Air-to-Air Collaborative Learning: A Multi-Task Orchestration in Federated Aerial Computing

Uchechukwu Awada*, Jiankang Zhang†, Sheng Chen‡, Shuangzhi Li*
*School of Information Engineering, Zhengzhou University, Zhengzhou 450001, China
†Department of Computing and Informatics, Bournemouth University, Poole, BH12 5BB, UK
‡School of Electronics and Computer Science, University of Southampton, Southampton SO17 1BJ, UK
awada@gs.zzu.edu; jzhang3@bournemouth.ac.uk; sqc@ecs.soton.ac.uk; ielsz@zzu.edu.cn

***Abstract*—Recent research on edge computing (EC) has proposed federated or collaborative learning technique, where machine learning models are shared among participating edge deployments, thereby benefiting from all available datasets without exchanging them. In addition, EC systems are currently exploiting attaching portable edge devices on drones for data processing close to the sources, to achieve high performance, fast response times and real-time insights. Existing research lack the potential to federate edge resources and manage corresponding service entities running across multiple drones, thus resulting to sub-optimal performance. Therefore, we introduce *AerialEdge*, a federated learning-based orchestration framework for a federated aerial EC system. We propose a federated multi-output linear regression models to estimate multi-task resource requirements and execution time, to select the closest drone deployment having congruent resource availability and flight time to execute ready tasks at any given time. For better utilization of resources, we propose a variant bin-packing optimization approach through *gang-scheduling* of multi-dependent containerized tasks that *co-schedules* and *co-locates* tasks tightly on nodes to fully utilize available resources. Extensive experiments on real-world data-trace from Alibaba cluster trace with information on task dependencies show the effectiveness, fast executions, and resource efficiency of our approach.**

***Index Terms*—Edge computing, dependency-aware, federated learning, edge federation, execution time, resource efficiency**

I. Introduction

Emerging technologies, such as connected and autonomous vehicles (CAVs), healthcare IoT systems, real-time augmented reality, smart cities, Industry 4.0, etc, rely on edge computing resources, to offload their computational intensive tasks, improve response times and achieve real-time insights. To this end, Cloud Computing providers, such as Amazon Web Services (AWS), Microsoft Azure, Alibaba Cloud, Google Cloud, Dell Technologies Cloud, IBM Cloud, etc., have recently introduced various portable edge devices and begun offering cloud computing services directly on these edge devices. Examples of these portable edge devices include, AWS Snowcone[1] (with 8TB, 2vCPU and 4GiB of resources), HIVECELL[2] (with 500GB 6vCPU 8GiB of resources), etc. Each weighs about $2.1 \sim 3$ kg, and capable of accelerated artificial intelligent (AI) inferencing.

Consequently, edge computing (EC) systems are recently exploiting attaching these portable edge devices on low altitude platform (LAP) unmanned aerial vehicles (UAVs) or drones as aerial deployments, to execute complex resource-hungry use cases. A state-of-the-art drone technology, called *Drone-in-a-box*[3], is most suitable for aerial EC deployment. A drone-in-a-box system can be deployed autonomously from a box that serves as a landing pad and charging base. After executing all its tasks, it returns to its box. However, a typical drone has a limited flight time due to power factor, which can lead to loss of job if it is not taking into consideration [1]. The critical issue is how to optimize both the drones' flight time and application execution on the attached edge device(s) in a timely manner, without jeopardizing application performance.

Most recently, research on EC has proposed a machine learning (ML) technique that trains an algorithm across multiple edge deployments holding local data samples, without exchanging them [2], [3]. This is called *Federated or Collaborative Learning*. Models learned in such way promise of greatly improving usability by powering more functionalities and intelligent applications [4]. However, efficient orchestration of complex dependencies among tasks in such individual deployments is challenging due to constrained resource capabilities, vendor lock-in, availability factors, etc. Existing research on edge federated/collaborative learning, i.e., [2], [3] do not consider the ability of keeping edge resources running across different edge deployments in a single pool, such that these resources can be holistically managed and controlled from a single federated plane, vendor lock-in situations can be eliminated, and applications can be deployed dynamically across the resources. Note that edge FL technique produces and shares a single global inference model among participating edge through a remote/central server, while the merging of resources among participating edge, such that these resources are holistically managed and controlled from a single federated plane is refereed to as *Edge Federation (EF)*. EF minimizes latency by serving users from the edge deployment that is the closest to them [5].

Nevertheless, to execute complex applications (i.e., multi-dependent tasks, where each task has diverse resource request) pose several challenges: (i) Given a federated clusters running

[1]https://aws.amazon.com/snowcone/
[2]https://hivecell.com/hardware/
[3]https://en.wikipedia.org/wiki/Drone_in_a_Box

across multiple drones, to automatically decide where a job or multi-task should be deployed is a challenging task. Previous works [6], [7] assume that each server can only execute one task or job at any time and schedules each task individually, which could result to higher communication overhead. (ii) The default schedulers deploy tasks randomly on nodes (virtual machines) that have sufficient resource availability without considering dependent tasks, which results in longer execution time, resource wastage through underutilized nodes, and a reduction in the number of tasks that can be executed, given the available resources. In addition, it does not pack tasks tightly on available resources in order to achieve high utilization.

This motivates our research to address these limitations and to extend the state-of-the-arts by proposing AerialEdge, which considers jointly task dependencies, heterogeneous resource demands, and flight times for drones in a federated autonomous aerial EC-enabled learning system. Specifically, we propose a federated multi-output linear regression models to estimate multi-task resource requirements and execution time, and a multi-task dispatching policy called *Closest* to select the closest drone deployment having congruent resource availability and flight time to execute ready tasks at a given time and to autonomously deploy the selected drone to the needed location. We also propose a variant bin-packing optimization through gang-scheduling of multi-dependent tasks that co-schedules and co-locates multi-tasks tightly on nodes to fully utilize available resources. We show that the proposed AerialEdge can minimize the actual completion time of tasks using minimum resources, such that the actual completion time is much less than the drones' flight time. In summary, to achieve our AerialEdge implementation, we address the following critical areas:

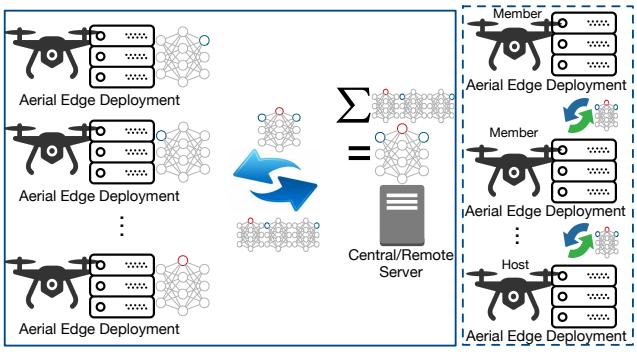
- Federated aerial edge state service: to stream all changes in the drones and the attached edge devices across the federated deployments, and to enable us to query the state of drones, in terms of flight time availability, and resource availability, so that informed decisions can be made on multi-task orchestration and co-location.
- Dynamic optimization strategy: to efficiently manage tasks across the federated edge resources or clusters, and to enable us to co-locate multi-tasks for fully utilizing available resources [8]–[10].
- Extensive experimentation and comparison: to evaluate AerialEdge on the real-world Alibaba Cluster data trace, and to compare it with existing deployment approaches.

II. Preliminaries and Our Motivation

In the light of the prior research, we first present some preliminaries and related work, and discuss our motivation.

A. Edge Federated Learning

The main goal of edge FL is to collaboratively learn a model from data stored across distributed deployments $\mathcal{D} = \mathcal{D}_1, ..., \mathcal{D}_N$, where each dataset $\mathcal{D}_i = \{(\mathrm{x}_i, \mathrm{y}_i)\}_{i=1}^{n}$ contain a



(a) FL system (b) EF system

Fig. 1: (a) The architecture of edge federated learning system, (b) The architecture of a merged edge-enabled federated learning system.

D-dimensional tensor[4] of data feature $\mathrm{x}_i \in \mathbb{R}^{1\times d}$, and C-dimensional tensor data label $\mathrm{y}_i \in \mathbb{R}^{1\times c}$. A conventional method is to aggregate the datasets in one server or datacenter, i.e., $\mathcal{D} = \mathcal{D}_1 \cup ... \cup \mathcal{D}_N$ and utilize it to train a model θ_{S}. In edge FL, the participating members/deployments independently train their models θ_{L_i}, based on their local dataset, , as shown in Fig. 1(a). Then at time $t \geq 0$, the participating deployments send their models to the remote/central server, i.e., $\sum_{i=1}^{N} \theta_{L_i}$, where the global update θ_G is computed by aggregating all the deployments models:

$$\theta_G^t = \sum_{i=1}^{N} \theta_{L_i}, \tag{1}$$

In return, the global model is distributed to the participants, where they use it to update their current models. Let the updated local models be $\theta_{L_1}^t, \theta_{L_2}^t, ..., \theta_{L_N}^t$, therefore the update at the ith deployment can be expressed as $\Theta_{L_i}^t = \theta_{L_i}^t - \theta_{L_i}$. At time $t+1$, the updates from each participant is sent back to the central server, i.e., $\sum_{i=1}^{N} \Theta_{L_i}$, where a global update is computed:

$$\theta_G^{t+1} = \theta_G^t + \sum_{i=1}^{N} \Theta_{L_i}. \tag{2}$$

For dataset $\mathcal{D}_i$ at deployment i, the learning problem is to minimize the loss function:

$$f(\Theta_L^*) = \underset{\Theta_L \in \mathbb{R}^{d\times c}}{\arg\min} \frac{1}{2n} \sum_{i=1}^{n} \|\mathrm{x}_i \Theta_L - \mathrm{y}_i\|_2^2 + \frac{\lambda}{2} \|\Theta_L\|_F^2, \tag{3}$$

where λ and $\|\cdot\|_F$ denote the regularization parameter and Frobenius norm respectively. Equation (3) is commonly solved using gradient-descent techniques, where the model is updated sequentially until convergence, i.e., $\Theta_L^{t+1} = \Theta_L^t - \eta(\frac{g}{n} + \lambda\Theta_L^t)$, where η denotes the learning rate, $g = \frac{1}{n}\mathbf{X}^T(\mathbf{X}\Theta_L^r - \mathbf{Y})$ denotes the gradient of the loss function, $\mathbf{X} = [\mathrm{x}_1^T, ..., \mathrm{x}_n^T]^T$

[4]In ML, a tensor is a multidimensional array and is a generalization of matrices and vectors.

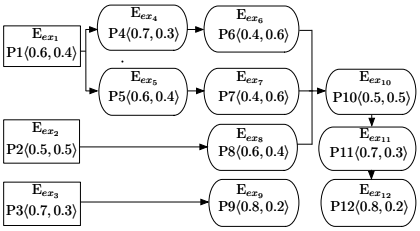


Fig. 2: DAG of a video processing job.

and $\mathbf{Y} = [\mathrm{y}_1^T, ..., \mathrm{y}_n^T]^T$ denote the feature set and label set respectively.

Although FL promises to produce a high-quality global model, however there is no resource coordination among the participants.

B. Merged Edge-Enable Federated Learning

This research focuses on Job orchestration in merged/federated edge computing enabled learning system, as shown in Fig 1(b). In the merged edge-enabled FL approach, participating deployments can potentially merge their resources, such that all the resources running across different edge deployments can be holistically managed and controlled, and applications can be deployed dynamically across the resources. In a merged edge clusters setup, the Federation Control Plane (FCP) is deployed on one of the clusters which serves as the `host cluster`. Participating clusters or members can be added or removed from the FCP. We assume that datasets $\mathcal{D}$ across distributed deployments share the same feature and label space, i.e.,

$$\mathrm{x}_i = \mathrm{x}_j,\ \mathrm{y}_i = \mathrm{y}_j,\ \forall \mathcal{D}_i, \mathcal{D}_j,\ i \neq j. \tag{4}$$

First, we propose a merged edge-enabled FL system, whereby the participating deployments can collaboratively learn a shared global model through the `host cluster`, thereby eliminating the need for a central server (i.e., the `host cluster` serves as the models aggregator). Secondly, we propose the use of this model, i.e., a multi-output linear regression model, to estimate multi-task resource requirement and execution time, for the selection of the closest deployment within the federation having congruent resource availability and flight time to execute ready tasks at any given time.

C. Task Dependency-Awareness

Dependency-awareness is critical for achieving efficient multi-task orchestration, i.e., dispatching and co-location. Most of the batch workloads of Alibaba cluster trace[5] for example are directed acyclic graphs (DAGs), and only some of them are independent. The average Alibaba cluster trace job has a dependency depth of 10. A job is typically consisted of several tasks whose dependencies are expressed by DAG. Clearly, if a task A is depending on task B, then task A cannot start until all the instances of task B are completed. For example, the DAG of a video processing job is shown in Fig. 2. The Job consists of 12 tasks. Knowledge about task characteristics, such as resource demands and dependencies, is necessary to pack or co-locate tasks effectively in a node or cluster, ultimately to minimize the response times and improve resource utilization [8], [9], [11]. Hence a key objective is to reduce the execution time of such tasks and to improve resource utilization by considering the inter-task dependency and resource demands.

[5]https://github.com/alibaba/clusterdata/blob/master/cluster-trace-v2018/trace-2018.md

D. Our Motivation

To illustrate the advantage of AerialEdge, we show a motivating example using Fig. 2. It shows each task of a video processing job, with its actual execution time E_{ex_i} and resource demand (CPU and memory $\langle c, m \rangle$). Our aim is to deploy the job on a node with requisite available resources, such that dependent tasks can communicate faster, compared to other deployments across different nodes. Here, we assume that the node has the requisite available resources to accommodate all the tasks, i.e., $\langle 8, 5 \rangle$. We illustrate the scheduling approach AerialEdge and its execution time together with three other state-of-the-art approaches, namely, Spear [11], Graphene [12] and Tetris [10], as well as the random approach. Our AerialEdge achieves the lowest execution time of $\sum_{i=1}^{n} E_{ex_i}/n$ (n is the number of tasks), due to the following reasons: (i) our approach utilizes gang scheduling [13], which co-schedules **all the tasks at a time**, and (ii) our packing strategy explores the available nodes to find the best one which has requisite available resources (CPU and memory) to execute all the tasks by packing or co-locating them tightly on the node. By contrast, Spear and Tetris deploy the same tasks individually or in parts, resulting in execution times of $\sum_{y=1}^{n} E_{ex_y} + \sum_{z=1}^{m} \sum_{i=1}^{k} E_{ex_{iz}}/k$ and $\sum_{z=1}^{m} \sum_{i=1}^{n} E_{ex_{iz}}/n$, respectively. In particular, Spear picks tasks along the critical path (CP) in the DAG. The CP of a task is the longest path from the task to the output. As an example, given a job with 100 DAGs, Spear deploys about 15% of the tasks at a time. Tetris on the other hand does not consider the task dependencies. It deploys at least 50% of any given tasks at a time and focuses on packing tasks on nodes to achieve high resource utilization. Graphene, a state-of-the-art dependency-aware scheduler, considers both task dependencies and resource packing. It first co-schedules some tasks identified as *troublesome tasks* and then places the rest of the tasks afterwards, resulting in an execution time of $\sum_{x=1}^{n} E_{ex_x} + \sum_{z=1}^{m} \sum_{i=1}^{k} E_{ex_{iz}}/k$. The random approach deploys a task randomly to any available node, and assumes a node can only execute a task at a time, resulting in an execution time of $\sum_{i=1}^{n} E_{ex_i}$. Generally, delay in scheduling dependent tasks directly impacts job completion time, and utilizing gang scheduling is beneficial for overall performance.

III. Proposed AerialEdge

In this section, we detail our proposed AerialEdge for achieving high resource utilization and minimizing the execution times of applications deployed in federated aerial edge resources. Our system model is depicted in Fig. 3.

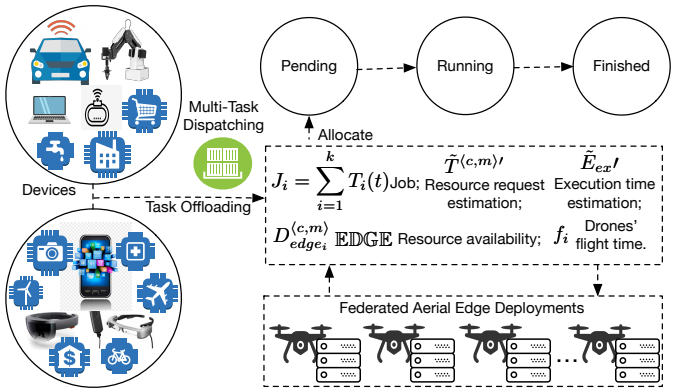


Fig. 3: Orchestration overview of AerialEdge.

A. System Model

Federated learning takes advantage of historical data across participating deployments to produce a high-level model. As multi-dependent containerized applications are admitted into the system, their resource requirements and execution time are estimated using linear regression model, in this case a *multi-output linear regression model*. The model is run on the multi-task features $\mathbb{F}$, such as type of tasks ϵ, dependency depth γ, data size δ, to estimate the values, i.e.,

$$\Theta(\mathbb{F}) = [\{\tilde{T}^{\langle c,m \rangle}\}_{i=1}^{\gamma}, \{\tilde{E}_{ex}\}_{i=1}^{\gamma}], \tag{5}$$

where $\tilde{T}^{\langle c,m \rangle}$ is the estimated resource requirements (in terms of CPU and memory) and $\tilde{E}_{ex}$ is the estimated execution time, and then to deploy it to the **closest** edge with congruent resource availability and flight time. To build this predictor Θ, we train a ML model based on Keras[6] with historical data from previously executed tasks/jobs. Keras is a library which wraps TensorFlow[7] complexity into simple and user-friendly API. The advantage of using containers to host applications at the edge is that these applications can be executed within its estimated resource requirement $\langle \delta, c, m \rangle$, where δ denotes the size of data, c denotes CPU requirement and m denotes memory requirement, in any edge deployment regardless of the resource type, configuration or vendor/provider.

Therefore, given a federated aerial edge deployment $\mathbb{EDGE}$, where each individual edge deployment D_{edge_i} is a cluster of container-instances (i.e., edge device(s) with virtualized container-optimized nodes) with total resource capacity or availability $D_{edge_i}^{\langle \delta,c,m \rangle}$, our aim is to deploy ready applications, $\mathbb{C}$, to the **closest** drone deployment with minimum flight time f_i^l from its box to the location l, having sufficient flight time f_i and resource capacity or availability to execute the tasks, such that the tasks are executed concurrently, namely,

$$\mathbb{C} \Rightarrow D_{edge_{i^\star}}, \tag{6}$$

where

$$D_{edge_{i^\star}} = \arg \min_{D_{edge_i} \in \mathbb{EDGE}} \left\{ f_i^l : f_i^l < f_i, D_{edge_i}^{\langle \delta,c,m \rangle} \text{ sufficient} \right\}. \tag{7}$$

[6]https://keras.io/
[7]https://www.tensorflow.org/

Existing works on UAV or drone-based task offloading and execution in EC systems [14], [15] do not consider drones' flight time and assume a drone can fly for unlimited amount of time, which can lead to loss of job due to drones' limited flight time [1]. Therefore, given a federated edge deployment $\mathbb{EDGE}$ and a set $\mathbb{C}$ of inter-dependent containerized applications, where each application $T \in \mathbb{C}$ serves as a task with its estimated resource demand and execution time denoted as $\tilde{T}^{\langle c,m \rangle}$ and $\tilde{E}_{ex}$ respectively, our goal is to use the estimated values for all the tasks to select an edge having congruent resource availability and flight time, such that we can intelligently schedule the tasks to minimize the actual execution time as well as to achieve high resource utilization efficiency.

For a task T, let E_s and E_c denote its actual starting and completion times, respectively. Therefore, the actual execution time of T is:

$$E_{ex} = E_c - E_s. \tag{8}$$

AerialEdge utilizes the gang scheduling [13] strategy to co-schedule all ready applications at a time. Hence the aggregate execution time of a multi (n)-task $\mathbb{C}$ is given as $\sum_{i=1}^{n} \frac{E_{ex_i}}{n}$. The federated edge system $\mathbb{EDGE}$ consists of all N participating individual edge deployments D_{edge_i}, $1 \leq i \leq N$, i.e.,

$$\mathbb{EDGE} = \sum_{i=1}^{N} D_{edge_i}. \tag{9}$$

Given a cluster of container-instances or nodes I in each deployment D_{edge_i}, let $I_i^{\langle c,m \rangle}$ denote each node's resource capacity or availability. For the purpose of simplicity, we will focus on the CPU and memory requirements/capacity of all tasks and resources. That is, the storage is sufficient for the size of data input δ, and hence the requirement $\langle \delta, c, m \rangle$ is

TABLE I: Common Notations

Notation	Description
$\mathbb{EDGE}$	Federated edge deployments
T	Individual application or task
$\langle \delta, c, m \rangle$	Storage, CPU and memory resources
$\mathbb{C}$	A set of containerized applications
$\tilde{T}^{\langle c,m \rangle}$	Task resource requirements estimation
D_{edge_i}	Individual edge deployment or cluster
$D_{edge_{i^\star}}$	Closest edge deployment or cluster
I_i	Container-instance or node in a cluster
$I_i^{\langle c,m \rangle}$	Resource capacity or availability of a node
$D_{edge_i}^{\langle c,m \rangle}$	Resource capacity/availability in an edge
$D_{edge_i U}^{\langle c,m \rangle}$	Resources used for execution
$D_{edge_i ARU}^{\langle c,m \rangle}$	Actual resources usage of jobs
E_s, E_c	Application/task start, completion time
E_{ex}	Application or task execution time
$\tilde{E}_{ex}$	Application or task execution time estimation
f_i	Drones' flight time
f_i^l	Drones' flight time from its box to location
$\rho_C^{\langle c,m \rangle}$	Cluster resource utilization
$\rho_C^{\langle c \rangle}$, $\rho_C^{\langle m \rangle}$	Cluster CPU, memory resource utilization
ϵ_J	The type of job
γ_J	Dependency depth of a job
$\mathbb{F}$	Set of multi-task features
J, $\mathbb{J}$	A Job, A set of Jobs
u, $\mathbb{U}$	A User, A set of Users

simplified as $\langle c, m\rangle$. The estimated resource demands and execution time of k containerized applications to be orchestrated, $\sum_{i=1}^{k} \tilde{T}_i^{\langle c,m\rangle}$ and $\sum_{i=1}^{k} \tilde{E}_{ex_i}$, the update state of the $\mathbb{EDGE}$ clusters, i.e., the resources availability $D_{edge_i}^{\langle c,m\rangle}$, and drones' flight time f_i are important information needed in order to make informed decision on where to deploy ready applications $\mathbb{C}$ at time t. Our strategy chooses the closest edge having requisite capacity and fight time. In real scenario where multi-users $u \in \mathbb{U}$ offload multi-tasks with multi-dependency at t, these applications are deployed as a multi-Job $\mathbb{J}$, where each Job J is a collection of each user's multi-tasks, with collective resource demand estimation denoted as $\sum_{i=1}^{k} \tilde{T}_i^{\langle c,m\rangle} = \tilde{T}^{\langle c,m\rangle\prime}$, and the aggregate execution time estimation as $\sum_{i=1}^{k} \tilde{E}_{ex_i} = \tilde{E}_{ex}\prime$. We can deploy all users' Jobs with dependency on the same cluster by jointly considering $\sum_{J\in\mathbb{J}} \tilde{T}^{\langle c,m\rangle\prime}, D_{edge_i}^{\langle c,m\rangle}, \sum_{J\in\mathbb{J}} \tilde{E}_{ex}\prime$ and f_i. Therefore, the aggregate of the actual execution time of a multi-job $\mathbb{J}$ is given as:

$$\sum_{J\in\mathbb{J}} \sum_{i=1}^{k} \frac{E_{ex_i}}{k} = E_{ex}\prime, \tag{10}$$

and we can deploy a multi-job to the **closest** edge, such that:

$$\mathbb{J} \Rightarrow D_{edge_{i^\star}}. \tag{11}$$

The resource utilization of the cluster for multi-job deployment is thus

$$\rho_C^{\langle c,m\rangle} = \frac{\sum_{J\in\mathbb{J}} \tilde{T}^{\langle c,m\rangle\prime}}{D_{edge_i}^{\langle c,m\rangle}}. \tag{12}$$

B. Problem Formulation

The basic notations adopted are described in Table II. AerialEdge includes an intelligent scheduling, which packs tasks tightly on nodes to fully utilize available resources at edge clusters, while considering task dependencies.

Our objectives are to maximize the cluster resource utilization, $\rho_C^{\langle c,m\rangle}$ of (12), and to minimize the overall actual execution time of tasks, $E_{ex}\prime$ of (10), subject to certain constraints.

Constraints: First, the collective resource demand or request estimation of a multi-job $\mathbb{J}$ or multi-task at any given time t cannot exceed the collective resource capacity or available in the cluster:

$$\sum\nolimits_{J\in\mathbb{J}} \tilde{T}^{\langle c,m\rangle\prime} \leq D_{edge_i}^{\langle c,m\rangle}, \quad \forall_{\langle c,m\rangle}. \tag{13}$$

Second, the aggregate execution time estimation of a multi-job $\mathbb{J}$ or multi-task at any given time t cannot exceed the flight time availability of any selected drone:

$$\sum\nolimits_{J\in\mathbb{J}} \tilde{E}_{ex}\prime \leq f_i, \quad \forall D_{edge_i} \in \mathbb{EDGE}. \tag{14}$$

Third, unused or inactive container-instance or node $I_i \in D_{edge_i}$ in the cluster would be shut down. All the nodes are in one of the two states: *Active* and *Inactive*. An *Active* node is a node that is ready to accept jobs or has at least a job being started, executing or completing. An *Inactive* node is a node that is not ready to accept jobs and not having at least a job that is being started, executing or completing. These two states can be expressed as follows:

$$\forall_{c,m}\ \beta\left(I_i\right) = \begin{cases} 1, & \textit{Active} \text{ if } J_i \in [E_s, E_c, E_{ex}], \\ 0, & \textit{Inactive} \text{ if } J_i \notin [E_s, E_c, E_{ex}], \end{cases} \tag{15}$$

where the indicator $\beta\left(I_i\right) = 1$ indicates that the node I_i is ready to accept new jobs, and at least a job J_i is being started, executing or completing, i.e., $J_i \in [Es, E_c, E_{ex}]$, on I_i; otherwise $\beta\left(I_i\right) = 0$.

Optimization formulation: Hence, maximizing utilization of a cluster depends on application orchestration:

$$\textbf{Maximize} \quad \rho_C^{\langle c,m\rangle} = \frac{\sum_{J\in\mathbb{J}} \tilde{T}^{\langle c,m\rangle\prime}}{D_{edge_i}^{\langle c,m\rangle}}, \tag{16}$$

$$\textit{subject to} \quad \mathbb{J} \Rightarrow D_{edge_{i^\star}}, \quad \exists, \tag{17}$$

$$\sum\nolimits_{J\in\mathbb{J}} \tilde{E}_{ex}\prime \leq f_i, \quad \forall_{D_{edge_i}\in\mathbb{EDGE}}, \quad \exists, \tag{18}$$

$$\beta\left(I_i\right) \in \{0, 1\}, \quad \exists, \tag{19}$$

$$\sum\nolimits_{J\in\mathbb{J}} \tilde{T}^{\langle c,m\rangle\prime} \leq D_{edge_i}^{\langle c,m\rangle}, \quad \forall_{\langle c,m\rangle}. \tag{20}$$

On the other hand, the overall actual execution time can be minimized depending on orchestration:

$$\textbf{Minimize} \quad \sum_{J\in\mathbb{J}} \sum_{i=1}^{k} \frac{E_{ex_i}}{k} = E_{ex}\prime, \tag{21}$$

$$\textit{subject to} \quad \mathbb{J} \Rightarrow D_{edge_{i^\star}}, \quad \forall_{\langle c,m\rangle}. \tag{22}$$

C. Algorithm

Our AerialEdge solution consists of three components: the resource requirement and execution time estimations, dispatching, and packing. The values for multi-job required by the dispatcher are first estimated. Our dispatching strategy is based on the orchestration of ready tasks to the closest cluster or drone deployment with the minimum flight time f_i^l to arrive at location l, and having requisite available resources to accommodate the tasks, while our packing strategy involves packing these tasks tightly on nodes or container-instances to fully utilize the available resources. Below we detail the procedures of the execution time estimation, dispatching, and co-location or packing.

Resource requirement and execution time estimation: When the set of tasks $\mathbb{C}$ are ready to be deployed, the first step is to estimate the collective resource requirement $\tilde{T}^{\langle c,m\rangle\prime}$ and execution time $\tilde{E}_{ex}\prime$. We have trained a ML regression model with historical data for this prediction task. The input to the prediction model is the set of multi-task features $\mathbb{F}$, such as type of tasks ϵ, dependency depth γ, and data size δ, and the output is the resource requirement and execution time estimation. Algorithm 1 describes the estimations for multi-job. Once the estimation values are extracted, they are used in the dispatching stage.

Dispatching: Our policy is to dispatch a set of tasks to the **closest** edge $D_{edge_{i^\star}}$ with the congruent resource capacity or availability and flight time availability for any selected drone, i.e., $T^{\langle c,m\rangle\prime} \cong D_{edge_{i^\star}}^{\langle c,m\rangle}$ and $\tilde{E}_{ex}\prime \cong f_{i^\star}$, respectively. Our

Algorithm 1 AerialEdge: Resource and Execution Time Estimation

Input: Multi-Job $\mathbb{J}$ released at time t in location l, set of features $\mathbb{F}$

Output: Resource requirement $\tilde{T}^{\langle c,m\rangle\prime}$ and Execution time estimation $\tilde{E}_{ex}\prime$ of a multi-job

1: **for** $J_i \in \mathbb{J}$ **do**
2: Type of Job of $J_i = \epsilon_{J_i}$
3: Data size of $J_i = \delta_{J_i}$
4: Dependency depth of $J_i = \gamma_{J_i}$
5: **for** $T_i \in J_i$ **do**
6: $ML_R(\mathbb{F})_{T_i} = \tilde{T}^{\langle c,m\rangle}_{T_i}$
7: $ML_E(\mathbb{F})_{T_i} = \tilde{E}_{ex_{T_i}}$
8: **end for**
9: $\tilde{T}^{\langle c,m\rangle\prime}_{J_i} = \tilde{T}^{\langle c,m\rangle\prime}_{J_i} + \tilde{T}^{\langle c,m\rangle}_{T_i}$
10: $\tilde{E}_{ex_{J_i}}\prime = \tilde{E}_{ex_{J_i}}\prime + \tilde{E}_{ex_{T_i}}$
11: **end for**

Algorithm 2 AerialEdge: Dispatching Policy

Input: Multi-Job $\mathbb{J}$ released at time t within location l, federated edge-drone deployments $D_{edge_i} \in \mathbb{EDGE}$ in location l and available flight time f_i of drones

Output: Closest drone with congruent flight time and resource availability, such that $\mathbb{J} \Rightarrow D_{edge_{i^\star}}$

1: **for** $D_{edge_i} \in \mathbb{EDGE}$ **do**
2: **if** $\sum_{J\in\mathbb{J}} \tilde{T}^{\langle c,m\rangle\prime} \cong D^{\langle c,m\rangle}_{edge_i}$ and $\sum_{J\in\mathbb{J}} \tilde{E}_{ex}\prime \cong f_i$ **then**
3: **if** $D_{edge_{i^\star}} = \arg\min_{D_{edge_i}\in\mathbb{EDGE}} (f^l_i)$ **then**
4: $\mathbb{J} \Rightarrow D_{edge_{i^\star}}$
5: **else**
6: Dispatch $\mathbb{J}$ to next closest edge
7: **end if**
8: **end if**
9: **end for**

strategy utilizes the *closest* heuristic to minimize the overall response time. This is based on the orchestration of ready tasks to the closest deployment or cluster having requisite available resources to execute the tasks. *Closest* is a widely adopted heuristic or principle in distributed systems, since mobile devices often need to communicate only with the closest or nearest edge-clouds. Most of the works on edge-clouds, e.g., [8], [9], [16], adopt the *closest* principle as the task offloading policy.

Algorithm 2 describes the dispatching procedure in 3 steps. First, it captures the collective resource demand of ready multi-task/job and location of users, and updates the state of $\mathbb{EDGE}$ resources. Second, it selects the closest edge having congruent resources (line 3). Lastly, it dispatches the multi-task/job to the selected cluster (line 4). If the closest edge does not have the required resources, the selection procedure is repeated until the next closest edge having congruent resources is found, and the multi-task/job is dispatched to the next closest edge (line 6).

Packing: At the edge cluster, we develop a new packing algorithm which uses the cluster resource capacity or availability and multi-job estimated resource requirement information to provide better packing, such that more efficient resource utilization is achieved in the federated system. Specifically, the gang scheduling is adopted to co-schedule all the multi-jobs at a time, while the variable-sized multi-capacity bin-packing (VSMCBP) algorithm [17] places the jobs on nodes by co-locating jobs tightly on each node. As multi-jobs arrive at the cluster $D_{edge_\star}$, the VSMCB algorithm scans the list of the jobs, and maps these jobs to nodes. The key difference between the VSMCBP and other bin-packing algorithms, such as first fit bin packing (FFBP) [18], is the criteria used to select which jobs should be co-located to fully utilize any given node(s). The FFBP algorithm requires the next job to be packed on the current node, and if this cannot be done, a new node is used. The VSMCBP algorithm on the other hand scans the given list of jobs and maps jobs randomly to nodes in full utilization.

Multi-job $\mathbb{J}$ is a collection of several jobs $J \in \mathbb{J}$. These jobs are packed tightly on nodes, so that fewer nodes are used in full utilization and all the jobs are executed concurrently. Hence our packing strategy is to solve the problem:

$$\textbf{Minimize} \sum_{I_i \in D_{edge_\star}} I_i, \tag{23}$$

$$\textit{subject to } \mathbb{J} \Rightarrow D_{edge_\star}, \tag{24}$$

$$\sum_{J\in\mathbb{J}} \Gamma\left[J,\ I_i\right] \cdot \tilde{T}^{\langle c,m\rangle\prime} \leq I^{\langle c,m\rangle}_i, \quad \forall_{c,m}, \tag{25}$$

$$\Gamma\left[J,\ I_i\right] = \begin{cases} 1, & \text{if } J \Rightarrow I_i, \\ 0, & \text{otherwise, } \forall I_i \in D_{edge_\star}. \end{cases} \tag{26}$$

The constraint (25) indicates that the total estimated resource requirements of co-located jobs cannot exceed the node resource capacity or availability, while the condition (26) means that if job J is deployed on the node I_i, the indicator returns a value of 1; otherwise, 0 is returned. This is to ensure that each job is placed in exactly one node. The powerful Google OR-Tools[8], which provides an interface to several mixed-integer programming (MIP) solvers, i.e., coin-or branch and cut (CBC)[9], is employed to solve this VSMCBP problem for multi-job packing.

Algorithm 3 describes the packing strategy which packs tasks tightly on nodes, such that for any given tasks/jobs, fewer nodes are used for execution. It takes the estimated resource demand of multi-task/job and resource availability of container-instances or nodes as input, then scans through the multi-task/job to select jobs having congruent resources matching the active node in full utilization. This process is repeated until all jobs are scheduled on nodes.

IV. Performance Evaluation

We evaluate our AerialEdge on real-time Alibaba cluster data traces, and compare its performance with three existing state-of-the-arts, and the *Random* approach.

[8]https://developers.google.com/optimization
[9]https://projects.coin-or.org/Cbc

TABLE II: Federated-Edge Resource Capacities

Deployments	Attached Edge Devices	Total Weight	CPU Capacity	Mem Capacity
Drone 1	AWS Snowcone + Acer aiSage + Huawei AR502H	3.5kg	12 Cores	8 GiB
Drone 2	Lenovo ThinkSystem SE350 + Dell 3000s	5.75kg	18 Cores	258 GiB
Drone 3	HPE Edgeline EL300 + HIVECELL(x2)	7.6kg	16 Cores	24 GiB
Drone 4	INTELLIEDGE G700 + Azure Stack Edge mini + Dell 5000s	9.6kg	26 Cores	72 GiB
Drone 5	Azure Stack Edge mini(x2) + INTELLIEDGE G700	11.8kg	40 Cores	112 GiB
Drone 6	Azure Stack Edge mini(x4) + HIVECELL(x2)	15.4kg	76 Cores	208 GiB

Algorithm 3 AerialEdge: Multi-job packing

Input: Multi-Job $\mathbb{J}$ dispatched to closest edge cluster $D_{edge_\star}$, resource capacity or availability $I_i^{\langle c,m\rangle}$ of all nodes $I_i \in D_{edge_\star}$
Output: Multi-Job co-location through packing, such that fewer container-instances or nodes are used in full utilization, i.e., **Minimize** $\sum_{I_i \in D_{edge_\star}} I_i$

for $I_i \in D_{edge_\star}$ **do**
  if $\beta(I_i) = 1$ **then**
    $I_i^{\langle c,m\rangle} = \langle c, m\rangle$, i.e., resource availability
    for $J \in \mathbb{J}$ **do**
      if $\Gamma[J, I_i] = 1$ **then**
        $J \Rightarrow I_i$
        $I_i^{\langle c,m\rangle} = I_i^{\langle c,m\rangle} + \tilde{T}^{\langle c,m\rangle\prime}$
      end if
    end for
    if $I_i^{\langle c,m\rangle} \geq \langle c, m\rangle$ **then**
      $i = i + 1$
    end if
  end if
end for

A. Setup

Computing Resources: We use 6 distributed and federated aerial edge deployments (autonomous drones), as summarized in Table II. The computing resources are made up of heterogeneous container-optimized nodes (container-instances). These drones have various resource capacities (up to 76 CPU cores and 208 GiB of memory) and weight (up to 15kg). We assume the selected drone has congruent flight time availability.

Applications: To evaluate our framework, we employ use-cases of real-world CPU and memory intensive data-trace from Alibaba, which records the activities of both long running containers (for Alibaba's e-commerce business) and batch jobs across an 8-day period. The data trace contains about 14,295,731 tasks (with about 12,207,703 dependencies) and 4,201,014 jobs, among which we randomly choose 46 jobs with total of 198 tasks (including dependencies) for our experiments.

B. Heuristics and Baselines

In our experiments, we assume that all tasks are of high priority. Our strategy utilizes the *closest* heuristic to minimize the overall response time. This is based on the orchestration of ready tasks to the closest drone deployment or cluster (i.e., with the smallest flight time to the needed location) having requisite flight time and available resources to execute the tasks. *Closest* is a widely adopted heuristic or principle in distributed systems, since mobile devices often need to communicate only with the closest or nearest edge-clouds. Most of the works on edge-clouds, e.g., [8], [9], [16], adopt the *closest* principle as the task offloading policy.

In comparison of our AerialEdge with the state-of-the-art dependency-aware task orchestration and packing strategies, therefore, we fix the dispatching policy to that of AerialEdge, i.e., the *closest* heuristic. We compare our scheduling strategy with the following state-of-the-art benchmarks, and the *Random* approach.

1) **Graphene** [12] is a state-of-the-art approach for dependency-aware task orchestration problems. First, it co-schedules some tasks identified as *troublesome tasks*. Then the remaining tasks are divided into parent, child and sibling subsets, which are placed afterward to ensure compactness and to respect dependencies. It deploys about 40% of a given DAG at a time.
2) **Tetris** [10] is an existing state-of-the-art approach for task packing problems, although it does not consider the task dependencies. It deploys at least 50% of any given tasks at a time and primarily focuses on packing tasks on nodes mainly to achieve high resource utilization. For every task, it computes a packing score $\texttt{pScore}_t$, as a dot product between the task resource requirements vector and the node's resource availability vector.
3) **Spear** [11] is a dependency-aware task scheduler, which applies Monte Carlo Tree Search (MCTS) with deep reinforcement learning. It utilizes the *Critical Path* (CP) to pick tasks along the CP in the DAG. Spear deploys about 15% of the tasks at a time.
4) **Random** approach deploys a task randomly to any available node, and assumes a node can only execute a task at a time.

C. Deployment Results and Performance Comparison

Our investigation focuses on CPU and memory usage/utilization, task deployment, scheduling and execution times. The multi-job execution information across the federated aerial edge deployments are listed in Table III. The results obtained by AerialEdge, Graphene, Tetris, Spear and Random are compared.

1) Actual Resource Usage and Resource Utilization: We first introduce a performance metric by defining the actual resources usage of jobs $D_{edge_i ARU}^{\langle c,m\rangle}$ as the ratio of the resources used for execution $D_{edge_i U}^{\langle c,m\rangle}$ over the edge's resource capacity or availability $D_{edge_i}^{\langle c,m\rangle}$:

$$D_{edge_i ARU}^{\langle c,m\rangle} = \frac{D_{edge_i U}^{\langle c,m\rangle}}{D_{edge_i}^{\langle c,m\rangle}}. \tag{27}$$

TABLE III: Multi-Task Execution in Federated Aerial Edge

D_{edge_i}	$\mathbb{J}$	$\mathbb{C}$	γ_J	$\tilde{T}^{\langle c,m \rangle}$	$\tilde{E}_{ex}$
Drone 1	1	17	17	$\langle 8.5, 3.5 \rangle$	384.9
Drone 2	3	24	$(1,\ 17]$	$\langle 14.5, 5.77 \rangle$	456.23
Drone 3	5	16	$(1,\ 10]$	$\langle 13.75, 4.94 \rangle$	506.75
Drone 4	8	26	$(1,\ 10]$	$\langle 21.45, 7.74 \rangle$	609.9
Drone 5	13	38	$(1,\ 8]$	$\langle 34, 12.92 \rangle$	387.83
Drone 6	16	77	$(1.\ 16]$	$\langle 64.05, 27.57 \rangle$	1261.65

Another metric is the resources utilization $\rho_C^{\langle c,m \rangle}$ given in (12). Similarly, $\rho_C^{\langle c,m \rangle}$ includes the CPU utilization $\rho_C^{\langle c \rangle}$ and the memory utilization $\rho_C^{\langle m \rangle}$, which are defined respectively by

$$\rho_C^{\langle c \rangle} = \frac{\sum_{J \in \mathbb{J}} T^{\langle c \rangle \prime}}{D_{edge_i U}^{\langle c \rangle}}, \tag{28}$$

$$\rho_C^{\langle m \rangle} = \frac{\sum_{J \in \mathbb{J}} T^{\langle m \rangle \prime}}{D_{edge_i U}^{\langle m \rangle}}, \tag{29}$$

where $\sum_{J \in \mathbb{J}} T^{\langle c \rangle \prime}$ and $\sum_{J \in \mathbb{J}} T^{\langle m \rangle \prime}$ are the total collective CPU and memory demands, respectively.

Fig. 4 compare the resource usage of AerialEdge with the three baseline schemes, and the Random approach. It can be seen that AerialEdge and Tetris use the fewest resources in the clusters, better than Graphene, Spear and Random, while Spear and Random consume the highest resources in the clusters. The CPU and memory resource utilization comparisons are shown in Figs. 5 and 6, respectively. Again, AerialEdge and Tetris are superior than Graphene, Spear and Random, achieving the highest resource utilization, while Spear and Random achieve the lowest resource utilization. We now exam the individual clusters in detail. Fig. 7 shows the percentage of tasks deployed on each drone.

In Drone−1 edge-cluster, we deploy 1 job with a total of 17 tasks, where the job has a task dependency depth of 17. AerialEdge first optimizes the deployment by co-locating as many jobs in a node as possible, to fully utilize the available resources in the node. Utilizing the gang scheduling strategy, AerialEdge co-schedules all the 17 tasks at a time. These tasks are tightly packed on nodes using the VSMCBP algorithm, which use 75% of the resources. Using the same configuration for the baseline schemes, Graphene and Tetris also use 75% of the resources while Spear use 92%. The Random approach utilize all resources in the cluster. Importantly, AerialEdge gain faster scheduling and execution times compared to the three baseline schemes and the Random approach, mainly due to the following reasons: (i) AerialEdge utilizes the *Gang Scheduling* strategy, which co-schedules **all the tasks at a time**, and (ii) its *Packing Strategy* explores the available nodes to find the best nodes which have requisite available resources to execute all the tasks by packing them tightly on each node. Figs. 8 and 9 compare the actual scheduling and execution times respectively, of AerialEdge with the baseline schemes, and the Random approach. We observed that AerialEdge achieves faster scheduling and execution up to 42 times and 15 times respectively, compared with the state-of-the-art schemes. Compared with the Random approach, AerialEdge achieves faster scheduling and execution times of 393 times and 77 times, respectively.

Drone−2 cluster is a memory intensive cluster. Here, 3 jobs with a total of 24 tasks are deployed, where each job has a task dependency in the range of $(1,\ 17]$. We optimize the deployment to ensure that resources are fully utilized. AerialEdge consume 11% fewer resources than Graphene Spear and Random approach, and 2% fewer resources than Tetris. AerialEdge and Tetris also gain 10% higher CPU

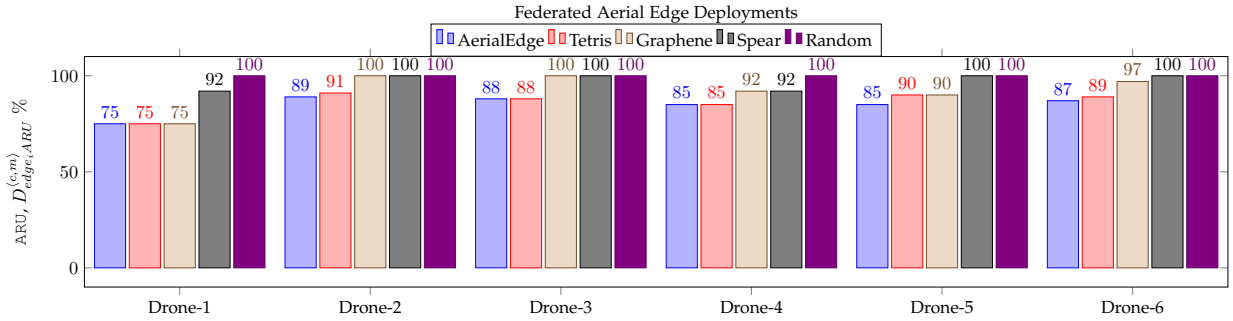


Fig. 4: Actual resource usage across the federated aerial edge clusters.

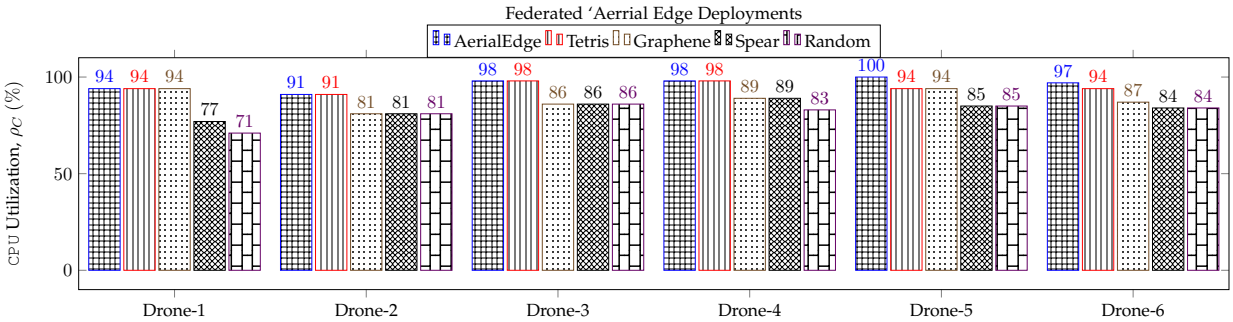


Fig. 5: CPU utilization across the federated aerial edge resources.

utilization over Graphene, Spear and Random, respectively, as well as 1% higher memory utilization than Graphene, Spear and Random, respectively. More significantly, AerialEdge is 1.1, 2, 2.7 and 31 times faster in the scheduling time than Tetris, Graphene, Spear and Random, respectively, while it is 2, 4, 5 and 24 times faster in the execution time over these state-of-the-art schemes and Random approach, respectively. It is worth recapping that in the case of Random, the results of actual resource usage, resource utilization, scheduling and execution times are for 38% of the tasks that it is able to deploy on `Drone-2`.

`Drone-3` has a total load of 7.6kg, and it is made up of 1 `HPE Edgeline EL300` and 2 `HIVECELL` edge devices, with total resource capacity of 16 Cores and 24 GiB. In this cluster, we deploy 5 jobs, with total 16 tasks, where each job has a task dependency range $(1,\ 10]$. AerialEdge and Tetris reduce resource usage by 12% compared with Graphene, Spear and Random. AerialEdge and Tetris achieve 12% and 3% higher CPU and memory utilization, respectively, compared to Graphene, Spear and Random. In terms of both scheduling and execution times, AerialEdge is about 4.5 and 2.2 times faster than Tetris. It is 6.7 times and 3.4 times faster than Graphene as well as 13 times and 5.5 times faster than Spear, in the scheduling and execution times, respectively. Not

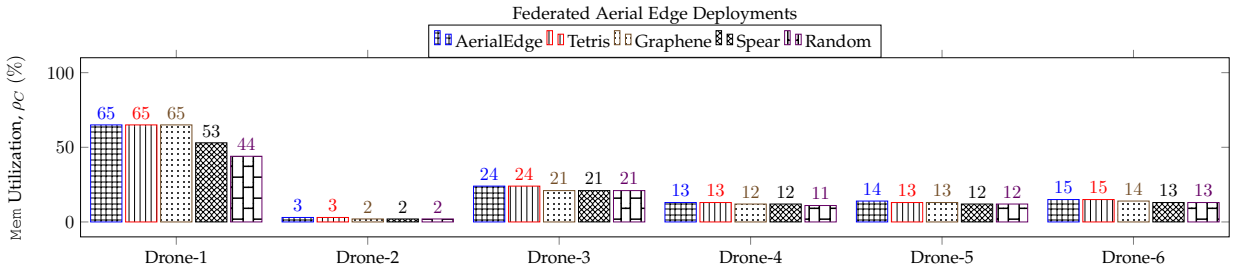


Fig. 6: Memory utilization across the federated aerial edge resources.

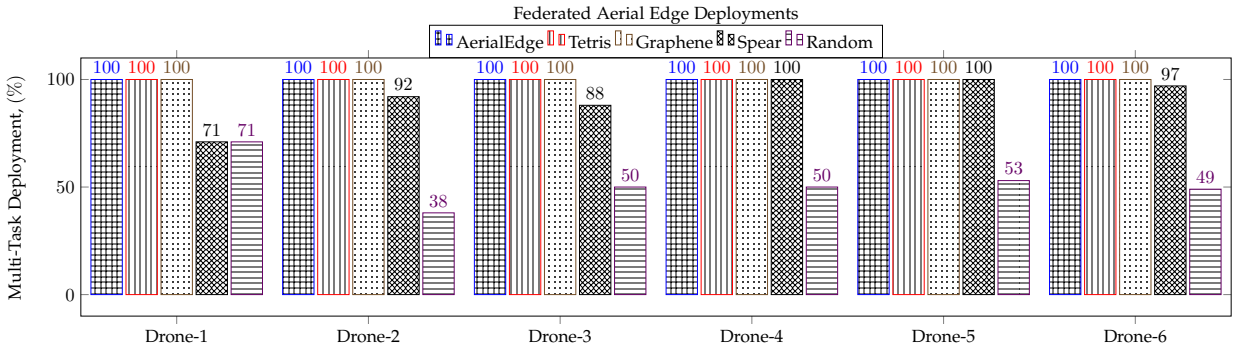


Fig. 7: Multi-task deployment across the federated aerial edge resources.

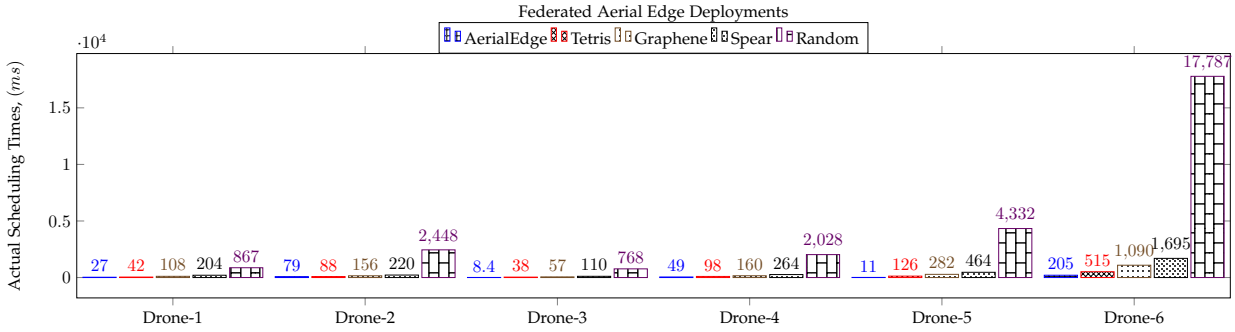


Fig. 8: Actual multi-task scheduling time across the federated aerial edge resources.

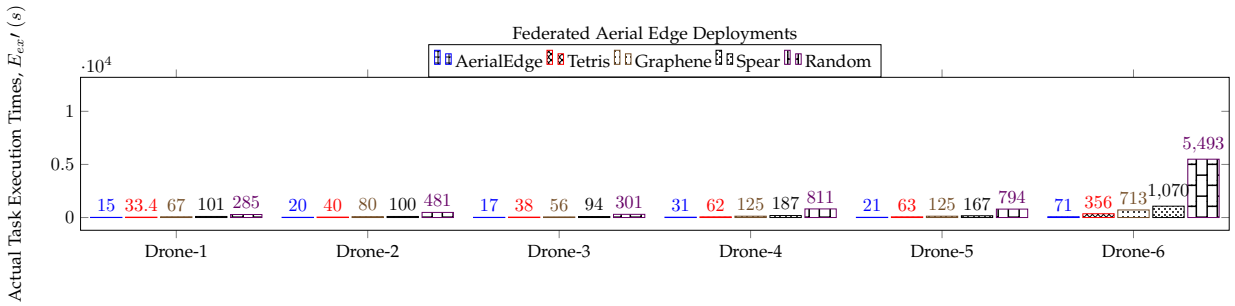


Fig. 9: Actual multi-task execution time across the federated aerial edge resources.

surprisingly, Random has the worst scheduling and execution time performance.

Drone−4 cluster is a high capacity cluster. Here, 8 jobs with total of 26 tasks are deployed, where each job has a dependency depth range $(1,\ 10]$. It can be seen that AerialEdge and Tetris consume 7% fewer resources than Graphene and Spear. AerialEdge and Tetris also achieve 9% higher CPU utilization as well as 1% higher memory utilization, over Graphene and Spear, respectively. Random can only deploy 50% of the tasks. By contrast, AerialEdge, Tetris, Graphene and Spear all deploy 100% of the tasks. In terms of scheduling time, AerialEdge is approximately 2 times, 3.2 times , 5.3 times, and 41.3 times faster than Tetris, Graphene, Spear and Random, respectively. In terms of execution time, AerialEdge is about 2 times, 4 times, 6 times and 26 times faster than Tetris, Graphene, Spear and Random, respectively. Again Random has the worst scheduling and execution time performance.

Drone−5 and Drone−6 are the largest clusters in terms of resource and load capacity. We deploy 13 and 36 jobs in these two clusters, respectively. The total number of tasks deployed in Drone−5 cluster is 38, while total of 77 tasks are deployed in Drone−6 cluster. The task dependency depth of each job is in the range of $(1,\ 16]$. In these two clusters, AerialEdge use up to 13% less resources compared with the baselines and the Random approach. Specifically, Spear and Random used up all available resources, with some nodes running under-utilization. AerialEdge and Tetris achieve the highest resource utilization and deploy 100% of the tasks in both clusters. Random approach could only deploy 53% and 49% of all tasks in Drone−5 and Drone−6, respectively. We observed that AerialEdge achieves faster scheduling and execution of up to 42 times and 15 times than the baseline schemes in Drone−5 and Drone−6, and faster scheduling and execution of up to 393 times and 77 times than the Random approach in both clusters.

2) Task Scheduling and Execution Times: The task scheduling time, which is the time it takes to place multi-jobs/tasks on the nodes in a cluster, is an important performance metric to assess the federated edge clusters. Another even more important performance metric is the aggregate job actual execution time $E_{ex}\prime$ defined in (10). Figs. 8 and 9 compare the scheduling times and execution times, respectively, attained by the four schemes. It can be seen that the scheduling times are typically very small, and the execution times by contrast are significantly larger. Across the federated edge clusters, AerialEdge consistently achieves the fastest scheduling and execution times, compared to the three benchmark strategies, and the Random approach.

The significant advantage of AerialEdge in terms of aggregate job execution time can be explained as follows. It deploys sets of multi-jobs/tasks as a unit through the gang scheduling strategy. These applications are deployed and executed concurrently. By contrast, the benchmark approaches deploy the given DAGs individually and in parts, resulting in longer time to schedule and execute the overall tasks.

V. Conclusions

This paper has presented a dependency-aware multi-task orchestration in a federated aerial edge computing-enabled learning system, called AerialEdge, to improve resource efficiency and enhance performance. We have utilized a resource-specific dispatching strategy that selects the closest drone suitable for given job(s), and a bin packing optimization strategy that co-locates tasks tightly on nodes to fully utilize available resources. Our approach involves multi-task resource requirements and execution time estimations, federated aerial edge clusters update state service, gang scheduling and co-location on container-optimized. We have compared our approach with the state-of-the-art dependency-aware task orchestration and task packing baseline strategies. AerialEdge achieves both the highest cluster resource utilization and the minimum execution time for multi-tasks/jobs compared to the baseline strategies. We observe that AerialEdge consumes up to 25% fewer resource and achieves up to 23% high cluster utilization, while leading to up to 393 times faster scheduling time and up to 77 times faster execution time.

References

[1] F. Giuseppe, G. Christian and S. Giovanni "Fog in the clouds: UAVs to provide edge computing to IoT devices," *ACM Trans. Internet Technology*, vol. 20, no. 3, pp. 26:1–26:26, 2020

[2] J. Ren, *et al.*, "Federated Learning-Based Computation Offloading Optimization in Edge Computing-Supported Internet of Things," *IEEE Access*, vol. 7, pp. 69194–69201, 2019.

[3] R. Yu and P. Li, "Toward Resource-Efficient Federated Learning in Mobile Edge Computing," *IEEE Network*, vol. 35, no. 1, pp. 148–155, January/February 2021.

[4] B. McMahan, *et al.*, "Communication-Efficient Learning of Deep Networks from Decentralized Data," *Proc. AISTATS 2017* (Fort Lauderdale, FL, USA), Apr. 20-Apr. 22, pp. 1273–1282.

[5] U. Awada and J. Zhang, "Edge Federation: A Dependency-Aware Multi-Task Dispatching and Co-location in Federated Edge Container-Instances," in *Proc. EDGE 2020* (Beijing, China), Oct. 19-Oct. 23, pp. 91–98.

[6] R. Urgaonkar, *et al.*, "Dynamic service migration and workload scheduling in edge-clouds," *Performance Evaluation*, vol. 91, pp. 205–228, 2015.

[7] L. Tong, Y. Li, and W. Gao, "A hierarchical edge cloud architecture for mobile computing," in *Proc. INFOCOM 2016* (San Francisco, CA, USA), Apr. 10-14, 2016, pp. 1–9.

[8] U. Awada and A. Barker, "Resource efficiency in container-instance clusters," in *Proc. 2nd Int. Conf. Internet of Things, Data and Cloud Computing* (Cambridge, UK), Mar. 22-23, 2017, pp. 1–5.

[9] U. Awada and A. Barker, "Improving resource efficiency of container-instance clusters on clouds," in *Proc. CCGRID 2017* (Madrid, Spain), May 14-17, 2017, pp. 929–934.

[10] R. Grandl, *et al.*, "Multi-resource packing for cluster schedulers," in *Proc. SIGCOMM 2014* (Chicago, IL, USA), Aug. 17-22, 2014, pp. 455–466.

[11] Z. Hu, J. Tu, and B. Li, "Spear: Optimized dependency-aware task scheduling with deep reinforcement learning," in *Proc. ICDCS 2019* (Dallas, TX, USA), Jul. 7-10, 2019, pp. 2037–2046.

[12] R. Grandl, *et al.*, "GRAPHENE: Packing and dependency-aware scheduling for data-parallel clusters," in *Proc. OSDI 2016* (Savannah, GA, USA), Nov. 2-4, 2016, pp. 81–97.

[13] Z. Han, *et al.*, "Scheduling placement-sensitive BSP jobs with inaccurate execution time estimation," in *Proc. INFOCOM 2020* (Toronto, ON, Canada), Jul. 6-9, 2020, pp. 1053–1062.

[14] M. Mukherjee, *et al.*, "Distributed deep learning-based task offloading for UAV-enabled mobile edge computing," in *Proc. INFOCOM WKSHPS 2020* (Toronto, ON, Canada), Jul. 6-9, 2020, pp. 1208–1212.

[15] Y. Liu, S. Xie and Y. Zhang, "Cooperative offloading and resource management for UAV-enabled mobile edge computing in power IoT system," *IEEE Trans. Vehicular Technology*, vol. 69, no. 10, pp. 12229–12239, 2020.

[16] Z. Han, *et al.*, "OnDisc: Online latency-sensitive job dispatching and scheduling in heterogeneous edge-clouds," *IEEE/ACM Trans. Networking*, vol. 27, no. 6, pp. 2472–2485, 2019.

[17] P. Lai, *et al.*, "Optimal edge user allocation in edge computing with variable sized vector bin packing," in *Proc. ICSOC 2018* (Hangzhou, China), Nov. 12-15, 2018, pp. 230–245.

[18] S. Rampersaud and D. Grosu, "Sharing-Aware Online Virtual Machine Packing in Heterogeneous Resource Clouds," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 7, pp. 2046–2059, 2017.