subregion of $(100\text{ m})^2$ using a random waypoint trajectory for a period of time chosen uniformly at random between 50s and 60s. The MSA pauses for a random time between 2s and 5s, and repeats the entire process. The speed of each MSA, in m/s, is randomly selected from the interval $[10, 20]$.

As the MSAs fly, they collect sensing data. Rather than collecting data at a constant rate, the collection rate increases as the MSAs are close to points of interests, e.g., representing wildfire fronts. For the sake of comparison, we standardize the total number of tasks generated per second to 6×10^6 bps, and each MSA generates a number of tasks proportional to its distance from the point(s) of interest. We consider two scenarios:

1) **Fixed points of interest.** We use two points of interest at locations $\mathbf{p}_1$ and $\mathbf{p}_2$. The point locations $\mathbf{p}_1$ and $\mathbf{p}_2$ are selected uniformly at random and remained fixed for the duration of the simulation. The quantity of tasks generated at MSA n at time t is proportional to $\|\mathbf{w}_n(t) - \mathbf{p}_1\|^{-1.5} + \|\mathbf{w}_n(t) - \mathbf{p}_2\|^{-1.5}$.
2) **Moving point of interest.** We use a single point of interest, initially at location $\mathbf{p}(0) = [4500, 4500, 0]^T$, which moves 500m every 15s at an angle of -45° across the region. The quantity of tasks generated at MSA n at time t is proportional to $\|\mathbf{w}_n(t) - \mathbf{p}(t)\|^{-1.5}$.

For the task processing experiments, there are $M = 6$ MCAs. The MCAs fly at fixed MSL altitude of $H_c = 100$ m. At the beginning of each window, starting with window 2, the MCAs pause for 0.1s to execute our algorithm and determine their new positions. They then fly to their new positions at a constant speed of 25 m/s and hover at their new positions until the end of the time window. The simulation parameters are given in Table I.

B. Task Field Estimation

We first explore the accuracy of the task field estimation phase. The MSAs collect data for time window $(0, \Delta]$. We then sample $d_n(\Delta)$, the amount of tasks they collected over these Δ seconds, from all 50 MSAs. We use these observations to generate the estimate ρ via Gaussian process regression. In our solution, this estimate of ρ is used to determine the MCA positions for the time window $(\Delta, 2\Delta]$. We therefore compare the estimate ρ against the distribution of tasks collected in time window $(\Delta, 2\Delta]$. Since we are interested in studying the estimation accuracy for a single time window, we use fixed points of interest in these experiments.

For the purposes of comparison, we discretize the region into $(50\text{ m})^2$ cells. For each MSA n, we consider the cells that it visits in window $(\Delta, 2\Delta]$, and we assign an equal portion of $d_n(t)$ to each of these cells. The resulting discrete field $\mathbf{T} \in \mathbb{R}^{100\times100}$ is the *discretized task field* for window $(\Delta, 2\Delta]$. We generate the discretized task field estimate $\mathbf{E} \in \mathbb{R}^{100\times100}$ by computing the value of ρ at the center of each cell. To compare $\mathbf{T}$ and $\mathbf{E}$, we use the normalized mean-square error (NMSE) $\|\mathbf{E} - \mathbf{T}\|_F / (\max(\mathbf{T}) - \min(\mathbf{T}))$. Here, $\|\cdot\|_F$ denotes the Frobenius norm.

Fig. 2 shows the NMSE for various values of Δ. For each Δ, the NMSE is the average of three experiments. As can be seen, the NMSE increases as Δ increases. This is as anticipated since the smaller the value of Δ, the closer our approximation is to a continuous time model.

This trend is further corroborated in Fig. 3. In Figs. 3a and 3d, we show the observed values $d_n(\Delta)$, each in the cell where its corresponding MSA n was located at time Δ, for $\Delta = 10$s and $\Delta = 50$s, respectively. Figs. 3b and 3e show the corresponding estimates of the task fields. We can see

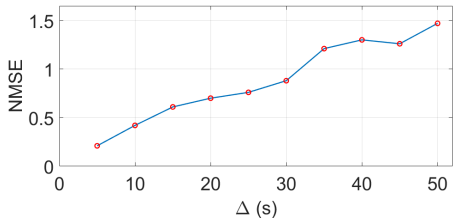


Fig. 2: Change in NMSE with increasing Δ.

that for both values of Δ, the estimate captures the locations and magnitudes of the observations. The estimated field is smooth; the benefit of this smoothing is that it can, in some part, account for the movement of the MSAs throughout the subsequent window. Figs. 3c and 3f show the discretized task fields for the subsequent window. We observe that the estimates reflect the locations of higher volumes of tasks, however, the magnitudes of the estimate are much larger than those of the discretized task fields. This is due to the fact that in our discretized task field, the MSA's tasks are split over all cells that it visits in the window. The magnitude discrepancy is larger for $\Delta = 50$s than $\Delta = 10$s because the MSAs move for a longer period of time and thus cover more cells. These results indicate that while the estimates differ in magnitude from the discretized task fields, they are still valuable for selecting the locations for MCAs to optimize the throughput.

C. Collaborative Task Processing

We next explore the performance of our full proposed solution. Based on the results of the previous section, we select two smaller values of Δ, $\Delta = 10$s and $\Delta = 20$s. Each experiment lasts 120s.

We evaluate three approaches:

1) **Baseline.** The MCA positions remained fixed for the duration of the experiment.
2) **Transmission rate maximization.** Every Δ seconds, the MCAs generate an estimate of ρ and then execute the transmission rate maximization algorithm and update their positions accordingly.
3) **Full solution.** Every Δ seconds, the MCAs generate an estimate of ρ and then execute the transmission rate maximization algorithm followed by the capacity balancing algorithm. They then update their positions accordingly.

For all three approaches, the MCAs are initially positioned in two rows of three, so that each MCA has the same size Voronoi region. For the baseline approach, each MSA continuously offloads to its closest MCA, provided that the MCA has capacity to process the workload. If the closest MCA does not have capacity, the MSA queues its data until the closest MCA has available capacity. Note that since the MSA moves throughout the experiment, the closest MCA may be different at different times. For the transmission rate maximization and full solution approaches, the MCAs start executing the algorithms after the first Δ seconds. At the start of each subsequent window, the MCAs pause for 0.1s to generate ρ and compute their new positions. They then move to those new positions. The MSAs continuously offload to the MCAs assigned to them by the Voronoi partitioning. If an MCA does not have sufficient capacity, the MSA queues any remaining tasks until it is assigned to an MCA that does have capacity as in the baseline approach.

TABLE II: Total tasks processed ($\times 10^8$) and warm start percent increase over baseline, with homogeneous capacities and fixed points of interest.

Approach	**$\Delta = 10$s**			**$\Delta = 20$s**		
	cold	**warm**	**% inc.**	**cold**	**warm**	**% inc.**
Baseline	4.54	4.14		4.54	3.75	
Rate Max.	4.56	4.17	1	4.54	3.75	0
Full Solution	5.29	4.90	18	5.01	4.22	13

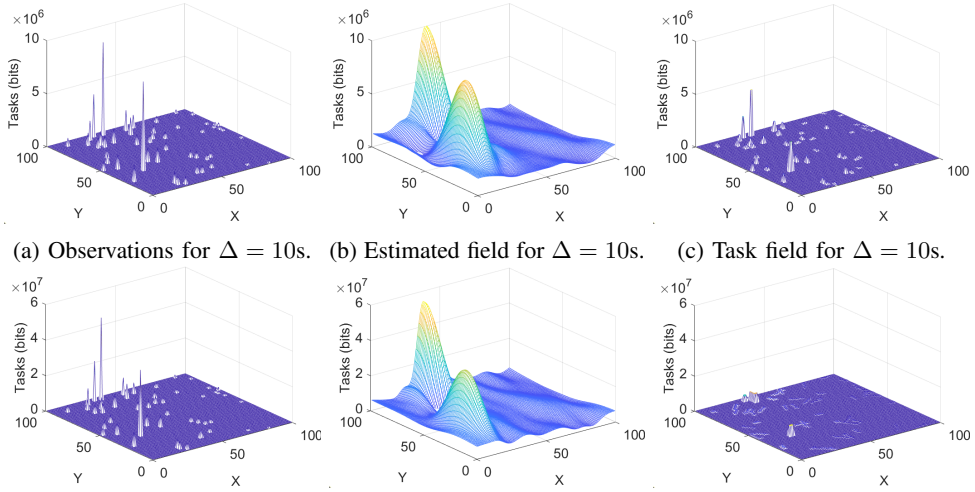
TABLE III: Total tasks processed ($\times 10^8$) and warm start percent increase over baseline, with heterogeneous capacities and fixed points of interest.

Approach	**$\Delta = 10$s**			**$\Delta = 20$s**		
	cold	**warm**	**% inc.**	**cold**	**warm**	**% inc.**
Baseline	3.88	3.54		3.88	3.21	
Rate Max.	3.87	3.53	0	3.86	3.19	-1
Full Solution	4.78	4.45	26	4.45	3.78	18

For ease of comparison, we set the total processing capacity of the system to be 6×10^6 bps so that it matches the total task generation rate. We study two scenarios, a *homogenous capacity* scenario, where each MCA has the same capacity of 10^6 bps, and a *heterogeneous capacity* scenario, where MCA 1 has capacity 2×10^6 bps, MCAs 2, 3, and 4 have capacities 10^6 bps, and MCAs 5 and 6 have capacities 0.5×10^6 bps.

For each experiment, we calculate the total amount of tasks that are processed in two ways. We show the *cold start total*, where we compute the total number of tasks processed over the entire 120s. In all three approaches, the MCAs have the same positions, and therefore the same amount of tasks processed in the first window, and so we also calculate the *warm start total*, which excludes the first Δ seconds, so as to only include the time in which the MCAs adapt their positions and task assignments. We also calculate the percentage improvement of our full solution over the baseline. All results are the averages over three experiments.

1) Fixed points of interest: Tables II and III show the results for the homogeneous and heterogeneous capacities, respectively, for the fixed points of interest scenario. The transmission rate maximization approach does not show much benefit over the baseline, indicating that optimizing for transmission rate alone is not sufficient to maximize the system throughput. We observe that the full solution yields significant improvement in total tasks processed over the baseline. Further, the benefit is greater for the smaller value of Δ, with an 18% improvement over the baseline for the warm start total with homogeneous capacities, and a 26% improvement for heterogeneous capacities. These results show the importance of aligning the task assignment with the MCA capacities—a novel feature of our proposed solution. We also observe that



(a) Observations for $\Delta = 10$s. (b) Estimated field for $\Delta = 10$s. (c) Task field for $\Delta = 10$s.

(d) Observations for $\Delta = 50$s. (e) Estimated field for $\Delta = 50$s. (f) Task field for $\Delta = 50$s.

Fig. 3: Examples of the observed task queues, estimated task fields from these observations $\mathbf{E}$, and the discretized task fields $\mathbf{T}$ for the window, for $\Delta = 10$s and $\Delta = 50$s.

TABLE IV: Total tasks processed ($\times 10^8$) and warm start percent increase over baseline, with homogeneous capacities and moving point of interest.

Approach	**$\Delta = 10$s**			**$\Delta = 20$s**		
	cold	**warm**	**% inc.**	**cold**	**warm**	**% inc.**
Baseline	4.09	3.75		4.09	3.42	
Rate Max.	4.25	3.91	4	4.25	3.58	5
Full Solution	4.88	4.55	21	4.81	4.14	21

TABLE V: Total tasks processed ($\times 10^8$) and warm start percent increase over baseline, with heterogeneous capacities and moving point of interest.

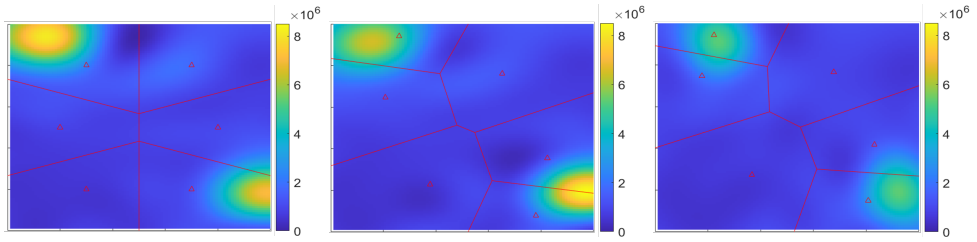
Approach	**$\Delta = 10$s**			**$\Delta = 20$s**		
	cold	**warm**	**% inc.**	**cold**	**warm**	**% inc.**
Baseline	3.33	3.06		3.33	2.80	
Rate Max.	3.52	3.25	6	3.43	2.90	3
Full Solution	4.20	3.93	28	4.06	3.53	26

in all cases, the total amount of tasks processed is less than the total amount of tasks generated (7.2×10^8 bits for cold start; 6.6×10^8 bits and 6.0×10^8 bits for warm start for $\Delta = 10$s and 20s, respectively). We believe this is due to two factors. First, the MCAs spend part of each window moving to their new positions. During this transition period, the Voronoi partitioning is sub-optimal with respect to transmission rates and capacities. Second, any inaccuracies in the estimate of ρ lead the MCAs to select positions that are less compatible with the true distribution of tasks. Thus, there is opportunity to improve on our solution with more sophisticated task field estimation techniques.

Fig. 4 shows the evolution of the MCA locations and the Voronoi partitioning over multiple time windows for the homogeneous capacity scenario. The MCA locations and region boundaries are superimposed on a heat map of ρ for that window. Initially (window 1), each MCA has a similar-sized Voronoi region; however, the volume of tasks in each cell is very different. In window 4, the Voronoi partitioning more accurately reflects the task distribution, and in window 8, the Voronoi partitioning is further adapted to minor changes in the tasks distribution that result from the MSAs' mobility and residual queued tasks.

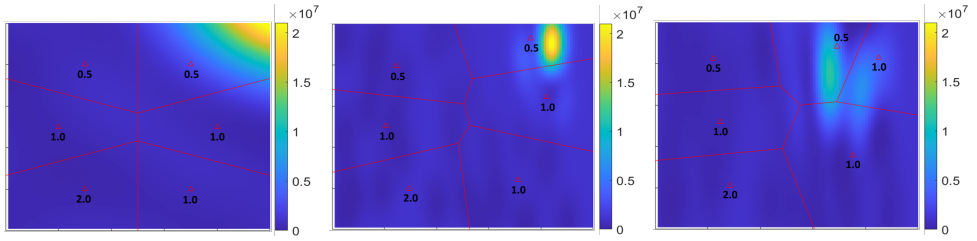
2) Moving point of interest: Tables IV and V show the results for the homogeneous and heterogeneous capacities, respectively for the moving point of interest. There is a small increase in the total number of tasks processed for the transmission rate maximization approach for both capacity types. Since the task field changes as the point of interest moves, it is intuitive that adapting the MCA positions should improve the overall transmission rate, thus improving the system throughput. Our full solution yields significant improvement over the baseline. For MCAs with homogeneous capacities, our full solution achieved a 21% improvement in the warm start total for both $\Delta = 10$s and $\Delta = 20$s. For the heterogeneous capacities, we observe improvements of 28% and 26% for $\Delta = 10$s and $\Delta = 20$s, respectively. This is a greater improvement than observed for the fixed points of interest, which shows a key benefit of our method in that it adapts to both small and large changes in the task field. As in the previous set of experiments, no approach processes all of the generated tasks; we believe the same reasons apply to this scenario. We also observe that in most cases, our full solution performs better with $\Delta = 10$s than with $\Delta = 20$s. This is expected because the estimate of ρ is more accurate for the smaller Δ. The improvement over $\Delta = 20$s is not very large though, indicating that it is possible to save on the motion energy of the MCAs without much decay in the system throughput.

Fig. 5 shows the evolution of the MCA locations and the Voronoi partitioning over multiple time windows for the heterogeneous capacity scenario with a moving point of interest.



(a) Window 1. (b) Window 4. (c) Window 8.

Fig. 4: Examples of homogeneous MCAs adjusting their positions to the task field for fixed points of interest for $\Delta = 10$s. The triangles represent the MCAs. The background color represents the intensity of the estimated task field.



(a) Window 1. (b) Window 4. (c) Window 8.

Fig. 5: Examples of heterogeneous MCAs adjusting their positions to the task field for a moving point of interest for $\Delta = 10$s. The triangles represent the MCAs. The background color represents the intensity of the estimated task field.

As the point of interest moves, the task field distribution changes accordingly, and in response, the MCAs adapt their positions and Voronoi regions. As expected, the MCA position changes are more pronounced than in the fixed points of interest setting since the changes in the task field are more significant. These figures illustrate the power of our method to autonomously adapt to changes in the task demand.

V. Related Work

Several works have proposed solutions using a single UAV as an MEC server for users on the ground. The majority of these [5], [13]–[15] assume that user demand is static and known to the UAV, and they propose methods to design a UAV trajectory that minimizes the motion energy cost while also optimizing for offloading latency or rate. [16] proposes a solution to optimize the UAV trajectory when the user demand distribution changes but in a constrained and known way along straight road segments. In contrast, in our setting, the demand distribution is continuously changing, with no constraints. Finally, [7] proposes a solution for a single UAV system that can adapt to changes in the user demand, however, the limitation to a single MEC UAV simplifies the problem setting over the one we study.

Several multi-UAV MEC systems for ground users have also been proposed. These works focus on designing the UAV trajectories and task assignments to optimize for similar performance measures as the single UAV systems. One line of work assumes that the user demand is fixed and known. Due to the intractability of the problem, these works utilize various relaxation approaches and meta-optimization methods such convex relaxation [11], [17], Deep Reinforcement Learning [8], and Differential Evolution [12]. The solutions proposed in these works are all centralized, and this centralization, along with the assumption about fixed demand, make them unsuitable for our setting. A few works support dynamic user demand, e.g., [18], which also proposes a solution based on Differential Evolution, [9], which transforms the problem to a maximum clique problem, and [10], which formulates the problem as an NP-Hard bin packing problem and provides an online heuristic solution. The algorithms in these three works are centralized, which limits their applicability in disaster scenarios that do not have the infrastructure to support centralized computation.

The recent work by Chen et al. [6] studies the problem of task offloading from UAVs to MEC servers on ground vehicles. The work assumes that the trajectories of the UAVs and ground vehicles are known and cannot be altered, and the UAVs pay the ground vehicles for use of the MEC servers. They present a centralized solution based on maximal matching that optimizes for the UAV's satisfaction and the ground vehicle's payoff. The problem considered in this work is distinctly different from ours in that the MEC server trajectories cannot be controlled.

Finally, a work that is similar in spirit to ours is [27], which considers a setting where UAVs serve as aerial base stations for a time-varying continuous distribution of ground user demand. The authors present a distributed algorithm that adapts the UAV positions so as to maximize the network coverage of the ground users. While this work also uses Gaussian process regression to estimate user demand, it assumes the demand itself is a continuous distribution, whereas the demand in our problem is generated by individual UAVs, and we demonstrate that this demand can be approximated using Gaussian process

regression. Further, in the coverage problem, the UAVs are assumed to have unlimited capacity and can cover any user demand within a certain range. This is in contrast with our setting, where the MCAs have throughput capacities that must be respected when determining the MCA positions. Thus, our objective is mathematically distinct from [27], and further, the inclusion of the capacity constraints makes the problem even more challenging, necessitating our multi-phase solution.

VI. Conclusion

We have presented a novel distributed solution for collaborative task processing in UAV MEC networks. One key contribution of this work is to show that the workload can be efficiently approximated by a continuous task field. By adopting this continuum model, we are able to devise a distributed approach to maximize the system throughput; the MCAs adaptively update their positions and the task assignments to maximize the task transmission rates while meeting their processing capacities. Our solution shows up to 28% improvement over a non-adaptive baseline strategy in experiments.

This work demonstrates the potential of utilizing continuous models for problems like task offloading, which are traditionally treated as discrete optimization problems. As discussed in Section IV-B, the accuracy of our task field estimation decreases as the time window Δ increases. To improve this estimation accuracy, in future work, we plan to investigate more sophisticated estimation techniques that incorporate MSA mobility prediction. In addition, we plan to extend our approach to optimize for the MCA motion energy and use techniques like work stealing to balance workload. Finally, we aim to relax the model to a continuous time approach, rather than a tumbling window approach, so that the MCAs move at independently determined times to optimize for the combination of energy and system throughput.

Acknowledgment

This research was partially supported by National Science Foundation Grant CNS-1816307 and Air Force Office of Scientific Research DDDAS Grant FA9550-19-1-0054.

References

[1] S. Seneviratne, X. Zhang, M. Adnan, W. Badi, C. Dereczynski, A. Di Luca, S. Ghosh, I. Iskandar, J. Kossin, S. Lewis, F. Otto, I. Pinto, S. Vicente-Serrano, M. Wehner, and B. Zhou, *2021: Weather and Climate Extreme Events in a Changing Climate.* Cambridge University Press, 2021.

[2] S. Hayat, E. Yanmaz, and R. Muzaffar, "Survey on unmanned aerial vehicle networks for civil applications: A communications viewpoint," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 4, pp. 2624–2661, 2016.

[3] M. Mozaffari, W. Saad, M. Bennis, Y.-H. Nam, and M. Debbah, "A tutorial on UAVs for wireless networks: Applications, challenges, and open problems," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 3, pp. 2334–2360, 2019.

[4] S. Verykokou, A. Doulamis, G. Athanasiou, C. Ioannidis, and A. Amditis, "UAV-based 3D modeling of disaster scenes for urban search and rescue," in *Proc. IEEE Int. Conf. Imaging Syst. Tech.*, 2016, pp. 106–111.

[5] M. Li, N. Cheng, J. Gao, Y. Wang, L. Zhao, and X. Shen, "Energy-efficient UAV-assisted mobile edge computing: Resource allocation and trajectory optimization," *IEEE Trans. Veh. Technol.*, vol. 69, no. 3, pp. 3424–3438, 2020.

[6] W. Chen, Z. Su, Q. Xu, T. H. Luan, and R. Li, "VFC-based cooperative UAV computation task offloading for post-disaster rescue," in *Proc. IEEE Conf. Comput. Commun.*, 2020, pp. 228–236.

[7] Y. Liu, K. Xiong, Q. Ni, P. Fan, and K. B. Letaief, "UAV-assisted wireless powered cooperative mobile edge computing: Joint offloading, CPU control, and trajectory optimization," *IEEE Internet Things J.*, vol. 7, no. 4, pp. 2777–2790, 2020.

[8] N. Zhao, Z. Ye, Y. Pei, Y.-C. Liang, and D. Niyato, "Multi-agent deep reinforcement learning for task offloading in UAV-assisted mobile edge computing," *IEEE Trans. Wireless Commun.*, pp. 1–1, 2022.

[9] Z. Liao, Y. Ma, J. Huang, J. Wang, and J. Wang, "HOTSPOT: A UAV-assisted dynamic mobility-aware offloading for mobile-edge computing in 3-D space," *IEEE Internet Things J.*, vol. 8, no. 13, pp. 10 940–10 952, 2021.

[10] J. Wang, K. Liu, and J. Pan, "Online UAV-mounted edge server dispatching for mobile-to-mobile edge computing," *IEEE Internet Things J.*, vol. 7, no. 2, pp. 1375–1386, 2020.

[11] X. Huang, X. Yang, Q. Chen, and J. Zhang, "Task offloading optimization for UAV-assisted fog-enabled Internet of Things networks," *IEEE Internet Things J.*, vol. 9, no. 2, pp. 1082–1094, 2022.

[12] L. Yang, H. Yao, J. Wang, C. Jiang, A. Benslimane, and Y. Liu, "Multi-UAV-enabled load-balance mobile-edge computing for IoT networks," *IEEE Internet Things J.*, vol. 7, no. 8, pp. 6898–6908, 2020.

[13] Q. Hu, Y. Cai, G. Yu, Z. Qin, M. Zhao, and G. Y. Li, "Joint offloading and trajectory design for UAV-enabled mobile edge computing systems," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 1879–1892, 2019.

[14] F. Zhou, Y. Wu, R. Q. Hu, and Y. Qian, "Computation rate maximization in UAV-enabled wireless-powered mobile-edge computing systems," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 9, pp. 1927–1941, 2018.

[15] Z. Na, Y. Liu, J. Shi, C. Liu, and Z. Gao, "UAV-supported clustered NOMA for 6G-enabled Internet of Things: Trajectory planning and resource allocation," *IEEE Internet Things J.*, vol. 8, no. 20, pp. 15 041–15 048, 2021.

[16] L. Zhang, Z. Zhao, Q. Wu, H. Zhao, H. Xu, and X. Wu, "Energy-aware dynamic resource allocation in UAV assisted mobile edge computing over social Internet of Vehicles," *IEEE Access*, vol. 6, pp. 56 700–56 715, 2018.

[17] H. Wang, H. Ke, and W. Sun, "Unmanned-aerial-vehicle-assisted computation offloading for mobile edge computing based on deep reinforcement learning," *IEEE Access*, vol. 8, pp. 180 784–180 798, 2020.

[18] H. Yan, W. Bao, X. Zhu, J. Wang, and L. Liu, "Data offloading enabled by heterogeneous UAVs for IoT applications under uncertain environments," *IEEE Internet Things J.*, pp. 1–1, 2022.

[19] T. X. Tran and D. Pompili, "Joint task offloading and resource allocation for multi-server mobile-edge computing networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 1, pp. 856–868, 2019.

[20] K. Cheng, Y. Teng, W. Sun, A. Liu, and X. Wang, "Energy-efficient joint offloading and wireless resource allocation strategy in multi-MEC server systems," in *Proc. IEEE Int. Conf. Commun.*, 2018, pp. 1–6.

[21] E. Schulz, M. Speekenbrink, and A. Krause, "A tutorial on Gaussian process regression: Modelling, exploring, and exploiting functions," *J. Math. Psychol.*, vol. 85, pp. 1–16, 2018.

[22] P. Bianchi and J. Jakubowicz, "Convergence of a multi-agent projected stochastic gradient algorithm for non-convex optimization," *IEEE Trans. Autom. Control*, vol. 58, no. 2, pp. 391–405, 2012.

[23] J. Zeng and W. Yin, "On nonconvex decentralized gradient descent," *IEEE Trans. Signal Process.*, vol. 66, no. 11, pp. 2834–2848, 2018.

[24] L. Blair, C. A. Varela, and S. Patterson, "A continuum approach for collaborative task processing in UAV MEC networks," *arXiv preprint arXiv:2206.02950*, 2022.

[25] H. Sayyaadi and M. Moarref, "A distributed algorithm for proportional task allocation in networks of mobile agents," *IEEE Trans. Autom. Control*, vol. 56, no. 2, pp. 405–410, 2011.

[26] A. Bujari, C. T. Calafate, J.-C. Cano, P. Manzoni, C. E. Palazzi, and D. Ronzani, "Flying ad-hoc network application scenarios and mobility models," *Int. J. Distrib. Sens. Netw.*, vol. 13, no. 10, p. 1550147717738192, 2017. [Online]. Available: https://doi.org/10.1177/1550147717738192

[27] T. Kimura and M. Ogura, "Distributed collaborative 3D-deployment of UAV base stations for on-demand coverage," in *Proc. IEEE Conf. Comput. Commun.*, 2020, pp. 1748–1757.