AirEdge: A Dependency-Aware Multi-Task Orchestration in Federated Aerial Computing

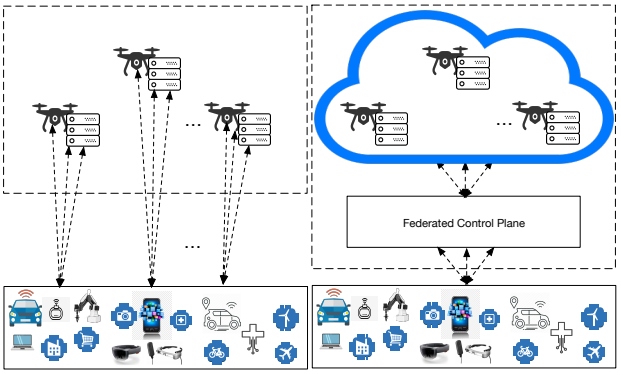
Uchechukwu Awada , *Student Member, IEEE,* Jiankang Zhang , *Senior Member, IEEE,* Sheng Chen , *Fellow, IEEE,* and Shuangzhi Li

***Abstract*—Emerging edge computing (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. To this end, existing EC research has proposed several multiple drone-based edge deployments for various purposes, such as data caching, task offloading, real-time video analytics, and computer vision. However, none of them consider the ability of seamlessly integrating edge resources running across multiple drones in a single pool, to holistically manage and control these resources as well as to eliminate vendor lock-in situations. This paper presents an intelligent resource scheduling solution for a federated aerial EC system, called AirEdge, which jointly considers task dependencies, heterogeneous resource demand and drones' flight time. We propose a multi-task execution time estimation and a dispatching policy, to select the closest drone deployment having congruent flight time and resource availability to execute ready tasks at any given time. For the utilization of the drones' attached edge resources, we propose a variant bin-packing optimization approach through gang-scheduling of multi-dependent tasks that co-locates tasks tightly on nodes to fully utilize available resources. Experiments on real-world data-trace from Alibaba cluster trace with information on task dependencies (about 12,207,703 dependencies) and resource demands show the effectiveness, fast executions, and resource efficiency of our approach.**

***Index Terms*—Edge computing, Aerial computing, Dependency-aware, Application container, Execution time, Resource efficiency**

I. Introduction

EDGE computing (EC) is an innovative distributed computing paradigm that brings computation and data storage closer to the location where they are needed, to improve response times and save bandwidth. Instead of housing these critical resources in a big data-center that could be hundreds or thousands of miles away from the data source, this enabling architecture deploys them at the edge of the network, and even beyond the edge of the network [1]. Emerging latency-sensitive technologies, such as connected and autonomous vehicles (CAVs) [2], healthcare IoT systems [3], real-time augmented reality [4], smart cities [5], Industry 4.0 [6], etc, rely on heterogeneous edge resources in close proximity, to offload their computational intensive tasks, improve response times and save bandwidth. To further improve latency, the low altitude platform (LAP) unmanned aerial vehicles (UAVs) or drones are currently being exploited by EC systems to execute complex resource-hungry use cases [7]–[10]. Drones can fly to the target locations with nearly no constraint due to their mobility, flexibility and adaptive altitude, to deliver faster execution closer to data source. 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 [11]. 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.



(a) Multiple Aerial Computing (b) Federated Aerial Computing

Fig. 1. (a) The architecture of multiple aerial edge computing system, and (b) the architecture of a federated aerial edge computing system. The latter requires a holistic approach to multiple aerial edge resource management.

Consequently, existing researches on EC has proposed several multiple drone-based edge deployments to cater for wide range of end devices [12]–[17]. However, none of them considers the ability of seamlessly integrating edge resources and service entities running across multi-drone deployments in a single pool, such that these resources can be holistically managed and controlled from a single federated plane, applications can be deployed dynamically across the resources, and vendor lock-in situations can be eliminated. Moreover, efficient orchestration of complex dependencies among tasks in such independent aerial deployments is challenging due to constrained resource capabilities, mobility and availability factors, etc. For example, Fig. 1 (a) shows a multiple aerial

Manuscript received February 7, 2021; revised September 29, 2021; accepted November 8, 2021. This work was supported in part by joint Funds of National Natural Science Foundation of China under grant 61571401 and 61901416 (part of the China Postdoctoral Science Foundation under grant 2021TQ0304), and the Innovative Talent of Colleges and the University of Henan Province under grant 18HASTIT021. The review of this paper was coordinated by Associate Editors of IEEE TVT. *(Corresponding authors; Jiankang Zhang and Shuangzhi Li.)*

Uchechukwu Awada and Shuangzhi Li are with the School of Information Engineering, Zhengzhou University, Zhengzhou, 450001, China (E-mails: awada@gs.zzu.edu.cn, ielsz@zzu.edu.cn).

Jiankang Zhang is with the Department of Computing and Informatics, Bournemouth University, Poole, BH12 5BB, UK (E-mail: jzhang3@bournemouth.ac.uk).

Sheng Chen is with the School of Electronics and Computer Science, University of Southampton, Southampton SO17 1BJ, UK (E-mail: sqc@ecs.soton.ac.uk).

0018-9545 © 2021 IEEE. Personal use of this material is permitted. However, permission to use this material for any other purposes must be obtained from the IEEE by sending a request to pubs-permissions@ieee.org.

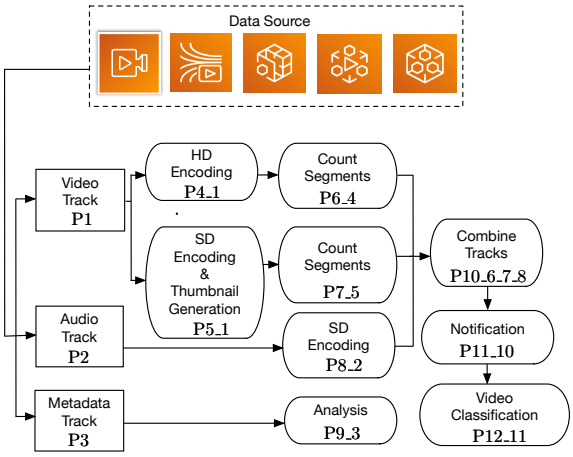


Fig. 2. The directed acyclic graph (DAG) of a video processing (VP) job.

computing system without any cooperation. Such an approach can result in resource overloaded and longer queuing delay for tasks due to insufficient resource availability, and hence it is not suitable for latency-sensitive edge workloads.

Aerial edge federation makes it easier to manage multiple drone deployments by synchronizing resources across multiple drones, enabling flexible tasks execution and preventing lock-in situations. Having a federated edge minimizes latency by serving users from the edge that is the closest to them [18], [19]. A recent lightweight Kubernetes-based edge tool, called *Kubermatic*[1], can deploy and manage multiple edge resources running across multi-drones with a single management interface, as shown in Fig. 1 (b). In a federated edge clusters setup, the federation control plane (FCP) is deployed in one of the clusters which serves as the `host cluster`. Participating edge deployments can be added or removed from the FCP. Nevertheless, to offload complex applications, e.g., a video processing application shown in Fig. 2, which consists of a large number of inter-dependent applications and requires substantial resources for execution, present several challenges. First, given a federated edge resources running across multi-drone, where each drone is attached with one or more edge devices, how to automatically decide where a job or multi-task should be executed is a tricky task. Previous works [20]–[22] assume that each edge deployment can only execute one task or job at any time and schedule each task individually, which results in high communication overhead. Second, the existing container schedulers deploy tasks randomly on nodes with sufficient resource availability without considering inter-task dependencies, 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. These schedulers do not pack tasks tightly on nodes to achieve high resource utilization.

In this paper, we show that machine learning (ML) techniques [23], [24] can help federated aerial edge systems to achieve effective dispatching strategy and to cope with stochastic service request arrivals. We propose AirEdge, which extends the state-of-the-arts by providing an intelligent dependency-aware multi-task dispatching and co-location scheme to achieve high resource utilization and fast execution of tasks in a federated autonomous aerial (drone-based) EC system. For a multi-task dispatch, a major issue is the complexity of federated aerial edge network, which consists several drones attached with edge devices and heterogeneous resource request from end users. A decentralized approach [25], [26], which interacts with individual member cluster, would exhibit high computation complexity and is far from trivial to realize. Therefore, we adopt an FCP to holistically obtain an update state from all participating drones, in terms of location, flight time availability and resource availability, through a single application programming interface (API), such that optimal multi-task dispatching is achieved. By contrast, if multi-tasks are scheduled naively, e.g., in an edge deployment which can only execute one task or job at any time, each task is scheduled individually [20]–[22], federated aerial edge can become unproductive. Hence, an efficient multi-task orchestration is needed to achieve optimal performance in federated aerial computing. With limited edge resources and drones' flight time, it is necessary to consider task dependencies in drone-based EC task offloading, by jointly optimizing the drones' flight time and resource availability such that all the tasks can be fast executed with minimum resources before the drone has to return for recharging. Hence, our aim is to schedule and execute all the tasks by considering dependencies and resource demands, such that the **actual** scheduling and execution time is minimized, and is much less than the drones' flight time. In summary, to achieve our AirEdge implementation, we address the following critical areas:

- We propose an intelligent scheduling through the joint optimization of the set of tasks packaged in lightweight containers, the drones' flight time and the cluster resources, that packs or co-locates the tasks tightly on nodes to fully utilize available resources.
- Specifically, we derive a multi-task ML based execution time estimation and a dispatching policy, called *closest*, to select the closest drone deployment having congruent flight time and resource availability to execute ready tasks at any given time and to autonomously deploy the selected drone to the needed location.
- To fully utilize the available drones' attached edge resources, we further propose a variant bin-packing optimization approach through gang-scheduling of multi-dependent tasks, to co-locate tasks tightly on nodes. The drone returns to its box after completing its mission.
- We show that the proposed AirEdge 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.
- We conduct extensive experiments and comparisons with real-world data-trace from Alibaba cluster trace[2], which provides information on task dependencies and resource demands, on federated aerial edge deployments.

The rest of the paper is organized as follows. In Section II, we discuss the related work. In Section III, we present some

[1]https://www.kubermatic.com/products/kubermatic/

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

TABLE I
RECENTLY INTRODUCED DRONE-BASED EDGE DEVICES

Edge Devices	Weight	Storage Capacity	GPU/vCPU Capacity	Memory Capacity	Onboard Computing
AWS Snowcone	2.1Kkg	8 TB	2 vCPUs	4 GiB	AWS IoT Greengrass
Acer aiSage	357g	32 GB eMMC	6 vCPUs + GPU	2 GiB	AI Inferencing
Huawei AR502H Series	1.1kg	Up to 250 GB	4 vCPUs	2 GiB DDR4	Docker
Lenovo ThinkSystem SE350	3.75kg	Up to 500 GB	Up to 16 vCPU	Up to 256 GiB	Lenovo XClarity
Dell Edge Gateway 3000*s*	1kg	Up to 64 GB	2 vCPUs	Up to 2 GiB	EDM
Dell Edge Gateway 5000*s*	1kg	Up to 512 GB	2 vCPUs	Up to 8 GiB	EDM
HPE Edgeline EL300	4.91kg	Up to 256 GB	Up to 4 vCPUs	Up to 8 GiB	HPE Edgeline IoT
INTELLIEDGE G700	5.45kg	Up to 512 GB	Up to 8 vCPUs	Up to 16 GiB	FUJITSU IoT Solution
Google Edge TPU	39g	8 GB eMMC	4 vCPUs	2 GiB	AI/ML Inferencing
Azure Stack Edge mini	3.17kg	1 TB	16 vCPUs	48 GiB	AI/ML Inferencing
HIVECELL	1.36kg	500 GB	6 vCPUs + GPU	8 GiB	AI/ML Inferencing
dynaEdge DE-100	309g	Up to 512 GB	2 vCPUs	Up to 16 GiB	DynaBook Vision
NVIDIA Jetson Xavier NX	47g	Up to 512 GB	6 vCPUs + GPU	8 GiB	AI/ML Inferencing

preliminaries on task dependency-awareness and discuss our motivation. In Section IV, we detail our proposed AirEdge for achieving high resource utilization and minimizing the execution times of applications deployed on federated aerial edge resources. In Section V, we compare the performance of our proposed AirEdge against those of several state-of-the-art approaches through extensive experiments. Finally, we conclude the paper in Section VI.

II. RELATED WORK

To support a wider implementation of drone-based edge deployment, cloud computing providers, i.e., Amazon Web Services (AWS), Microsoft Azure, etc., have recently introduced various drone-based edge computing devices and begun offering edge/cloud computing services directly on these devices. Table I lists some of the recently introduced drone-based edge devices. A typical drone-based edge deployment can attach one or more or different combination of these devices, depending on the drones' load capacity.

Multiple UAVs/drones can be deployed to provide EC services for IoT and other end devices. The authors of [12] proposed a multi-UAV-aided mobile-edge computing (MEC) system, where multiple UAVs act as MEC nodes to provide computing offloading services for ground IoT nodes of limited local computing capabilities. The work [13] proposed a multi-UAV-enabled MEC system, where edge servers are equipped on multiple UAVs to provide flexible computation assistance to IoT devices with hard deadlines. In [14], a two-layer optimization method was presented for jointly optimizing the deployment of UAVs and task scheduling to minimize system energy consumption. The work [15] formulated a computation efficiency maximization problem in a multi-UAV assisted MEC system, where both computation bits and energy consumption were considered. In [16] a multi-UAV enabled MEC system was introduced, where the energy consumption for ground users is minimized by jointly optimizing the UAV task scheduling, bit allocation, and UAV trajectory. The authors of [17] proposed a cluster of multi-UAVs to provide computing task offloading and resource allocation services to IoT devices. They further proposed a multi-agent deep reinforcement learning (MADRL)-based approach to minimize the overall network computation cost while ensuring the quality of service (QoS) requirements of IoT devices.

Contrary to on-premise edge deployments and cloud computing, drone-based edge resources are limited, and therefore managing resources is one of the key challenges in aerial edge deployment [10]. Task co-location of different workloads on the same computing cluster has gained popularity as a heuristic solution for improving resource utilization and system throughput in both cloud and edge computing. A workload co-location mechanism was proposed in [27]–[30] to maximize the resource utilization. Our work in this paper differs substantially from the previous works [29], [30], which focused on workload co-location in cloud environment. To further improve edge resources, a resource management scheme which seamlessly integrates or federates resources across multiple edge, such that the resources are holistically managed has been proposed in [18], [19], [25], [26]. Our recent work [19] considered a dependency-aware task dispatching and co-location in a federated edge system. Dependency usually exists among the tasks of a job. A task cannot start running until its dependent tasks have been completed. Modern applications are complex and consist of a large number of inter-dependent applications. The problem of task scheduling based on task dependency was investigated in [31]–[34]. The goal of these approaches is to identify task scheduling decision that minimizes the average completion time of multiple applications.

The aforementioned schemes on multiple drone-based task offloading and execution however do not consider the ability of seamlessly integrating edge resources and service entities running across multi-drone deployments in a single pool, such that these resources can be holistically managed and controlled from a single federated plane, applications can be deployed dynamically across the resources, and vendor lock-in situations can be eliminated. They do not consider tasks dependencies and assume that each UAV can only execute a task. In addition, they do not consider drones' flight time, and assume that a drone can fly for unlimited amount of time. It is important to note that such an approach is impractical, except for tethered drone systems which have limited navigation. Therefore, an effective completion time estimation of ready applications is needed to produce a dispatching plan in a federated aerial EC system, such that these applications can be offloaded to a drone having sufficient flight time and resources to execute the applications. To this end, research on other types of EC systems has proposed several methods of predicting or estimating tasks execution time, based on ML [23], [24] and

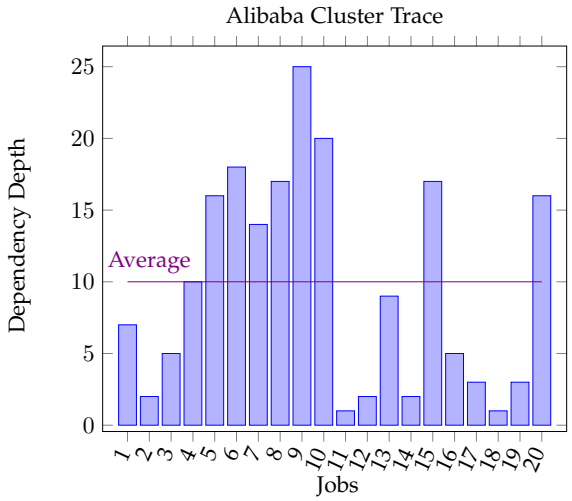


Fig. 3. Dependency depth of randomly selected Alibaba cluster jobs.

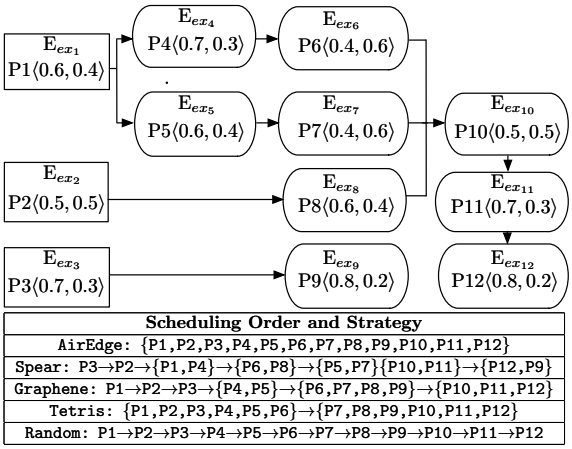


Fig. 4. The example of scheduling strategies.

incremental learning [35], as well as of scheduling [36]–[39].

III. Preliminaries and Our Motivation

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

A. Preliminaries

Dependency-awareness is critical for achieving efficient multi-task dispatching and co-location. Most of the batch workloads of Alibaba cluster trace for example are directed acyclic graphs (DAGs), and only some of them are independent. Fig. 3 plots the dependency depth of 20 randomly selected jobs from Alibaba cluster trace. It can be seen that the average job has a 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. The DAG of the tasks in a job can be deduced from the `Task_Name` field of all the tasks of this job. For example, the DAG of a video processing (VP) job is shown in Fig. 2, where multi-dependent tasks together complete the video classification computation. The job consists of inter-task dependency depth γ of 12, i.e., (`P1`, `P2`,...,`P12`). The DAGs of the 12 tasks are expressed with their `Task_Names`. Task '`P1`' means that `P1` is an independent task and can be started without waiting for any other task. Task '`P4_1`' indicates that task `P4` depends on the completion of task `P1`. Similarly, '`P10_6_7_8`' means that task `P10` depends on the finishing of tasks `P6`, `P7` and `P8`. A task is characterized by the type ϵ, data size δ, resource requirements in terms of CPU $\langle c \rangle$ and memory $\langle m \rangle$. The complex inter-task dependency with multidimensional resource demands, i.e., various amounts of CPU and memory resources, and communication requirements, make resource management in such an EC system very challenging. 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 [29], [30], [40]. 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.

B. Our Motivation

To illustrate the advantage of AirEdge, we show a motivating example in Fig. 4. The upper part 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 in a drone edge with requisite available resources, such that dependent tasks can communicate faster to make applications more interactive, compared to other deployments across different drones [10]. Here, we assume that the selected drone has the requisite fight time and available resources to accommodate all the tasks, i.e., $\langle 8, 5 \rangle$. The lower part show the scheduling of AirEdge together with three other state-of-the-art approaches, namely, Spear [40], Graphene [41] and Tetris [42], as well as the random approach. Our AirEdge 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 [36], 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 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 afterward, 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

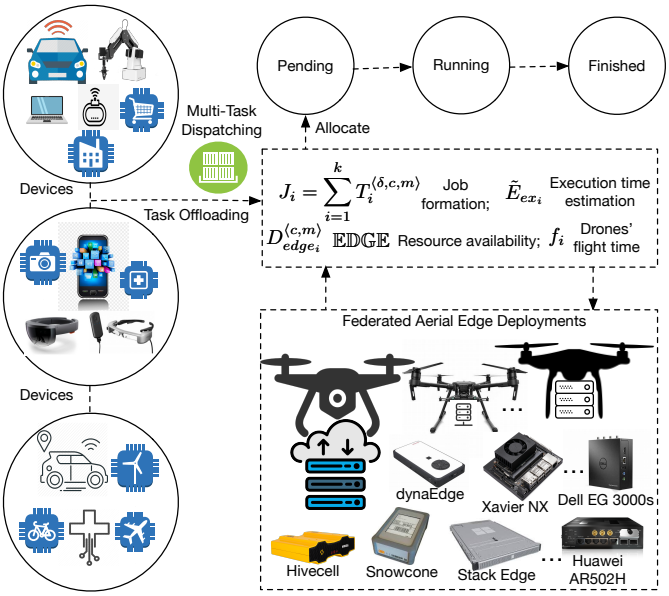


Fig. 5. Orchestration overview of AirEdge.

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.

IV. Proposed AirEdge

In this section, we detail our proposed AirEdge for achieving high resource utilization and minimizing the execution times of applications deployed on federated aerial edge resources. Our system model is depicted in Fig. 5.

A. System Model

The most important feature of EC is the ability to provide storage and computing resources close to where it is needed, so that applications can process data and return results with a minimum time. One of the advantages of using federated edge system is the ability to synchronize core data across all edge deployments, such that same high efficiency can be achieved wherever it is needed. For example, a CAV moving within a road segment at a constant speed v should be able to access the **closest** edge deployment and react immediately to changing road conditions, without first offloading its core data which could lead to an increased latency. Therefore, data synchronization is a best-fit solution for CAVs to fully exploit EC. Another example is IQ Smart City solution[3], which implements an artificial intelligence (AI) based multi-sensory analytic system for video (face recognition, license plate recognition, behavior analysis, etc.), sound and smell analytics in 60 countries. Assuming that all the sensor devices and their locations D_i^l are federated with the edge, such that their data can be synchronized among the participating edge deployments, then any closest available drone deployment can fly autonomously to the needed location D_i^l to execute tasks without the need of prior data offloading. Autonomous drones have tools onboard to help them move around and plan paths, as well as to estimate their fight time f_i^l from their locations l_i to any needed location D_i^l. For applications with small data sizes, it is possible to package the applications and database in containers, and then to deploy it to the **closest** edge whenever it is needed. For such applications, let $\langle \delta, c, m \rangle$ represent the size of data input, CPU and memory requirements, respectively. The advantage of using containers to host applications at the edge is that these applications can be executed in any edge deployment regardless of the resource type, configuration or vendor/provider. All the recently introduced edge devices in Table I are made of container-instances (container optimized nodes), which provide an efficient route to application execution within a lightweight, isolated and well-defined execution environment, where each container can run within its specified resource demand $\langle \delta, c, m \rangle$. For instance, a class of AWS EC2 instances under the family name *snc1*, i.e., *snc1.micro*, *snc1.small* and *snc1.medium* can be provisioned on AWS Snowcone edge device.

Given a federated aerial EC deployment $\mathbb{EDGE}$, where each participating edge deployment D_{edge_i} is a drone or cluster of container-instances i.e., edge device(s) with virtualized container-optimized nodes, an update state from the FCP which include each drones' fight time availability f_i, location l_i and total resource capacity or availability $D_{edge_i}^{\langle \delta,c,m \rangle}$ is needed to dispatch ready applications, $\mathbb{C}$, to the **closest** drone deployment $D_{edge_{i^\star}}$ with minimum flight time f_i^l from its location l_i to the end device(s) location D_i^l, having sufficient flight time f_i and resource capacity or availability $D_{edge_i}^{\langle \delta,c,m \rangle}$ to execute the tasks, such that the tasks are dispatched concurrently, namely,

$$\mathbb{C} \Rightarrow D_{edge_{i^\star}}, \tag{1}$$

where

$$D_{edge_{i^\star}} = \arg \min_{D_{edge_i} \in \mathbb{EDGE}} \left\{ f_i^l : f_i^l < f_i, D_{edge_i}^{\langle \delta,c,m \rangle} \text{ sufficient} \right\}. \tag{2}$$

Existing works on multi-UAV or drone-based task offloading and execution in EC systems [10], [12]–[17] 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 [11]. Some researches [43], [44] have proposed a greedy approach to deploy a job to an edge which brings the least increase to the response time. But this approach can lead to job waiting at the server due to insufficient resource availability, which is not suitable for latency-sensitive jobs. Hence an execution time estimation, tasks resource demands, edge resource availability and drone's flight time should be jointly considered in a drone-enabled EC system. Therefore, given a federated drone-based edge deployment $\mathbb{EDGE}$ and a set $\mathbb{C}$ of inter-dependent containerized applications, where each application $T_i \in \mathbb{C}$ serves as a task with its resource demand denoted as $T_i^{\langle c,m \rangle}$, our goal is to use the predicted or estimated execution time of all the tasks to select a drone having congruent flight time and resource availability, such that we can intelligently schedule the tasks to minimize the actual execution time as well as to achieve high resource utilization efficiency.

As multi-dependent containerized applications are admitted into the system, their execution times are estimated using

[3]https://iomni.ai/

linear regression model. The multi-task features $\boldsymbol{f}_{\mathrm{mt}}$, including type (number) of tasks ϵ, dependency depth γ, resource demand $\langle c, m\rangle$ and data size δ, are feed into the ML model Θ to estimate the values of their execution times, i.e.,

$$\boldsymbol{f}_{\mathrm{mt}} \cdot \Theta = \left[\tilde{E}_{ex_1}\ \tilde{E}_{ex_2} \cdots \tilde{E}_{ex_\epsilon}\right]. \tag{3}$$

Assuming that $\boldsymbol{f}_{\mathrm{mt}} \in \mathbb{R}^{1\times d}$ is a d-dimensional vector (tensor), then Θ is a $(d\times\epsilon)$-dimensional parameter matrix. To build this predictor Θ, we train it using historical data from previously executed tasks/jobs based on Keras[4]. Keras is a library which wraps TensorFlow[5] complexity into simple and user-friendly API. The dataset $\mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^{n}$ contain d-dimensional tensors of data features $\boldsymbol{x}_i \in \mathbb{R}^{1\times d}$ and ϵ-dimensional tensors of labels (actual execution times) $\boldsymbol{y}_i \in \mathbb{R}^{1\times\epsilon}$. The learning problem is to solve the following optimization:

$$\Theta^{\star} = \arg\min_{\Theta\in\mathbb{R}^{d\times\epsilon}} \frac{1}{2n}\sum_{i=1}^{n} \|\boldsymbol{x}_i\Theta - \boldsymbol{y}_i\|_2^2 + \frac{\lambda}{2}\|\Theta\|_F^2, \tag{4}$$

where λ is the regularization parameter and $\|\cdot\|_F$ denotes the Frobenius norm. The optimization (4) is solved using gradient-descent, where the model is updated iteratively until convergence, i.e., $\Theta^{t+1} = \Theta^{t} - \eta\left(\frac{1}{n}\boldsymbol{g}(\Theta^{t}) + \lambda\Theta^{l}\right)$, in which η is the learning rate, $\boldsymbol{g}(\Theta) = \frac{1}{n}\boldsymbol{X}^{\mathrm{T}}\left(\boldsymbol{X}\Theta - \boldsymbol{Y}\right)$ denotes the gradient of the loss function, $\boldsymbol{X} = \left[\boldsymbol{x}_1^{\mathrm{T}} \cdots \boldsymbol{x}_n^{\mathrm{T}}\right]^{\mathrm{T}}$ and $\boldsymbol{Y} = \left[\boldsymbol{y}_1^{\mathrm{T}} \cdots \boldsymbol{y}_n^{\mathrm{T}}\right]^{\mathrm{T}}$ are the feature set and label set, respectively.

Note that the dispatcher only has the value of $\tilde{E}_{ex_i}$, instead of the actual execution time, when making a decision to select a drone. Also it is important to note that existing researches [23], [24], [35], [37]–[39] do not consider the scheduling strategy when estimating execution time of tasks. However, since the scheduling actually influences the job execution time, it should be taken into consideration when estimating the execution time. We show that with this estimation of job execution time, AirEdge can minimize the actual execution time of multi-dependent tasks and achieve high resource utilization in a federated aerial edge system.

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

$$E_{ex} = E_c - E_s. \tag{5}$$

AirEdge utilizes the gang scheduling [36] 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}, $1 \le i \le N$, i.e.,

$$\mathbb{EDGE} = \sum\nolimits_{i=1}^{N} D_{edge_i}. \tag{6}$$

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 simplified as $\langle c, m\rangle$. The resource demands of k containerized applications to be orchestrated, $\sum_{i=1}^{k} T_i^{\langle c,m\rangle}$, the update state of the $\mathbb{EDGE}$ clusters, i.e., the resources availability $D_{edge_i}^{\langle c,m\rangle}$, drones' flight time f_i and location l_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 drone having requisite capacity $D_{edge_i}^{\langle c,m\rangle}$ and fight time f_i. 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 denoted as $\sum_{i=1}^{k} T_i^{\langle c,m\rangle} = T^{\langle c,m\rangle\prime}$, and the aggregate execution time estimation as $\sum_{i=1}^{k}\tilde{E}_{ex_i} = \tilde{E}_{ex}\prime$. We can dispatch all users' Jobs with dependency on the same cluster by jointly considering $\sum_{J\in\mathbb{J}} T^{\langle c,m\rangle\prime}$, $D_{edge_i}^{\langle c,m\rangle}$, $\sum_{J\in\mathbb{J}}\tilde{E}_{ex}\prime$ and f_i. Hence the aggregate of the actual execution time of a multi-job $\mathbb{J}$ is given as:

$$\sum_{J\in\mathbb{J}}\sum_{i=1}^{k}\frac{E_{ex_i}}{k} = E_{ex}\prime, \tag{7}$$

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

$$\mathbb{J} \Rightarrow D_{edge_{i^\star}}. \tag{8}$$

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

$$\rho_C^{\langle c,m\rangle} = \frac{\sum_{J\in\mathbb{J}} T^{\langle c,m\rangle\prime}}{D_{edge_i}^{\langle c,m\rangle}}. \tag{9}$$

B. Problem Formulation

The basic notations adopted are described in Table II. AirEdge includes an intelligent scheduling, which packs tasks

TABLE II
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
$T^{\langle c,m\rangle}$	Application or task resource requirements
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
l_i	Drones' location
D_i^l	End device location
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
γ_J	Dependency depth of a job
$\boldsymbol{f}_{\mathrm{mt}}$	Set of multi-task runtime parameters
J, $\mathbb{J}$	A Job, A set of Jobs
u, $\mathbb{U}$	A User, A set of Users

[4]https://keras.io/

[5]https://www.tensorflow.org/

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 (9), and to minimize the overall actual execution time of tasks, $E_{ex}\prime$ of (7), subject to certain constraints.

Constraints: First, the collective resource demand or request 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:

$$\sum\nolimits_{J \in \mathbb{J}} T^{\langle c,m \rangle \prime} \leq D_{edge_i}^{\langle c,m \rangle}, \quad \forall_{c,m}. \tag{10}$$

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:

$$\sum\nolimits_{J \in \mathbb{J}} \tilde{E}_{ex}\prime \leq f_i, \quad \forall_{D_{edge_i} \in \mathbb{EDGE}}. \tag{11}$$

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:

$$\forall_{c,m}\ \beta\left(I_i\right) = \begin{cases} 1, & \textit{Active} \text{ if } J_i \in [E_s, E_c, E_{ex}], \\ 0, & \textit{Inactive} \text{ if } J_i \notin [E_s, E_c, E_{ex}], \end{cases} \tag{12}$$

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:

$$\textbf{Maximize} \quad \rho_C^{\langle c,m \rangle} = \frac{\sum_{J \in \mathbb{J}} T^{\langle c,m \rangle \prime}}{D_{edge_i}^{\langle c,m \rangle}}, \tag{13}$$

$$\textit{subject to} \quad \mathbb{J} \Rightarrow D_{edge_{i^\star}}, \quad \exists, \tag{14}$$

$$\sum\nolimits_{J \in \mathbb{J}} \tilde{E}_{ex}\prime \leq f_i, \quad \forall_{D_{edge_i} \in \mathbb{EDGE}}, \quad \exists, \tag{15}$$

$$\sum\nolimits_{J \in \mathbb{J}} T^{\langle c,m \rangle \prime} \leq D_{edge_i}^{\langle c,m \rangle}, \quad \forall_{c,m}, \tag{16}$$

$$\beta\left(I_i\right) \in \{0, 1\}, \quad \exists. \tag{17}$$

The constraint (14) indicates the dispatching of multi-job to the closest edge. We shall discuss the details of our dispatching policy in Subsection IV-C and in Algorithm 2. Constraint (15) indicates that the overall execution time estimation of the multi-job should be congruent to the flight time availability of the selected drone. The constraint (16) indicates that the collective resource demand of a multi-job cannot exceed the resource availability of the selected drone, while the condition (17) indicates that active nodes ($\beta\left(I_i\right) = 1$) should be used for execution, and inactive nodes ($\beta\left(I_i\right) = 0$) should be shut down. Hence, our aim is to minimize the number of active nodes used for execution by co-locating or packing tasks tightly on each node in order to maximize resource utilization. We shall discuss the details of our packing strategy in Subsection IV-C and in Algorithm 3.

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

$$\textbf{Minimize} \quad \sum_{J \in \mathbb{J}} \sum_{i=1}^{k} \frac{E_{ex_i}}{k} = E_{ex}\prime, \tag{18}$$

$$\textit{subject to} \quad \mathbb{J} \Rightarrow D_{edge_{i^\star}}, \quad \forall_{c,m}. \tag{19}$$

The constraint (19) indicates the dispatching of multi-job to the closest edge. The execution time of multi-dependent tasks can be minimized by executing them on the same cluster, i.e., by enabling the inter-dependent tasks to communicate faster. The details of our dispatching policy are given in Subsection IV-C and in Algorithm 2.

C. Algorithm

Our AirEdge solution consists of three components: the execution time estimation, dispatching, and packing. These components aim at finding the optimal solution for the problem formulation in (13) and the formulation in (18). The execution time for multi-job required by the dispatcher is 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 D_i^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.

Execution time estimation: When the set of multi-dependent tasks $\mathbb{C}$ are ready to be deployed, the collective execution time $\tilde{E}_{ex}\prime$ is first estimated. We train a ML regression model with historical data for this prediction task. The input to the prediction model is the set of runtime parameters $\boldsymbol{f}_{\mathrm{mt}}$, such as task dependency depth γ, resource demands $T^{\langle c,m \rangle \prime}$, data size δ and type of task ϵ, and the output is the execution time estimation $\tilde{E}_{ex}\prime$. Algorithm 1 describes the execution time estimation for multi-job. Once the execution time estimation value is extracted, it is used in the dispatching stage.

Algorithm 1 AirEdge: Execution Time Estimation

Input: Multi-Job $\mathbb{J}$ released at time t in location D_i^l, set of runtime parameters $\boldsymbol{f}_{\mathrm{mt}}$

Output: Execution time estimation $\sum_{J \in \mathbb{J}} \tilde{E}_{ex}\prime$ of a multi-job

```
for J_i ∈ 𝕁 do
    Data size of J_i = δ_{J_i}
    Dependency depth of J_i = γ_{J_i}
    for T_i ∈ J_i do
        T_i = ⟨c, m⟩, i.e., resource demand
        ML(f_mt)_{T_i} = Ẽ_{ex_{T_i}}
    end for
    Ẽ_{ex_{J_i}} = Ẽ_{ex_{J_i}} + Ẽ_{ex_{T_i}}
end for
```

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, 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. For the rationale of this strategy,

again consider the smart city solution, IQ Smart City, which provides an AI based multi-sensory analytic systems for video, sound and smell analytics, integrated with a V2X platform, i.e., Ericsson Connected Vehicle Platform (CVP)[6], to serve about 4.5 million active vehicles across more than 130 countries. Assume that there are updates from a given set of vehicles and sensors at location D_i^l and time t. Then it is better to deploy a closest drone having congruent resource and flight time availability to serve these vehicles and sensors at the same time, i.e., $\mathbb{J} \Rightarrow D_{edge_{i^\star}}$.

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., [29], [30], [43], [44], adopt the *closest* principle as the task offloading policy.

Algorithm 2 AirEdge: Dispatching Policy

Input: Multi-Job $\mathbb{J}$ released at time t within location D_i^l, execution time estimation $\sum_{J\in\mathbb{J}} \tilde{E}_{ex'}$, and federated edge-drone deployments $D_{edge_i} \in \mathbb{EDGE}$ update state
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** $\sum_{J\in\mathbb{J}} T^{\langle c,m\rangle\prime} \cong D_{edge_i}^{\langle c,m\rangle}$ and $\sum_{J\in\mathbb{J}} \tilde{E}_{ex'} \cong f_i$ **then**
3: **if** $D_{edge_{i^\star}} = \arg\min_{D_{edge_i}\in\mathbb{EDGE}} \left(f_i^l\right)$ **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**

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 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 [45] places the jobs on nodes by co-locating jobs tightly on each node. As multi-jobs arrive at the

[6]https://www.ericsson.com/en/internet-of-things/automotive/connected-vehicle-cloud

Algorithm 3 AirEdge: 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** $\sum_{I_i\in D_{edge_\star}} I_i$

1: **for** $I_i \in D_{edge_\star}$ **do**
2: **if** $\beta\left(I_i\right) = 1$ **then**
3: $I_i^{\langle c,m\rangle} = \langle c, m\rangle$, i.e., resource availability
4: **for** $J \in \mathbb{J}$ **do**
5: **if** $\Gamma\left[J, I_i\right] = 1$ **then**
6: $J \Rightarrow I_i$
7: $I_i^{\langle c,m\rangle} = I_i^{\langle c,m\rangle} + T^{\langle c,m\rangle\prime}$
8: **end if**
9: **end for**
10: **if** $I_i^{\langle c,m\rangle} \geq \langle c, m\rangle$ **then**
11: $i = i + 1$
12: **end if**
13: **end if**
14: **end for**

cluster $D_{edge_\star}$, the VSMCBP 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) [46], 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. Some jobs consist of a single task and do not have dependent or do not depend on other task(s), and such jobs are also co-located with other jobs.

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:

$$\textbf{Minimize} \sum_{I_i\in D_{edge_\star}} I_i, \tag{20}$$

$$\textit{subject to } \mathbb{J} \Rightarrow D_{edge_\star}, \tag{21}$$

$$\sum_{J\in\mathbb{J}} \Gamma\left[J,\ I_i\right] \cdot T^{\langle c,m\rangle\prime} \leq I_i^{\langle c,m\rangle}, \quad \forall_{c,m}, \tag{22}$$

$$\Gamma\left[J,\ I_i\right] = \begin{cases} 1, & \text{if } J \Rightarrow I_i, \\ 0, & \text{otherwise, } \forall I_i \in D_{edge_\star}. \end{cases} \tag{23}$$

The constraint (22) indicates that the total resource requirements of co-located jobs cannot exceed the node resource capacity or availability, while the condition (23) 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[7], which provides an interface to several mixed-integer programming (MIP) solvers, i.e., coin-or branch and cut (CBC)[8], is employed to solve this VSMCBP problem for

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

TABLE III
FEDERATED-EDGE RESOURCE CAPACITIES

Edge Deployments	Attached Edge Devices(s) and Total Weight	CPU Capacity (Cores)	Mem Capacity (GiB)
Drone 1	Snowcone + Huawei AR502H (3kg)	6 Cores	6 GiB
Drone 2	Dell 3000 + Dell 5000 + aiSage + dynaEdge (5.03kg)	12 Cores	28 GiB
Drone 3	HPE EL300 + Stack Edge mini (8.1kg)	20 Cores	58 GiB
Drone 4	INTELLIEDGE G700(x2) (11kg)	16 Cores	32 GiB
Drone 5	Lenovo SE350 + HIVECELL + Xavier NX + Dell 3000 (13.12kg)	30 Cores	274 GiB
Drone 6	Stack Edge mini(x4) + HIVECELL + Huawei AR502H (14.96kg)	74 Cores	202 GiB

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 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.

V. PERFORMANCE EVALUATION

We evaluate our AirEdge on real-time Alibaba cluster data traces. With the aid of the job execution time estimate $\tilde{E}_{ex}\prime$, we show that AirEdge can minimize the actual execution time of multi-dependent tasks, achieve high resource utilization and avoid loss of job in a federated aerial edge system. We conduct extensive experiments with orchestrated sets of multi-dependent tasks having heterogeneous resource requests across the computing resources. For each deployment, we compare our AirEdge with some existing state-of-the-arts.

A. System Setup

Computing Resources: We use 6 federated aerial edge deployments (drones), as summarized in Table III. The computing resources are made up of heterogeneous container-optimized nodes (container-instances). These drones have various resource capacities (up to 74 CPU cores and 202 GiB of memory) and weights (up to 15 kg). We assume that the selected drones have congruent flight time to execute ready applications. This assumption is reasonable as practical drones have such capacity. For example, the Easy Aerial Falcon[9] is an autonomous drone with load capacity up to 2 kg and flight time up to 45 minutes. The Bell ATP70[10] is another autonomous drone with load capacity up to 31 kg and can cover up to 35 miles on a single charge while carrying its maximum load.

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. An Alibaba cluster is a set of 4,034 machines, packed into racks, and connected by a high-bandwidth cluster network. Workload arrives at the cluster in the form of jobs. A job is comprised of one or more tasks, each of which is accompanied by a set of resource requirements used for scheduling the tasks onto machines. 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 201 jobs with total of 857 tasks (including dependencies) for our experiments. The number of tasks in each multi-job ranges from $(12,\ 302]$, while the task dependency depth among the jobs ranges from $(1,\ 18]$. Task dependencies [47] in Alibaba data trace is valuable for our investigation. In our experiments, we assume that all tasks are of high priority.

[9]https://easyaerial.com

[10]https://www.bellflight.com/products/bell-apt

B. Heuristics and Baselines

As explained previously, the closest heuristic or principle is widely adopted as the dispatching policy in distributed systems. Therefore, we fix the dispatching policy of the state-of-the-art dependency-aware task orchestration benchmarks compared to that of AirEdge, i.e., the closest heuristic. We compare our scheduling or packing strategy with the following three state-of-the-art schemes and the random approach.

1) **Graphene** [41] is a state-of-the-art approach in the literature 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** [42] 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** [40] 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 utilization, task deployment, scheduling and execution times. We use the cluster data trace from Alibaba to obtain resource requirements (CPU, Memory) and all the task dependencies. The multi-job execution information across the federated aerial edge deployments are listed in Table IV. The results obtained by AirEdge, Graphene, Tetris, Spear and Random are compared.

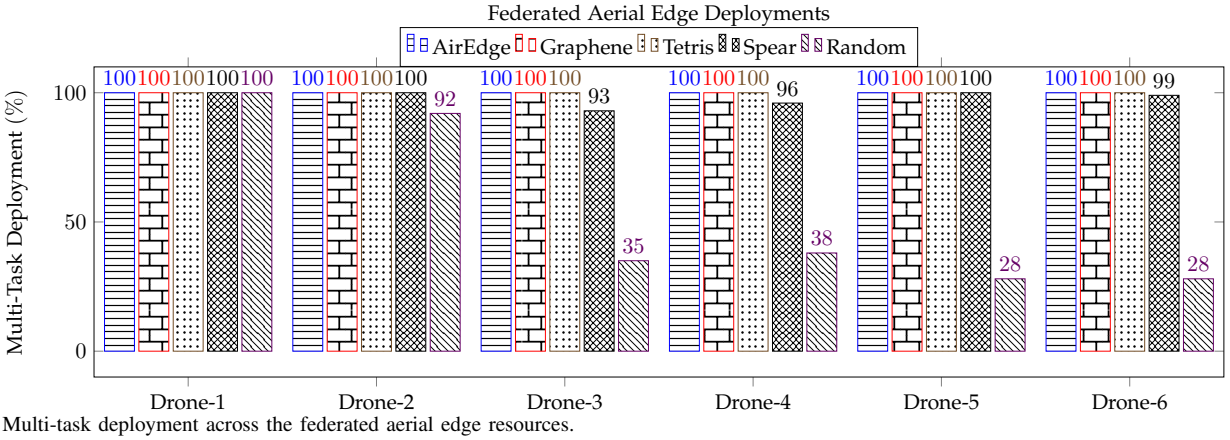


Fig. 6. Multi-task deployment across the federated aerial edge resources.

TABLE IV
MULTI-TASK EXECUTION IN FEDERATED AERIAL EDGE.

D_{edge_i}	$\mathbb{J}$	$\mathbb{C}$	λ	$\tilde{E}'_{ex}(s)$
Drone 1	1	6	5	63.67
Drone 2	3	13	(1, 6]	138.04
Drone 3	9	31	(1, 13]	522.38
Drone 4	11	26	(1, 4]	289.68
Drone 5	15	42	(1, 10]	468.19
Drone 6	30	103	(1. 13]	1313.45

We first investigate the capabilities of the five schemes compared to deploy the required tasks across the six drone resources. The multi-task deployments of all the five schemes across the federated aerial edge resources are depicted in Fig. 6. It can be see that AirEdge, Tetris and Graphene all are able to deploy 100% of all the tasks. Spear is slightly inferior and could not deploy 100% of the tasks across all the six drones. Specifically, it can only achieve 93%, 96% and 99% of the multi-task deployments on Drones 3, 4 and 6, respectively. As expected, Random is the worst and the percentage of its deployed tasks is much lower. Random approach deploys a task randomly to any available node, and assumes that a node can only execute a task at a time. This results in resource under utilization and inability to deploy all its tasks, given the available resources. Note that in the case that it is impossible to deploy 100% of the tasks on a drone, other drone, which has the additional congruent resource and flight time availability, will have to deploy and execute these remaining tasks. In the following performance comparison, however, we only show the results for the tasks which are deployed successfully in the federated aerial deployments.

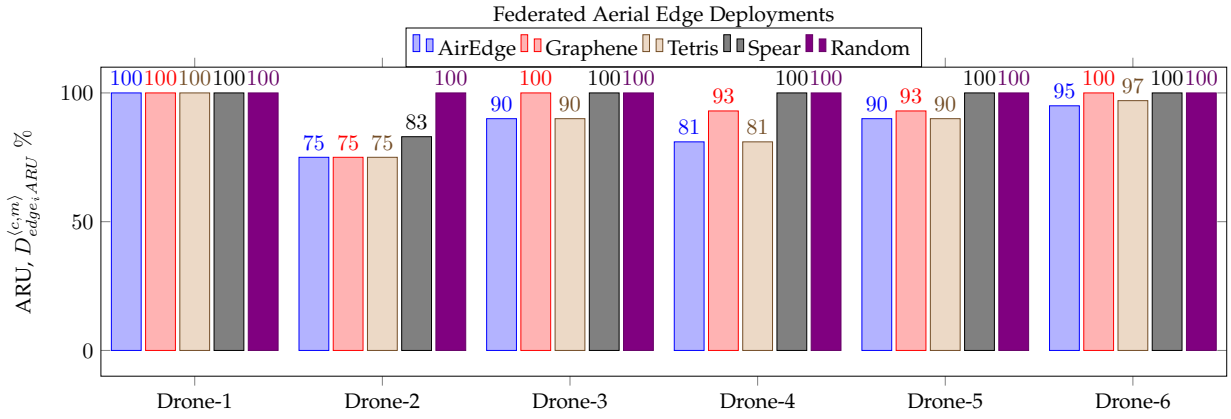


Fig. 7. Actual resource usage across the federated aerial edge clusters.

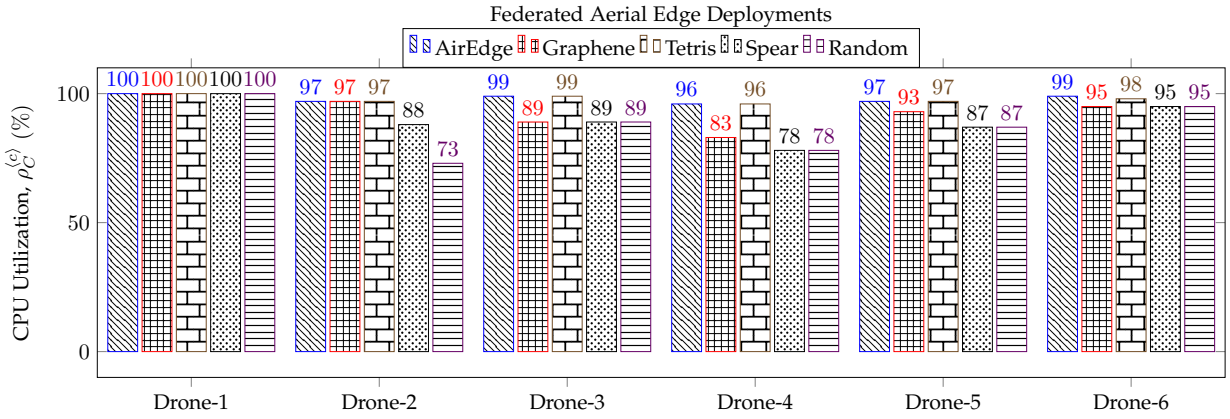


Fig. 8. CPU utilization across the federated aerial edge resources.

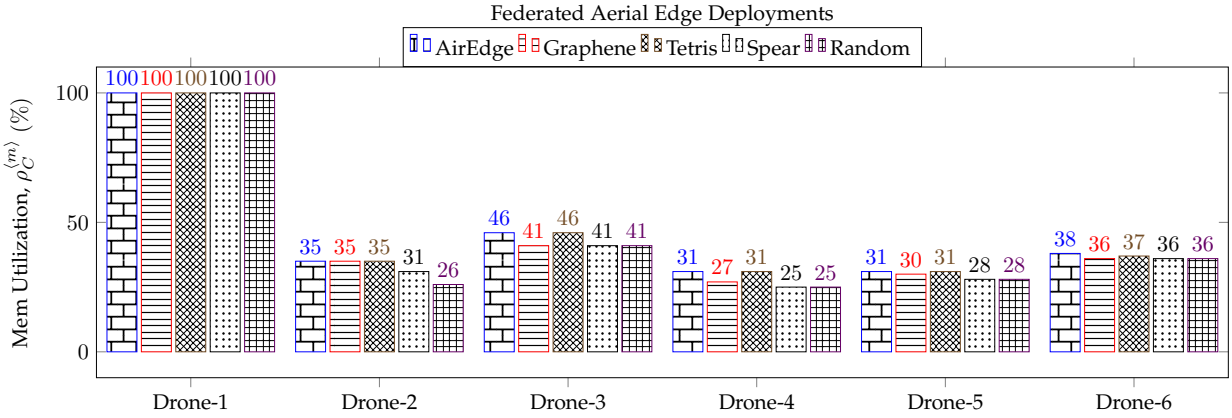


Fig. 9. Memory utilization across the federated aerial edge resources.

1) Performance comparison across federated aerial edge: We first introduce a performance metric, the actual resources usage of jobs $D^{\langle c,m \rangle}_{edge_i ARU}$, which is defined as the ratio of the resources used for execution $D^{\langle c,m \rangle}_{edge_i U}$ over the edge's resource capacity or availability $D^{\langle c,m \rangle}_{edge_i}$:

$$D^{\langle c,m \rangle}_{edge_i ARU} = \frac{D^{\langle c,m \rangle}_{edge_i U}}{D^{\langle c,m \rangle}_{edge_i}}. \tag{24}$$

Another metric, the resources utilization $\rho_C^{\langle c,m \rangle}$, is already given in (9). The metric $\rho_C^{\langle c,m \rangle}$ includes the CPU utilization $\rho_C^{\langle c \rangle}$ and the memory utilization $\rho_C^{\langle m \rangle}$, which can be defined respectively according to

$$\rho_C^{\langle c \rangle} = \frac{\sum_{J \in \mathbb{J}} T^{\langle c \rangle \prime}}{D^{\langle c \rangle}_{edge_i U}}, \tag{25}$$

$$\rho_C^{\langle m \rangle} = \frac{\sum_{J \in \mathbb{J}} T^{\langle m \rangle \prime}}{D^{\langle m \rangle}_{edge_i U}}, \tag{26}$$

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. 7 compares the actual resource usage of AirEdge with those of the three baseline schemes and the random approach. It can be seen that AirEdge consumes the fewest resources in the clusters with Tetris as the very close second best, while Random uses the highest resources with Spear as the second worst. Graphene ranks in the middle, in terms of resource usage. The CPU and memory resource utilization comparisons are shown in Figs. 8 and 9, respectively. Again, AirEdge and Tetris are superior than Graphene, Spear and Random, and they achieve the highest and close second highest resource utilization, respectively, while Spear and Random achieve the second lowest and lowest resource utilization, respectively.

Two other key metrics are the actual multi-task scheduling time and, more importantly, the actual multi-task execution time. Figs. 10 and 11 compares the actual multi-task scheduling time and multi-task execution time of AirEdge with those of the four benchmarks, respectively. The results show that AirEdge is the best, Tetris is the second best, and Graphene is the third best, while Random is the worst and Spear the second worst, in terms of both scheduling and execution times. Moreover, the superior performance of AirEdge over the other benchmark schemes is overwhelmingly clear.

2) Performance comparison in individual clusters: Having compared the performance of all the five schemes across the entire federated aerial edge, in terms of actual resource usage and resource utilization, task scheduling and execution times, which are shown in Figs. 7 to 11, respectively, we now exam the the performance of all the five schemes in individual clusters in detail.

In `Drone-1` edge-cluster, we deploy 1 job with a total of 6 tasks, where the job has a task dependency depth of 5. AirEdge 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, AirEdge co-schedules all the 6 tasks at a time. These tasks are tightly packed on nodes using the VSMCBP algorithm, which uses all the nodes in the cluster to execute the job. Note that this cluster is a small-capacity cluster, having 6 CPU cores and 6 GiB of memory. Using the same configuration for the baseline schemes, Graphene, Tetris and Spear also utilize the full resources. The random approach utilizes the full resources as well. Importantly, AirEdge has the fastest scheduling and execution times compared to the three state-of-the-art schemes and the random approach, mainly due to the following reasons: (i) AirEdge utilizes the gang scheduling to co-schedule all the tasks at a time, and (ii) its packing strategy explores the available nodes to find the best node, which has the requisite available resources to execute all the tasks, by packing them tightly on the node. Observe that AirEdge is more than 2 times faster than the second best Tetris in both the scheduling and execution times. It is more than 3 times and 2 times faster than Graphene as well as 9 times and 3 times faster than Spear in the scheduling and execution times, respectively. Compared with the worst Random, AirEdge is 48 times and 6 times faster in terms of scheduling and execution times, respectively.

`Drone-2` cluster is also a small-capacity cluster but with higher CPU and memory capacity than `Drone-1`. Here, 3 jobs with a total of 13 tasks are deployed, where each job has a task dependency in the range of $(1,\ 6]$. We optimize the deployment to ensure that resources are fully utilized. Containers provide isolation to running applications, making it possible to co-locate multiple applications on the same node without any interference. A single container-optimized

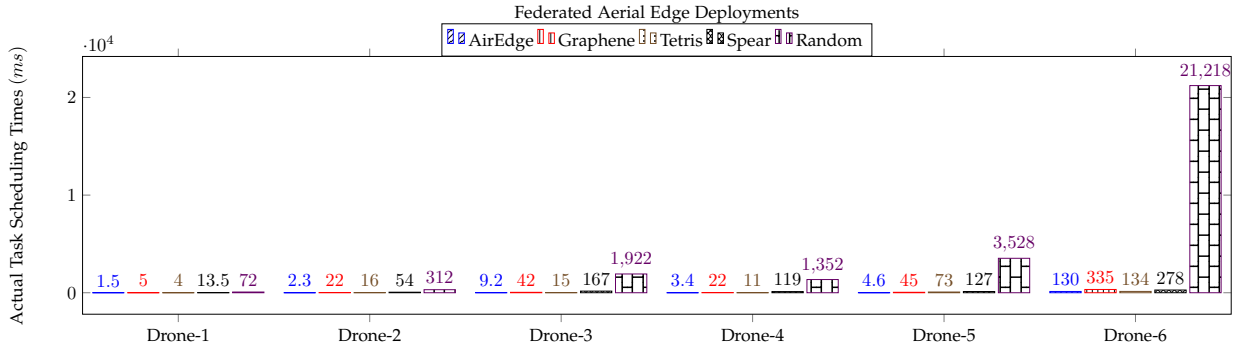


Fig. 10. Actual multi-task scheduling time across the federated aerial edge resources.

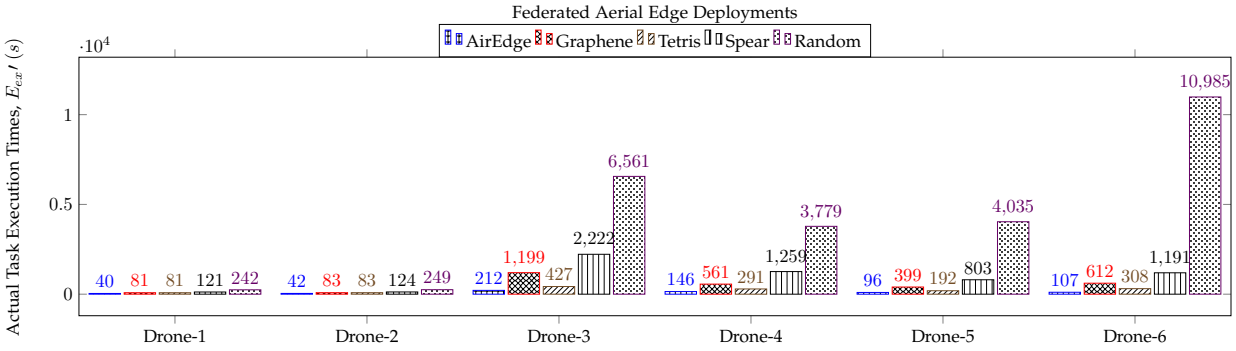


Fig. 11. Actual multi-task execution time across the federated aerial edge resources.

node can execute more containerized applications, given that there are sufficient available resources. AirEdge, Graphene and Tetris consume 8% fewer resources than Spear, and 25% fewer resources than Random. AirEdge, Graphene and Tetris also gain 9% and 24% higher CPU utilization over Spear and Random, respectively, as well as 4% and 9% higher memory utilization than Spear and Random, respectively. More significantly, AirEdge is 9, 6 and 23 times faster in the scheduling time than Graphene, Tetris and Spear, respectively, while it is 2, 2 and 3 times faster in the execution time over these state-of-the-art schemes, 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 92% of the tasks that it is able to deploy on `Drone-2`.

`Drone-3` has high load capacity (up to 8kg) compared to `Drone-1` and `Drone-2`. This cluster is made up of 1 `HPE EL300` and 1 `Stack Edge mini` edge devices, with total resource capacity of 20 Cores and 28 GiB. In this cluster, we deploy 9 jobs, with total 31 tasks, where each job has a task dependency range $(1,\ 13]$. AirEdge and Tetris reduce resource usage by 10% compared with Graphene, Spear and Random. AirEdge and Tetris achieve 10% and 5% higher CPU and memory utilization, respectively, compared to Graphene, Spear and Random. In terms of both scheduling and execution times, AirEdge is about 2 times faster than Tetris. It is 4 times and 5 times faster than Graphene as well as 18 times and 10 times faster than Spear, in the scheduling and execution times, respectively. Not surprisingly, Random has the worst scheduling and execution time performance.

`Drone-4` edge-cluster is a high capacity cluster. Here, 11 jobs with total of 26 tasks are deployed, where each job has a dependency depth range $(1,\ 4]$. It can be seen that AirEdge and Tetris consume 12% and 19% fewer resources than Graphene and Spear, respectively. AirEdge and Tetris also achieve 13% and 18% higher CPU utilization as well as 4% and 6% higher memory utilization, over Graphene and Spear, respectively. It is worth pointed out that although Spear and the Random approach utilize the same amount of resources, Random can only deploy 38% of the tasks but Spear deploys 96% of the tasks. By contrast, AirEdge, Graphene and Tetris all deploy 100% of the tasks. In terms of scheduling time, AirEdge is approximately 7 times, 3 times and 30 times faster than Graphene, Tetris and Spear, respectively. In terms of execution time, AirEdge is about 4 times, 2 times and 8 times faster than Graphene, Tetris and Spear, 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 15 and 30 jobs in these two clusters, respectively. The total number of tasks deployed in `Drone-5` cluster is 42, while total of 103 tasks are deployed in `Drone-6` cluster. The task dependency depth of each job is in the range of $(1,\ 13]$. Observe that Random can only deploy 28% of the tasks in these two resource and capacity rich drone clusters, respectively.

For `Drone-5` edge-cluster, AirEdge and Tetris use 3% and 10% less resources, compared with Graphene and Spear, respectively. AirEdge and Tetris also achieve 4% and 10% higher CPU utilization as well as 1% and 3% higher memory utilization than Graphene and Spear, respectively. In terms of scheduling time, AirEdge is approximately 10 times, 16 times, 30 times and 800 times faster than Graphene, Tetris, Spear and Random, respectively. In terms of execution time, AirEdge is about 4 times, 2 times, 8 times and 40 times faster than Graphene, Tetris, Spear and Random, respectively.

For `Drone-6` edge-cluster, AirEdge uses 2% less resources than Tetris as well as 5% less resources than Graphene and

Spear. AirEdge also achieve 1% higher CPU utilization than Tetris as well as 4% higher CPU utilization than Graphene and Spear. In terms of memory utilization, AirEdge is 1% higher than Tetris as well as 2% higher than Graphene and Spear. In terms of scheduling time, AirEdge is slightly faster than Tetris as well as 2 times faster than Graphene and Spear. In terms of execution time, AirEdge is 6 times faster, 3 times faster and 11 times faster than Graphene, Tetris and Spear, respectively. AirEdge is 160 times faster and 100 times faster than Random in the scheduling time and execution time, respectively.

D. Summary

Overall, AirEdge has demonstrated superior QoS in resource management and multi-task orchestration in federated edge clusters. Our algorithm consistently achieves both the highest cluster resource utilization as well as the fastest scheduling and execution times for multi-tasks/jobs, compared to the three state-of-the-arts, Graphene, Tetris and Spear. Increasing resource utilization by just a few percentage points can save millions of dollars in large scale-computing. Achieving faster scheduling time and in particular faster execution time are crucial for modern applications to perform better.

The gains achieved by AirEdge as observed from our experiments include load-balancing in a distributed and federated edge clusters and an increase in the number of tasks that can be deployed at a time as well as faster execution time of the overall tasks and improved usage of cluster resources. The significant advantage of AirEdge can be explained as follows. It deploys sets of multi-jobs/tasks as a unit through the gang scheduling strategy, and these applications are deployed and executed concurrently. Unlike AirEdge, the existing state-of-the-art methods do not deploy all ready tasks at a time or do not respect task dependencies, leading to more resource usage and cluster under utilization as well as causing scheduling delay and longer task execution times.

VI. Conclusions

This paper has presented a dependency-aware multi-task orchestration in a federated aerial edge computing system, called AirEdge, to improve resource efficiency and enhance performance. We have utilized a resource-specific dispatching strategy that selects the closest edge cluster or drone suitable for given job(s) based on estimated value of their execution time, and a container-based bin packing optimization strategy that packs or co-locates tasks tightly on nodes to fully utilize available resources. Our approach involves capturing the high-level resource request of tasks, federated aerial edge clusters update state service, execution time estimation, gang scheduling and co-locating multi-task on container-optimized nodes called *container-instances*. To evaluate our approach, we have illustrated use cases of real-world CPU and memory intensive tasks from Alibaba cluster trace, which records the activities of both long running containers (for Alibaba's e-commerce business) and batch jobs across an 8-day period. We have compared our approach with the state-of-the-art dependency-aware task orchestration and task packing baseline strategies. Our experimental results have demonstrated that AirEdge achieves both the highest cluster resource utilization and the fastest execution time for multi-tasks/jobs compared to the existing state-of-the-art strategies.

References

[1] J. Ren, *et al.*, "A survey on end-edge-cloud orchestrated network computing paradigms: Transparent computing, mobile edge computing, fog computing, and cloudlet," *ACM Computing Survey*, vol. 52, no. 6, pp. 125:1–125:36, 2019.

[2] S. Liu, *et al.*, "Edge computing for autonomous driving: Opportunities and challenges," *Proc. IEEE*, vol. 107, no. 8, pp. 1697–1716, Aug. 2019.

[3] S. U. Amin and M. S. Hossain, "Edge intelligence and Internet of Things in healthcare: A survey," *IEEE Access*, vol. 9, pp. 45–59, 2021.

[4] Z. Tan, *et al.*, "UAV-aided edge/fog computing in smart IoT community for social augmented reality," *IEEE Internet of Things J.*, vol. 7, no. 6, pp. 4872–4884, Jun. 2020.

[5] L. U. Khan, *et al.*, "Edge-computing-enabled smart cities: A comprehensive survey," *IEEE Internet of Things J.*, vol. 7, no. 10, pp. 10200–10232, Oct. 2020.

[6] R. Mahmud, A. N. Toosi, K. Ramamohanarao, and R. Buyya, "Context-aware placement of industry 4.0 applications in fog computing environments," *IEEE Trans. Industrial Informatics*, vol. 16, no. 11, pp. 7004–7013, Nov. 2020.

[7] H. Guo and J. Liu, "UAV-enhanced intelligent offloading for Internet of Things at the edge," *IEEE Trans. Industrial Informatics*, vol. 16, no. 4, pp. 2737–2746, Apr. 2020.

[8] Z. Yu, Y. Gong, S. Gong, and Y. Guo, "Joint task offloading and resource allocation in UAV-enabled mobile edge computing," *IEEE Internet of Things J.*, vol. 7, no. 4, pp. 3147–3159, Apr. 2020.

[9] X. Zhang, *et al.*, "Resource allocation for a UAV-enabled mobile-edge computing system: Computation efficiency maximization," *IEEE Access*, vol. 7, pp. 113345–113354, 2019.

[10] 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, Oct. 2020.

[11] 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

[12] L. Yang, *et al.*, "Multi-UAV-enabled load-balance mobile-edge computing for IoT networks," *IEEE Internet of Things J.*, vol. 7, no. 8, pp. 6898–6908, Aug. 2020.

[13] C. Zhan, *et al.*, "Multi-UAV-enabled mobile edge computing for time-constrained IoT applications," *IEEE Internet of Things J.*, early access, doi:10.1109/JIOT.2021.3073208.

[14] Y. Wang, Z.-Y. Ru, K. Wang, and P.-Q. Huang, "Joint deployment and task scheduling optimization for large-scale mobile users in multi-UAV-enabled mobile edge computing," *IEEE Trans. Cybernetics*, vol. 50, no. 9, pp. 3984–3997, Sep. 2020.

[15] J. Zhang *et al.*, "Computation-efficient offloading and trajectory scheduling for multi-UAV assisted mobile edge computing," *IEEE Trans. Vehicular Technology*, vol. 69, no. 2, pp. 2114–2125, Feb. 2020.

[16] Y. Luo, W. Ding, and B. Zhang, "Optimization of task scheduling and dynamic service strategy for multi-UAV-enabled mobile-edge computing system," *IEEE Trans. Cognitive Communications and Networking*, vol. 7, no. 3, pp. 970–984, Sep. 2021.

[17] A. M. Seid, *et al.*, "Multi-agent DRL for task offloading and resource allocation in multi-UAV enabled IoT edge network," *IEEE Trans. Network and Service Management*, early access, doi:10.1109/TNSM.2021.3096673.

[18] X. Cao, *et al.*, "Edge federation: Towards an integrated service provisioning model," *IEEE/ACM Trans. Networking*, vol. 28, no. 3, pp. 1116–1129, Jun. 2020.

[19] 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. 19-23, 2020, pp. 91-98.

[20] M. Jia, J. Cao, and W. Liang, "Optimal cloudlet placement and user to cloudlet allocation in wireless metropolitan area networks," *IEEE Trans. Cloud Computing*, vol. 5, no. 4, pp. 725–737, 2017.

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

[22] 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.

[23] T. Pham, J. J. Durillo, and T. Fahringer, "Predicting workflow task execution time in the cloud using a two-stage machine learning approach," *IEEE Trans. Cloud Computing*, vol. 8, no. 1, pp. 256–268, 2020.

[24] F. Nadeem, *et al.*, "Using machine learning ensemble methods to predict execution time of e-science workflows in heterogeneous distributed systems," *IEEE Access*, vol. 7, pp. 25138–25149, 2019.

[25] L. Larsson, H. Gustafsson, C. Klein, and E. Elmroth, "Decentralized Kubernetes federation control plane," in *Proc. IEEE/ACM UCC 2020* (Leicester, UK), Dec. 7-10, 2020, pp. 354–359.

[26] Y. Han, *et al.*, "Tailored learning-based scheduling for Kubernetes-oriented edge-cloud system," in *Proc. INFOCOM 2021* (Vancouver, BC, Canada), May 10-13, 2021, pp. 1–10.

[27] V. S. Marco, B. Taylor, B. Porter, and Z. Wang, "Improving spark application throughput via memory aware task co-location: A mixture of experts approach," in *Proc. Middleware 2017* (Las Vegas, NV, USA), Dec. 11-15, 2017, pp. 95–108.

[28] Y. Li, D. Sun, and B. C. Lee, "Dynamic colocation policies with reinforcement learning," in *ACM Trans. Architecture and Code Optimization*, vol. 17, no. 1, pp. 1:1–1:25, Mar. 2020.

[29] 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.

[30] 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.

[31] C. Shu, *et al.*, "Multi-user offloading for edge computing networks: A dependency-aware and latency-optimal approach," *IEEE Internet of Things J.*, vol. 7, no. 3, pp. 1678–1689, Mar. 2020.

[32] J. Liu and H. Shen, "Dependency-aware and resource-efficient scheduling for heterogeneous jobs in clouds," in *Proc. CloudCom 2016* (Luxembourg City, Luxembourg), Dec. 12-15, 2016, pp. 110–117.

[33] J. Lee, H. Ko, J. Kim, and S. Pack, "DATA: Dependency-aware task allocation scheme in distributed edge clouds," *IEEE Trans. Industrial Informatics*, vol. 16, no. 12, pp. 7782–7790, Dec. 2020.

[34] Y. Liu, *et al.*, "Dependency-aware task scheduling in vehicular edge computing," *IEEE Internet of Things J.*, vol. 7, no. 6, pp. 4961–4971, Jun. 2020.

[35] M. H. Hilman, M. A. Rodriguez, and R. Buyya, "Task runtime prediction in scientific workflows using an online incremental learning approach," in *Proc. UCC 2018* (Zurich, Switzerland), Dec. 17-20, 2018, pp. 93–102.

[36] 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.

[37] W. Xiao, *et al.*, "Gandiva: Introspective cluster scheduling for deep learning," in *Proc. OSDI 2018* (Carlsbad, CA, USA), Oct. 8-10, 2018, pp. 595–610.

[38] S. Venkataraman, *et al.*, "Ernest: Efficient performance prediction for large-scale advanced analytics," in *Proc. OSDI 2016* (Santa Clara, CA, USA) Mar. 16-18, 2016, pp. 363–378.

[39] Y. Peng, *et al.*, "Optimus: An efficient dynamic resource scheduler for deep learning clusters," in *Proc. EuroSys 2018* (Porto, Portugal), Apr. 23-26, 2018, pp. 1–14.

[40] 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.

[41] 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.

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

[43] H. Tan, Z. Han, X. Li, and F. C. M. Lau, "Online job dispatching and scheduling in edge-clouds," in *Proc INFOCOM 2017* (Atlanta, GA, USA), May 1-4, 2017, pp. 1–9.

[44] 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, Dec. 2019.

[45] 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.

[46] S. Rampersaud and D. Grosu, "Sharing-aware online virtual machine packing in heterogeneous resource clouds," *IEEE Trans. Parallel and Distributed Systems*, vol. 28, no. 7, pp. 2046–2059, Jul. 2017.

[47] H. Tian, Y. Zheng, and W. Wang "Characterizing and synthesizing task dependencies of data-parallel jobs in Alibaba cloud," in *Proc. SoCC 2019* (Santa Cruz, CA, USA), Nov. 20-23, 2019, pp. 139–151.