Digital Communications and Networks 10 (2024) 1837–1850

Contents lists available at ScienceDirect

Digital Communications and Networks

journal homepage: www.keaipublishing.com/dcan

Collaborative learning-based inter-dependent task dispatching and co-location in an integrated edge computing system ☆

Uchechukwu Awada [a,b], Jiankang Zhang [c,*], Sheng Chen [d,e], Shuangzhi Li [b,*], Shouyi Yang [b]

[a] School of Software, Henan Institute of Science and Technology, Xinxiang 453003, China
[b] School of Electrical and Information Engineering, Zhengzhou University, Zhengzhou 450001, China
[c] Department of Computing and Informatics, Bournemouth University, Poole BH12 5BB, UK
[d] School of Electronics and Computer Science, University of Southampton, Southampton SO17 1BJ, UK
[e] Faculty of Information Science and Engineering, Ocean University of China, Qingdao 266100, China

ARTICLE INFO

Keywords:
Edge computing
Collaborative learning
Resource utilization
Execution time
Edge federation
Gang scheduling

ABSTRACT

Recently, several edge deployment types, such as on-premise edge clusters, Unmanned Aerial Vehicles (UAV)-attached edge devices, telecommunication base stations installed with edge clusters, etc., are being deployed to enable faster response time for latency-sensitive tasks. One fundamental problem is where and how to offload and schedule multi-dependent tasks so as to minimize their collective execution time and to achieve high resource utilization. Existing approaches randomly dispatch tasks naively to available edge nodes without considering the resource demands of tasks, inter-dependencies of tasks and edge resource availability. These approaches can result in the longer waiting time for tasks due to insufficient resource availability or dependency support, as well as provider lock-in. Therefore, we present *EdgeColla*, which is based on the integration of edge resources running across multi-edge deployments. *EdgeColla* leverages *learning* techniques to intelligently *dispatch* multi-dependent tasks, and a variant bin-packing optimization method to *co-locate* these tasks firmly on available nodes to optimally utilize them. Extensive experiments on real-world datasets from Alibaba on task dependencies show that our approach can achieve optimal performance than the baseline schemes.

1. Introduction

Edge Computing (EC) is a distributed computing model which places cloud computing [1] services closer to data sources to achieve faster response time and real-time insights. Devices can offload their computational intensive tasks and latency-sensitive tasks to the edge and after executions, the results are sent back to the devices. To this end, several independent edge deployment types, such as on-premise edge clusters [2], Unmanned Aerial Vehicles (UAV)-enabled EC [3,4], telecommunication base stations endowed with edge clusters,[1] edge nodes [2], etc., have been proposed. However, one fundamental problem is where and how to offload and schedule multi-dependent tasks in such diverse deployments so that their collective execution time is minimized and high resource utilization is achieved. A common practice is to randomly offload tasks individually to available edge nodes without considering the resource demands of tasks, inter-dependencies of tasks and edge resource availability, as shown in Fig. 1 (a). Such a disjointed approach would result in the longer waiting time for tasks due to insufficient resource availability, dependency support, and vendor lock-in situations. Hence it is not appropriate for latency-sensitive tasks.

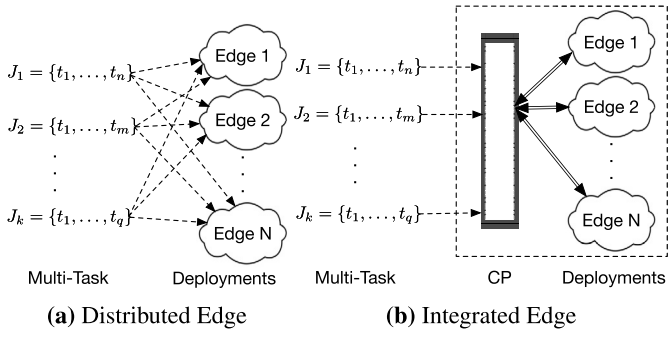
For this reason, we wish to consider an approach that can seamlessly integrate all edge resources running across N deployments (i.e., on-premise edge clusters, edge nodes, telecommunication base stations equipped with edge clusters, and UAVs attached with edge devices) in a single pool as shown in Fig. 1 (b), such that these resources can be holistically monitored from a Control Plane (CP), and multiple tasks can be dispatched dynamically across these edge resources. This approach

☆ Peer review under the responsibility of the Chongqing University of Posts and Telecommunications.

* Corresponding authors.
E-mail addresses: awada@hist.edu.cn (U. Awada), jzhang3@bournemouth.ac.uk (J. Zhang), sqc@ecs.soton.ac.uk (S. Chen), ielsz@zzu.edu.cn (S. Li), iesyyang@zzu.edu.cn (S. Yang).

[1] https://aws.amazon.com/wavelength/.

https://doi.org/10.1016/j.dcan.2024.08.002
Received 9 January 2022; Received in revised form 31 July 2024; Accepted 2 August 2024
Available online 13 August 2024



(a) Distributed Edge **(b)** Integrated Edge

Fig. 1. (a) An example of random multi-tasks dispatching without considering their dependencies and cluster resource status, and (b) An example of intelligent multi-tasks dispatching, where both tasks dependencies and cluster resource status are considered.

is called *Edge Federation* (EF) [2,5]. For example, recently introduced EC frameworks, i.e., KubeEdge,[2] MicroK8s,[3] etc., have the capabilities of integrating edge resources running across multiple deployments for containerized tasks to eliminate provider lock-in situations. One of the benefits that EF brings is minimized latency by serving devices from the cluster closest to them [2,5–7]. The EF setup consists of a `host cluster` and `member cluster(s)`. Given N independent edge deployments, the CP is deployed in one of the deployments as the `host cluster`, while the remaining $N-1$ deployments are regarded `member cluster(s)`, which can be added or removed from the CP. The EF system is given as

$$\mathbb{EF} = \bigcup_{i=1}^{N} \text{Edge}_i \tag{1}$$

Through the CP, resource availability status, as well as running task status can be obtained from all the deployments (`host cluster` and `member cluster(s)`), thus enabling informed decisions on optimal multi-task dispatching. The work presented in this paper differs considerably from prior works [6,7], which addressed the problem of multi-dependent task orchestration in a federated autonomous drone-enabled EC system, while considering the drones' flight time to avoid the loss of jobs [8].

In this paper, we present *EdgeColla*, which leverages the Collaborative Learning (CL) technique [9–11] to estimate multi-task resource requirements and execution time, and to dispatch these tasks to the closest `member cluster` having matching available resources, while considering their dependencies. The effectiveness of such a CL-based multi-task dispatching method in N edge deployments is critically dependent on the state information update process, in terms of the resource availability of all the clusters. One drawback of this concept is that the inaccurate estimation of the multi-task resource requirements and execution time would cause EF to perform poorly. Similarly, if multi-dependent tasks are randomly dispatched, e.g., in an offloading strategy that dispatches tasks individually without considering their dependencies and cluster resource status [12,13], EF might not yield optimal performance. Therefore, we first investigate the accuracy of our trained linear regression model by estimating the resource requirements and execution time of multi-dependent tasks, using the Normalized Absolute Estimate Error (NAEE) method. This serves as the estimation accuracy measure for the trained linear regression model. Then we adopt the gang-scheduling [14] strategy and a variant bin-packing optimization method to efficiently co-schedule and co-locate all the tasks, where both their dependencies and cluster resource status are considered, such that their **actual** completion time is minimized, as well as the optimal resource usage is achieved. To avoid interference and resource contention among co-located tasks, we provide isolation to co-located tasks through containerization [15]. Containerization provides isolation to running tasks and enables tasks to be executed in any edge deployment regardless of the architecture or provider.

We summarize the main contributions of our EdgeColla implementation as follows:

- An intelligent multi-dependent task dispatching method through the joint optimization of their resource requirements and cluster resource status is proposed.
- Specifically, we derive a CL-based multi-dependent task resource requirement, and execution time, and cluster resource status estimation approach for an integrated edge system through the CP, such that multi-dependent tasks are intelligently dispatched to the *closest* edge cluster having sufficient available resources.
- To guarantee the optimal usage of cluster resources, we further propose a variant bin-packing optimization approach through gang-scheduling of multi-dependent tasks, which co-schedules and co-locate tasks firmly on available nodes to avoid resource wastage.
- We show that EdgeColla is capable of minimizing the actual completion time of multi-dependent tasks using minimum resources and we conduct extensive experiments to compare the performance of EdgeColla with several existing approaches on real-world datasets from Alibaba,[4] which provides information on task dependencies.

1.1. Motivating examples

Dependency-awareness is critical for achieving optimal performance in task dispatching and scheduling problems. In Fig. 2, we show an example of multi-dependent tasks, where some of the tasks require the output of another task(s), as well as other resources,[5] i.e., CPU $\langle c \rangle$ and memory $\langle m \rangle$, for its execution. For example, T_1, T_2, and T_3 are independent tasks, i.e., they do not have dependencies and they can start executing without requiring input from other tasks. Tasks T_4 and T_5 require input from T_1 to be able to complete their executions. Similarly, tasks T_6, T_7, and T_8 depend on the completion of tasks T_4, T_5, and T_2, respectively. Deploying these tasks on the same cluster would enable dependent tasks to communicate and share data faster, compared to individual tasks execution across different clusters [16]. The complex inter-task dependency with heterogeneous resource demands and diverse edge deployments with heterogeneous resource capacities make resource management in EC a non-trivial task. Considering such demands and resource capacities is necessary to achieve effective dispatching and scheduling, ultimately to achieve optimal performance [17,18]. Hence a key objective of our EdgeColla is to reduce the collective execution time of such tasks and to improve cluster resource usage by considering inter-task dependencies.

Given n multi-dependent tasks $T_1, T_2, \cdots, T_n$ as shown in Fig. 2, EdgeColla adopts the gang-scheduling [14] strategy and a variant bin-packing optimization method to efficiently co-schedule and co-locate them in a cluster. We consider EdgeColla as a Full Dependency and Full Packing (FDFP) approach. Therefore, the scheduling time can be expressed as

$$\sum_{z=1}^{m} \sum_{i=1}^{k_z} S_{ch_{z_i}} / k_z \tag{2}$$

where m is the number of scheduling units and k_z is the number of tasks within the z-th scheduling unit having tasks $\{T_{z_1}, T_{z_2}, \cdots, T_{z_{k_z}}\}$.

We illustrate the advantage of the scheduling approach in EdgeColla over 3 other existing schemes as follows. (i) An approach that does

[2] https://kubeedge.io/en/.
[3] https://microk8s.io/.
[4] https://github.com/alibaba/clusterdata/blob/master/cluster-trace-v2018/trace-2018.md.
[5] Here we focus on CPU and memory resources, since these resources are limited in edge systems.

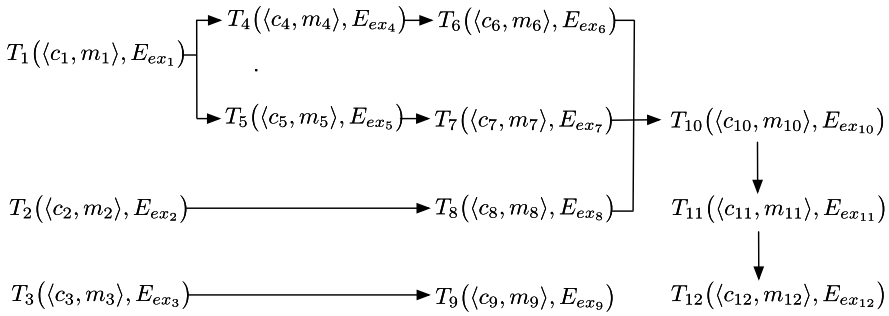


Fig. 2. An example of multi-dependent tasks, with each task's CPU and memory resource requirements denoted as $\langle c, m\rangle$, and execution time denoted as E_{ex}.

Table 1
Scheduling orders and units of various schemes.

Scheme	Scheduling Order	Scheduling Units
EdgeColla	$\{T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8, T_9, T_{10}, T_{11}, T_{12}\}$	1
PDNP	$T_3 \rightarrow T_2 \rightarrow \{T_1, T_4\} \rightarrow \{T_6, T_8\} \rightarrow \{T_5, T_7\} \rightarrow \{T_{10}, T_{11}\} \rightarrow \{T_{12}, T_9\}$	7
PDFP	$T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow \{T_4, T_5\} \rightarrow \{T_6, T_7, T_8, T_9\} \rightarrow \{T_{10}, T_{11}, T_{12}\}$	6
NDFP	$\{T_1, T_2, T_3, T_4, T_5, T_6\} \rightarrow \{T_7, T_8, T_9, T_{10}, T_{11}, T_{12}\}$	2
Random	$T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_4 \rightarrow T_5 \rightarrow T_6 \rightarrow T_7 \rightarrow T_8 \rightarrow T_9 \rightarrow T_{10} \rightarrow T_{11} \rightarrow T_{12}$	12

not consider task dependency but schedules 50% of any given multi-dependent task by mainly focusing on task co-location. We refer to this approach as No Dependency and Full Packing (NDFP), and it is similar to the approach in [19]. (ii) An approach that schedules up to 15% of any given multi-dependent tasks at a time, but does not consider task co-location. We refer to this approach as Partial Dependency and No Packing (PDNP), which is similar to the approach in [20]. (iii) An approach that schedules up to 40% of any given multi-dependent task with task co-location. We consider this approach as Partial Dependency and Full Packing (PDFP), which is similar to the approach in [21]. (iv) Finally the Random approach does not consider both task dependencies and task co-location. We refer to this approach as No Dependency and No Packing (NDNP). It is important to note that delays in scheduling inter-dependent tasks directly impact their collective execution time. For the multi-dependent tasks in Fig. 2 with $n = 12$ tasks, Table 1 lists the scheduling orders and scheduling units for the schemes compared. EdgeColla only needs one scheduling unit (m=1) that has k_1=12 tasks and it also achieves the lowest execution time of $\frac{1}{12}\sum_{i=1}^{12} E_{ex_i}$. By contrast, Random has m=12 scheduling units, each having a single task. Hence it has the highest execution time of $\sum_{i=1}^{12} E_{ex_i}$. Thus, EdgeColla achieves the lowest scheduling and execution time. PDNP, PDFP, and NDFP deploy individual or subsets of tasks at a time.

The remaining parts of this paper are structured as follows. Section 2 presents related work on learning-based resource allocation schemes used in cloud and edge computing. In Section 3, we detail our proposed EdgeColla for achieving high resource utilization and minimizing the execution time of applications deployed on EF resources. In Section 4, we compare the performance of our proposed EdgeColla against several existing schemes through extensive experiments. Finally, we conclude the paper in Section 5.

2. Related work

Effective multi-task dispatching techniques in edge systems can benefit from resource availability status, multi-task resource requirements and execution time, such that these tasks can be offloaded to the closest edge cluster with sufficient resources. Information about task execution time is most important for drone-based edge deployments [6,7]. This is because a typical drone has limited flight time, which could possibly lead to a delayed task execution if it is not taken into consideration [8]. Hence, the effective and accurate execution time estimation of multiple task is needed to select a drone with corresponding flight time and resources to execute tasks. Consequently, existing studies have presented a huge number of learning methods to estimate tasks' resource requirements and execution time, CL [6,9], Machine Learning (ML) [7,22,23], Incremental Learning (IL) [24], scheduling [14,25–27] and statistical models [28]. Previous works [6,7] focused on multi-dependent task orchestration in autonomous drone-enabled EC systems, while considering the drones' flight time, to avoid the loss of jobs [8]. Specifically, the authors in [6] proposed a multi-output linear regression model based on CL to estimate multi-dependent tasks' resource requirements and execution time, to select the closest drone deployment with matching resource availability and flight time to execute ready tasks at a given time. In [7], the authors proposed an ML-based multi-dependent task dispatching method over a federated autonomous drone-enabled EC platform, using the total estimated value of the multi-dependent tasks' execution time to select a suitable drone. The authors of [9] proposed a distributed training scheme based on CL, where multiple Deep Reinforcement Learning (DRL) agents are deployed on IoT devices to enable joint resource allocation. The work in [22] proposed a method to predict the execution time of a task, by first predicting its run-time parameters, then it uses these run-time parameters to finally predict the execution time of the task. In [23], two novel multi-model ML ensemble systems with the mixture of experts and dynamic selection of experts were presented to predict the execution time of workflows in distributed environments. The work in [24] presented an online incremental approach for the run-time prediction of scientific workflows in cloud computing environments using time-series monitoring data. In [25], the authors proposed a cluster scheduling framework called Gandiva, which exploits intra-job predictability to share GPUs efficiently across multiple jobs, to achieve low latency. The authors of [26] proposed an approach that can accurately predict the performance of a given job. Their main idea is to run a set of instances of the entire job on the samples of the input, and use the data from these training runs to create a performance model. The work in [27] proposed a Deep Learning (DL) job scheduler, which aims to minimize the training time for jobs. This scheduler is based on an online prediction model used to accurately estimate the training speed, as a function of the allocated resources in each job. In [14], the authors proposed a scheduling algorithm for Bulk Synchronous Parallel (BSP) jobs. They showed that their solution is robust against inaccurate estimations. The work in [28] presented a cluster management system called Quasar, using classification techniques to quickly and accurately determine the impact of the scales of resources, types of resources, and interference on performance for each workload and dataset. Then, it uses the clas-

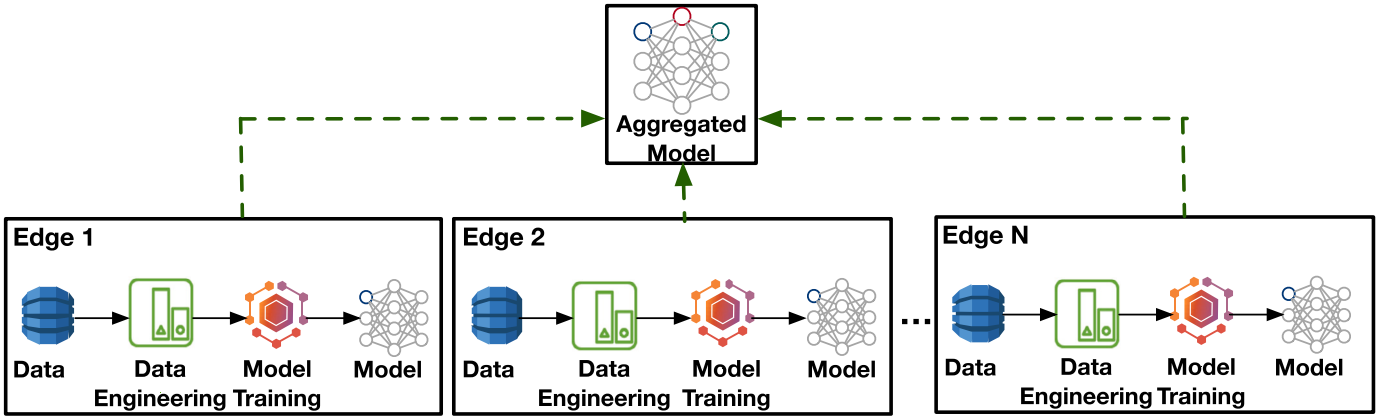


Fig. 3. CL model training and aggregation.

sification results to jointly perform resource allocation and assignment. The authors of [29] proposed a deep DL-based Point of Interest Recommendation (Deep-PR) method for mobile edge networks, where hidden feature components from both local and global sub-spaces are deeply abstracted via representative learning schemes, so that recommendation accuracy can be ensured.

With limited edge resources, it is extremely important to avoid any form of resource wastage, i.e., resource underutilization. Efficiently managing edge resources directly dictates service quality and performance [30]. As a result, task co-location has gained attention both in academia and industry as an optimal solution for improving resource utilization and system throughput in distributed systems. However, effective task co-location is a non-trivial task, as it requires an understanding of the computing resource requirements of co-running tasks to determine how many of them can be co-located. To this end, task co-location mechanism was proposed in [31], by accurately estimating the resource level needed, to effectively determine how many tasks can be co-located on the same host to improve the system throughput, taking into consideration the memory and CPU requirements of co-running tasks. With the aim to maximize resource utilization, the authors of [32] utilized Reinforcement Learning (RL) approach to co-locate interactive services with batched ML workloads. Previous works [17,18] focused on workload co-location in cloud environments rather than edge systems. To further improve edge resource management, a resource management scheme was proposed in [2,5] which unifies distributed edge resources, such that they are holistically managed. Previous work in [2] proposed a dependency-aware task scheduling scheme in such a unified system. Existing EC applications are usually structured with inter-task dependencies, where a task depends on input from other task(s). A huge number of existing studies, i.e., [33–36] have tackled the problem of scheduling such inter-dependent tasks or multi-dependent tasks, and their common goal is to formulate a scheduling decision that minimizes the average completion time of such tasks.

Existing works on CL-based approaches for task offloading and execution in multi-edge deployments do not consider task dependencies and do not unify distributed edge resources, such that they are holistically managed and monitored from a single CP, where multi-tasks can be timely dispatched and co-located without interference or resource contentions. This motivates our research to extend existing schemes by proposing EdgeColla to address these problems. Specifically, we propose a CL-based multi-dependent task resource requirement and execution time estimation method through a linear regression model, and cluster resource status for an integrated edge system through the Control Panel (CP), such that multi-dependent tasks are intelligently dispatched to the *closest* edge cluster having sufficient available resources. We further propose a variant bin-packing optimization approach through gang-scheduling of multi-dependent tasks, which co-schedules and co-locates tasks firmly on available nodes to avoid resource wastage. We finally show that EdgeColla is capable of minimizing the actual completion time of multi-dependent tasks using minimum resources through extensive experiments and comparisons.

3. System model, problem formulation and algorithm framework

The goal of edge CL is to collaboratively learn a model from data $\mathcal{D}_1, \cdots, \mathcal{D}_N$, stored across N distributed clusters, where each dataset $\mathcal{D}_i = \{(\boldsymbol{x}_{i,j}, \boldsymbol{y}_{i,j})\}_{j=1}^{n_i}$ contains d-dimensional tensors of data features $\boldsymbol{x}_{i,j} \in \mathbb{R}^{1\times d}$ and c-dimensional tensor data labels $\boldsymbol{y}_{i,j} \in \mathbb{R}^{1\times c}$. The selection of training data is an important topic in any learning problem. Given the multiple tasks to be deployed, we should select the training data from the historical data that have characteristics as close as possible to those of the current multiple tasks to be deployed. This is critical to ensure the accuracy of the model learned. For example, if the multiple tasks to be deployed are Video Processing (VP) jobs, it is desired to select the training data that include historical VP data if possible to build the model.

Before now, a prevalent method is to integrate datasets in one cluster, i.e., $\mathbb{D} = \bigcup_{i=1}^{N} \mathcal{D}_i$, and use this integrated data $\mathbb{D}$ to train a model Θ_S. Recent CL approaches train models over distributed datasets without the need for datasets aggregation, as shown in Fig. 3. The following steps narrate the process of CL model training and aggregation in EF systems:

1. The `member cluster(s)` D_{edge_i} separately train their models $\Theta_{D_{edge_i}}$ based on their local datasets $\mathcal{D}_i$.
2. Then at time $t > 0$, the `member cluster(s)` send their models, denoted as $\Theta_{D_{edge_i}}^{(t-1)}$, $1 \le i \le N$, to the `host cluster`, where global update $\Theta_G^{(t)}$ is computed by aggregating all the `member cluster` models [37–40]:

$$\Theta_G^{(t)} = \sum_{i=1}^{N} \Theta_{D_{edge_i}}^{(t-1)} \tag{3}$$

3. In response, $\Theta_G^{(t)}$ is distributed to the `member cluster(s)`, where it is used to update $\Theta_{D_{edge_i}}^{(t-1)}$ according to [38]

$$\Theta_{D_{edge_i}}^{(t)} = \Theta_G^{(t)} - \Theta_{D_{edge_i}}^{(t-1)}, \ 1 \le i \le N \tag{4}$$

4. At time $t+1$, updates $\Theta_{D_{edge_i}}^{(t)}$, $1 \le i \le N$, from the `member cluster(s)` are sent back to the `host cluster(s)`, where a global update is computed [38]:

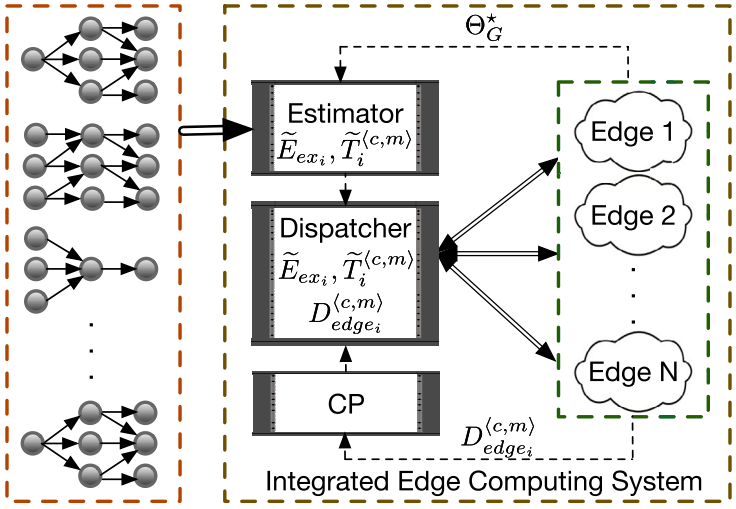


Fig. 4. Overview of EdgeColla.

$$\Theta_G^{(t+1)} = \Theta_G^{(t)} + \sum_{i=1}^{N} \Theta^{(t)}_{D_{edge_i}} \tag{5}$$

For `member cluster(s)` D_{edge_i} with local dataset $\mathcal{D}_i$, the associated ML problem is to solve the following optimization:

$$\Theta^{\star}_{D_{edge_i}} = \arg \min_{\Theta_{D_{edge_i}} \in \mathbb{R}^{d\times c}} \frac{1}{2n_i} \sum_{j=1}^{n_i} \|\boldsymbol{x}_{i,j}\Theta_{D_{edge_i}} - \boldsymbol{y}_{i,j}\|_2^2 + \frac{\lambda}{2}\|\Theta_{D_{edge_i}}\|_F^2 \tag{6}$$

where λ is the regularization parameter and $\|\cdot\|_F$ denotes the Frobenius norm. Optimization (6) is solved using gradient-descent by updating the model iteratively until convergence with formula $\Theta^{(t+1)}_{D_{edge_i}} = \Theta^{t}_{D_{edge_i}} - \eta\left(\frac{1}{n_i}\boldsymbol{g}\left(\Theta^{t}_{D_{edge_i}}\right) + \lambda\Theta^{t}_{D_{edge_i}}\right)$, in which η is the learning rate, $\boldsymbol{g}\left(\Theta^{t}_{D_{edge_i}}\right) = \frac{1}{n_i}\boldsymbol{X}_i^{\mathrm{T}}\left(\boldsymbol{X}_i\Theta^{t}_{D_{edge_i}} - \boldsymbol{Y}_i\right)$ is the gradient of the loss function, $\boldsymbol{X}_i = \left[\boldsymbol{x}_{i,1}^{\mathrm{T}}, \cdots, \boldsymbol{x}_{i,n_i}^{\mathrm{T}}\right]^{\mathrm{T}}$ and $\boldsymbol{Y}_i = \left[\boldsymbol{y}_{i,1}^{\mathrm{T}}, \cdots, \boldsymbol{y}_{i,n}^{\mathrm{T}}\right]^{\mathrm{T}}$ are the feature set and label set, respectively.

As multi-dependent tasks arrive into the system, their features $\boldsymbol{f}_{\mathrm{mt}}(\omega, \epsilon, \gamma)$, where ω is the number of instances, ϵ is the type of tasks, γ is the dependency depth, are fed into global model $\Theta^{\star}_G$ to estimate the values of the resource requirements and execution time according to

$$\boldsymbol{f}_{\mathrm{mt}} \cdot \Theta^{\star}_G = \left[\widetilde{E}_{ex_1}\widetilde{T}_1^{\langle c,m\rangle}\ \widetilde{E}_{ex_2}\widetilde{T}_2^{\langle c,m\rangle} \cdots \widetilde{E}_n\widetilde{T}_n^{\langle c,m\rangle}\right] \tag{7}$$

where $\widetilde{T}_i^{\langle c,m\rangle}$ and $\widetilde{E}_{ex_i}$ are the estimated resource requirements (in terms of CPU and memory $\langle c, m\rangle$) and estimated execution time for task i, respectively. Note that the dispatcher has the estimated values and the update status of all the `member clusters` $\in \mathbb{EF}$ when deciding where to dispatch tasks. We show that with these estimated values, multi-dependent tasks can be intelligently dispatched with the aim of minimizing their actual completion time using minimum resources in an integrated edge system, as shown in Fig. 4. We train our model based on Keras[6] with historical data from previously executed tasks/jobs. Keras is a library that wraps TensorFlow[7] complexity into a simple and user-friendly Application Programming Interface (API).

Multi-dependent tasks resource requirements and execution time estimation values, and `member clusters` available resources denoted as $D_{edge_i}^{\langle c,m\rangle}$ are needed to effectively dispatch multi-dependent tasks $\mathbb{C}$ to the **closest** `member cluster` denoted as $D_{edge_{i\star}}$ with sufficient resources denoted as $D_{edge_i}^{\langle c,m\rangle}$. The **closest** `member cluster` is an edge deployment with the minimum combined upload $\wedge^{\uparrow}_{\mathbb{C}\to C_{edge_i}}$ and download $\wedge^{\downarrow}_{\mathbb{C}\to C_{edge_i}}$ transmission loads:

$$\mathbb{C} \Rightarrow D_{edge_{i\star}} \tag{8}$$

where $D_{edge_{i\star}}$ is the solution of the following optimization:

$$\begin{aligned} \min_{D_{edge_i}\in\mathbb{EF}} \quad & \left(\wedge^{\uparrow}_{\mathbb{C}\Rightarrow D_{edge_i}} + \wedge^{\downarrow}_{\mathbb{C}\Rightarrow D_{edge_i}}\right) \\ \text{s.t.} \quad & D_{edge_i}^{\langle c,m\rangle} \text{ is sufficient for tasks} \end{aligned} \tag{9}$$

If a selected `member cluster` D_{edge_i} is a drone, in addition to meeting the requirement that $D_{edge_i}^{\langle c,m\rangle}$ is sufficient, its flight time f_i should also be sufficient, such that the estimated execution time $\widetilde{E}_{ex} \le f_i$, to avoid job losses [8]. For example, autonomous drone systems such as Drone-in-a-Box[8] have the capabilities to fly intelligently and to estimate their overall fight time f_i^l, including from source location l_i^c to destination location l_i^d to conduct on-demand tasks. Therefore, for drone deployments, constraint $\widetilde{E}_{ex} \le f_i$ should be added to optimization (9).

For task $T \in \mathbb{C}$, its actual starting time and completion time are denoted as E_{st} and E_{cp}, respectively. Thus, its actual execution time is given as

$$E_{ex} = E_{cp} - E_{st} \tag{10}$$

Hence the collective execution time of a (n)-task $\mathbb{C}$ is given as

$$\sum_{i=1}^{n} \frac{E_{ex_i}}{n} \tag{11}$$

Given a node I in each `member cluster`, let $I_p^{\langle c,m\rangle}$ denote the p-th node's resource availability. The estimated resource demands and execution time of k-dependent tasks to be orchestrated, $\sum_{q=1}^{k}\widetilde{T}_q^{\langle c,m\rangle}$ and $\sum_{q=1}^{k}\widetilde{E}_{ex_q}$, the updated resource availability status of each `member cluster` cluster $D_{edge_i}^{\langle c,m\rangle}$, and drones' flight time f_i (for aerial deployments) are needed to make an effective dispatching decision on $\mathbb{C}$ at time t. Our system extends to handle massive requests from multiple

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

users $u \in \mathbb{U}$ [41]. For example, we consider a telecom platform[9] that provides EC services to connected cars and autonomous vehicles. Suppose at t, there are n service requests from $\mathbb{U}$ at location l_i^d, where each user u is offloading $\mathbb{C}$. The collective n request from $\mathbb{U}$ can be dispatched as multi-job $\mathbb{J}$, where $\mathbb{J} = \sum_{i=1}^{n} \mathbb{C}_i$, with the collective resource demand estimation denoted as $\sum_{q=1}^{k} \widetilde{T}_q^{\langle c,m \rangle} = \widetilde{T}^{\langle c,m \rangle\prime}$ and the aggregate execution time estimation as $\sum_{q=1}^{k} \widetilde{E}_{ex_q} = \widetilde{E}_{ex\prime}$. We can dispatch $\mathbb{J}$ to the same `member cluster` by jointly considering the estimated total resource requirements:

$$\sum\nolimits_{J \in \mathbb{J}} \widetilde{T}^{\langle c,m \rangle\prime} = \widetilde{T}_{\text{total}}^{\langle c,m \rangle\prime} \tag{12}$$

where the edge or cluster resource capability is $D_{edge_i}^{\langle c,m \rangle}$ and the total estimated execution time is

$$\sum\nolimits_{J \in \mathbb{J}} \widetilde{E}_{ex\prime} = \widetilde{E}_{ex\prime}^{\text{total}} \tag{13}$$

Let $D_{edge_{i\star}}$ be the **closest** edge having sufficient $D_{edge_{i\star}}^{\langle c,m \rangle}$ and $f_{i\star}$ to accommodate $\widetilde{T}_{\text{total}}^{\langle c,m \rangle\prime}$ and $\widetilde{E}_{ex\prime}^{\text{total}}$, we can dispatch $\mathbb{J}$ to $D_{edge_{i\star}}$:

$$\mathbb{J} \Rightarrow D_{edge_{i\star}} \tag{14}$$

The estimated resource utilization of the cluster or edge for multi-job deployment is given by

$$\widetilde{\rho}_{edge_i}^{\langle c,m \rangle} = \frac{\widetilde{T}_{\text{total}}^{\langle c,m \rangle\prime}}{D_{edge_i}^{\langle c,m \rangle}} \tag{15}$$

For a `member cluster` D_{edge_i}, let the aggregate execution time of multi-job $\mathbb{J}$ be

$$\sum\nolimits_{J \in \mathbb{J}} \sum\nolimits_{q=1}^{k} \frac{E_{ex_q}}{k} = \sum\nolimits_{J \in \mathbb{J}} E_{ex\prime} = E_{ex\prime}^{\text{total}} \tag{16}$$

and the total resources actually assigned for multi-job $\mathbb{J}$ be $D_{edge_iU}^{\langle c,m \rangle}$.

Proof. In IoT, edge, and cloud computing systems, timestamp[10] values are assigned to various events or tasks based on when they occur, i.e., to indicate a task's starting time (E_{st}), completion time (E_{cp}), etc. Events are timestamped based on when they occurred for a range of use cases. For example, it is used to deduce a task's execution time, i.e., E_{ex} of a task as expressed in Equation (10). Therefore, for a set of multi-job tasks, the aggregate execution time is expressed in Equation (16). However, the actual execution time of tasks is unknown at this stage, hence we replace it with the estimation values as expressed in Equation (13). Specifically, the estimated execution time value is essential in selecting a UAV or drone-based EC deployment with sufficient flight time to avoid job losses [6–8,42].

Under the condition that estimated total resource demand $\widetilde{T}_{\text{total}}^{\langle c,m \rangle\prime}$ is accurate, i.e., $\widetilde{T}_{\text{total}}^{\langle c,m \rangle\prime} \approx D_{edge_iU}^{\langle c,m \rangle}$, then the actually total resources $D_{edge_iU}^{\langle c,m \rangle}$ assigned for $\mathbb{J}$ will not exceed $D_{edge_i}^{\langle c,m \rangle}$. Similarly, under the condition that estimated total execution time $\widetilde{E}_{ex\prime}^{\text{total}}$ is accurate, i.e., $\widetilde{E}_{ex\prime}^{\text{total}} \approx E_{ex\prime}^{\text{total}}$, drone D_{edge_i} will have sufficient flight time f_i for multi-job execution.

Our CL-based approach has significant advantages over non-learning counterparts. By accurately estimating the resource requirements and execution time of multiple tasks/jobs, our scheme can intelligently co-locate multi-dependent tasks in the closest edge having sufficient resources, such that these dependent tasks can communicate and execute faster, ultimately minimizing the response time and improving resource utilization. The accuracy of the estimated resource requirements and execution time can be ensured by constructing multiple training datasets for different multi-task/multi-job classes from historical data to learn multiple models, one for a single multi-task/multi-job class. Given multiple tasks/jobs to be deployed, the model that is the most similar to them is employed to estimate the resource requirements and execution time. Since the estimated total resource demand $\widetilde{T}_{\text{total}}^{\langle c,m \rangle\prime}$ and execution time $\widetilde{E}_{ex\prime}^{\text{total}}$ are accurate estimates of the actual total resource to be allocated $D_{edge_iU}^{\langle c,m \rangle}$ and actual execution time $E_{ex\prime}^{\text{total}}$, it is unlikely that the selected drone edge D_{edge_i} will not have sufficient resources. In other words, it is very unlikely that

$$E_{ex\prime}^{\text{total}} > f_i \text{ and/or } D_{edge_iU}^{\langle c,m \rangle} > D_{edge_i}^{\langle c,m \rangle} \tag{17}$$

which would lead to job losses. By contrast, standard non-learning schemes have no means to intelligently choose appropriate edge deployments for ensuring that the selected edge D_{edge_i} will have sufficient resources, and the probability of (17) occurring can be much higher than our EdgeColla approach. There also exist simple and effective measures to guard against estimation errors. It is obvious that job losses may only occur in underestimation scenarios. Instead of using the estimates of the resource demand and execution time for selecting edge deployments, we can add the two standard deviations of the estimation to the corresponding estimates and use these 'modified' or 'overly' estimated values to select edge deployments. This will reduce the probability of (17) occurring to almost zero. It is straightforward to provide both the estimate and estimation standard deviation by dividing the training data into multiple subsets and running the estimation procedure multiple times. □

3.1. Problem formulation

The notations adopted are listed in Table 2. EdgeColla includes an intelligent multi-dependent task dispatching method, which co-locates tasks firmly on available nodes to avoid resource wastage in any `member cluster` $\in \mathbb{EF}$, while considering task dependencies. Our objectives are to maximize the actual cluster resource utilization and to minimize the overall execution time of multi-dependent tasks, subject to certain constraints.

3.1.1. Constraints

First, the collective resource demand estimation of $\mathbb{J}$ at any given time t cannot exceed the available resources of a selected `member cluster` $\in \mathbb{EF}$. Since the actual total resources that need to be assigned to multi-job $D_{edge_iU}^{\langle c,m \rangle}$ is unknown at the scheduling stage, we use the estimated total resource demand $\widetilde{T}_{\text{total}}^{\langle c,m \rangle\prime}$ to replace it:

$$\widetilde{T}_{\text{total}}^{\langle c,m \rangle\prime} \leq D_{edge_i}^{\langle c,m \rangle}, \quad \forall D_{edge_i} \in \mathbb{EF} \tag{18}$$

Second, the aggregate execution time of $\mathbb{J}$ at any given time t cannot exceed the flight time of any selected drone. Since the actual execution time $E_{ex\prime}^{\text{total}}$ is unavailable at the scheduling stage, we replace it with $\widetilde{E}_{ex\prime}^{\text{total}}$:

$$\widetilde{E}_{ex\prime}^{\text{total}} \leq f_i, \quad \forall f_i \in \mathbb{EF} \tag{19}$$

Third, unused or inactive nodes $I_i \in D_{edge_i}$ in a selected cluster would be shut down. All the nodes can be expressed in one of these two states: *Active* and *Inactive*. An *Active* node is a node that is running and currently considered for allocation or has at least 1 job being started, executed or completed. An *Inactive* node is a node that is not running and currently considered for allocation and not having at least 1 job that is being started, executed or completed. These two states can be expressed as follows:

$$\forall c, m \ \beta\left(I_i\right) = \begin{cases} 1, & \textit{Active} \text{ if } J_i \in [E_s, E_c, E_{ex}] \\ 0, & \textit{Inactive} \text{ if } J_i \notin [E_s, E_c, E_{ex}] \end{cases} \tag{20}$$

[9] https://stellar.tc/.

[10] https://learn.microsoft.com/en-us/stream-analytics-query/timestamp-by-azure-stream-analytics.

Table 2
Common notations.

Notation	Description
$\mathbb{EF}$	Integrated edge deployments
T	Individual applications or tasks
$\langle c, m\rangle$	CPU and memory resources
$\mathbb{C}$	A set of containerized applications
$\widetilde{T}^{\langle c,m\rangle}$	Task resource requirement estimation
$\widetilde{T}^{\langle c,m\rangle\prime}_{\text{total}}$	Estimated total resource requirements for jobs
$T^{\langle c,m\rangle\prime}_{\text{total}}$	Actual total resources consumed for jobs
$T^{\langle c\rangle\prime}_{\text{total}}, T^{\langle m\rangle\prime}_{\text{total}}$	Actual CPU, memory resources consumed for jobs
D_{edge_i}	Individual edge deployment or `member cluster`
$D_{edge_{i\star}}$	Closest edge deployment with required resources
I_i	A 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 edge/cluster
$D_{edge_iU}^{\langle c,m\rangle}$	Actual resources used or assigned for jobs
$D_{edge_iU}^{\langle c\rangle}, D_{edge_iU}^{\langle m\rangle}$	Actual CPU, memory assigned for jobs
$D_{edge_iARU}^{\langle c,m\rangle}$	Actual resource usage for executing jobs
$\widetilde{\rho}_{edge_i}^{\langle c,m\rangle}$	Estimated resource utilization of jobs
$\rho_{edge_i}^{\langle c\rangle}, \rho_{edge_i}^{\langle m\rangle}$	Actual cluster CPU, memory resource utilization
E_{st}, E_{cp}	Application or task starting time, completion time
E_{ex}	Application or task execution time
$E^{\text{total}}_{ex\prime}$	Actual total execution time for jobs
$\widetilde{E}_{ex}$	Application or task execution time estimation
$\widetilde{E}^{\text{total}}_{ex\prime}$	Estimated total execution time for jobs
l_i^c, l_i^d	Drone's current location and destination location
f_i	Drone's flight time
$\wedge^{\uparrow}_{\mathbb{C}\Rightarrow D_{edge_i}}$	Upload transmission
$\wedge^{\downarrow}_{\mathbb{C}\Rightarrow D_{edge_i}}$	Download transmission
ω_J	The number of instances of a job
ϵ_J	The type of a job
γ_J	Dependency depth of a job
$\boldsymbol{f}_{\text{mt}}$	Set of multi-task runtime parameters
Θ	Linear regression model
J, $\mathbb{J}$	A Job, A set of Jobs
u, $\mathbb{U}$	A User, A set of Users

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

3.1.2. Optimization formulation

As the actual resource utilization of a cluster/edge is unknown, we maximize the estimated resource utilization:

$$\textbf{Maximize} \quad \widetilde{\rho}_{edge_i}^{\langle c,m\rangle} \tag{21}$$

$$\textit{subject to} \quad \mathbb{J} \Rightarrow D_{edge_{i\star}},\ \exists \tag{22}$$

$$\widetilde{E}^{\text{total}}_{ex\prime} \le f_i,\ \forall f_i \in \mathbb{EF},\ \exists \tag{23}$$

$$\widetilde{T}^{\langle c,m\rangle\prime}_{\text{total}} \le D_{edge_i}^{\langle c,m\rangle},\ \forall D_{edge_i} \in \mathbb{EF},\ \exists \tag{24}$$

$$\beta(I_i) \in \{0,1\},\ \exists \tag{25}$$

Provided that the estimated resource utilization $\widetilde{\rho}_{edge_i}^{\langle c,m\rangle}$ is accurate, little optimality will be lost.

The constraints (22) to (24) indicate dispatching multi-job $\mathbb{J}$ to the closest edge having sufficient resources and flight time. More specifically, (22) is $\mathbb{J}$, guaranteeing that $\mathbb{J}$ is dispatched to a cluster, such that dependent tasks within each $J \in \mathbb{J}$ can communicate and execute faster. Constraint (23) guarantees that $\widetilde{E}^{\text{total}}_{ex\prime}$ of $\mathbb{J}$ should not exceed f_i of any selected drone deployment and constraint (24) guarantees that $\widetilde{T}^{\langle c,m\rangle\prime}_{\text{total}}$ of $\mathbb{J}$ should not exceed $D_{edge_i}^{\langle c,m\rangle}$ of any selected `member cluster` $\in \mathbb{EF}$. We shall discuss the details of our multi-job dispatching principle in Subsection 3.2 and Algorithm 2. Condition (25) guarantees that active nodes ($\beta(I_i)=1$) should be used for execution, and inactive nodes ($\beta(I_i)=0$) should be shut down. Hence, our aim is to minimize the number of active nodes used for execution by co-locating jobs tightly on each node to maximize resource utilization. We shall discuss the details of our co-location strategy in Subsection 3.2 and Algorithm 3.

Then again, $\widetilde{E}^{\text{total}}_{ex\prime}$ of $\mathbb{J}$ can be minimized depending on dispatching:

$$\textbf{Minimize} \quad \widetilde{E}^{\text{total}}_{ex\prime} \tag{26}$$

$$\textit{subject to} \quad \mathbb{J} \Rightarrow D_{edge_{i\star}},\ \exists \tag{27}$$

Note that the actual overall execution time $E^{\text{total}}_{ex\prime}$ is unknown at this stage and we use the estimated overall execution $\widetilde{E}^{\text{total}}_{ex\prime}$ to replace it in the optimization. Again, provided that $\widetilde{E}^{\text{total}}_{ex\prime}$ is accurate, little optimality will be lost. Constraint (27) guarantees that $\mathbb{J}$ is dispatched to a cluster, such that dependent tasks within each $J \in \mathbb{J}$ can communicate and execute faster. The details of our multi-job dispatching principle are given in Subsection 3.2 and in Algorithm 2.

Algorithm 1 Linear Regression Estimation.

Input: $\mathbb{J}$ arrives at time t from l_i^d; $\boldsymbol{f}_{\text{mt}}$ is fed into $\Theta^{\star}_G$
Output: $\sum_{J\in\mathbb{J}} \widetilde{T}_J^{\langle c,m\rangle\prime}$ and $\sum_{J\in\mathbb{J}} \widetilde{E}_{ex_J\prime}$

```
for J ∈ 𝕁 do
    Type of job J = ϵ_J
    Number of instances of job J = ω_J
    Dependency depth of job J = γ_J
    for T_i ∈ J do
        (f_mt)_{T_i} · Θ*_G = [T̃^{⟨c,m⟩}_{T_i}  Ẽ_{ex_{T_i}}]
    end for
    T̃_J^{⟨c,m⟩′} = T̃_J^{⟨c,m⟩′} + T̃^{⟨c,m⟩}_{T_i}
    Ẽ_{ex_J}′ = Ẽ_{ex_J}′ + Ẽ_{ex_{T_i}}
end for
```

Algorithm 2 Multi-Job Dispatching.

Input: $\mathbb{J}$ arrives at time t within l_i^d; $D_{edge_i} \in \mathbb{EF}$; $\sum_{J\in\mathbb{J}} \widetilde{T}_J^{\langle c,m\rangle\prime}$; $\sum_{J\in\mathbb{J}} \widetilde{E}_{ex_J\prime}$
Output: Dispatch $\mathbb{J}$ to $D_{edge_{i\star}}$ with matching $D_{edge_i}^{\langle c,m\rangle}$ and f_i for any selected drone, such that $\mathbb{J} \Rightarrow D_{edge_{i\star}}$

```
for D_edge_i ∈ 𝔼𝔽 do
    if Σ_{J∈𝕁} T̃_J^{⟨c,m⟩′} ≤ D_{edge_i}^{⟨c,m⟩} and Σ_{J∈𝕁} Ẽ_{ex_J}′ ≤ f_i then
        if D_edge_i = arg min_{D_edge_j ∈ 𝔼𝔻𝔾𝔼} (Q_{𝕁⇒D_edge_j}) then
            𝕁 ⇒ D_edge_i = D_edge_{i⋆}
        else
            Dispatch 𝕁 to next D_edge_{i⋆}
        end if
    end if
end for
if 𝕁 cannot be composed as a whole then
    for D_edge_i ∈ 𝔼𝔽 do
        for J ∈ 𝕌 do
            if T̃_J^{⟨c,m⟩′} ≤ D_{edge_i}^{⟨c,m⟩} and Ẽ_{ex_J}′ ≤ f_i then
                if D_edge_i = arg min_{D_edge_j ∈ 𝔼𝔽} (Q_{J⇒D_edge_j}) then
                    J ⇒ D_edge_i = D_edge_{i⋆}
                else
                    Dispatch J to next D_edge_{i⋆}
                end if
            end if
        end for
    end for
end if
```

3.2. EdgeColla algorithm framework

Our EdgeColla approach consists of estimation, dispatching, and co-location. These 3 components aim at providing optimal performance for multi-task execution in an integrated edge system, such that optimization (21) and optimization (26) are achieved. The values of the estimations are required by the dispatcher, as well as the update state of the clusters for effective multi-job dispatching to the closest

Algorithm 3 Multi-job Co-location.

Input: $\mathbb{J}$ dispatched to closest `member cluster` $D_{edge_\star}$, $\sum_{J\in\mathbb{J}} \widetilde{T}_J^{\langle c,m\rangle\prime}$, resource availability $I_i^{\langle c,m\rangle}$ of all nodes $I_i \in D_{edge_\star}$
Output: $\mathbb{J}$ is co-located to **Minimize** $\sum_{I_i\in D_{edge_\star}} I_i$

1: **for** $I_i \in D_{edge_\star}$ **do**
2: **if** $\beta(I_i) = 1$ **then**
3: $I_i^{\langle c,m\rangle} = \langle c, m\rangle$, i.e., initial resource available
4: **for** $J \in \mathbb{J}$ **do**
5: **if** $\Gamma[J, I_i] = 0$ and $\widetilde{T}_J^{\langle c,m\rangle\prime} \le I_i^{\langle c,m\rangle}$ **then**
6: $J \Rightarrow I_i$
7: $\Gamma[J, I_i] = 1$
8: $I_i^{\langle c,m\rangle} = I_i^{\langle c,m\rangle} - \widetilde{T}_J^{\langle c,m\rangle\prime}$
9: **end if**
10: **if** $I_i^{\langle c,m\rangle}$ close to zero **then**
11: **break**
12: **end if**
13: **end for**
14: **end if**
15: **end for**

`member cluster` $D_{edge_\star}$ with the minimum combined upload and download transmission loads:

$$Q_{\mathbb{J}\Rightarrow D_{edge_i}} = \wedge^{\uparrow}_{\mathbb{J}\Rightarrow D_{edge_i}} + \wedge^{\downarrow}_{\mathbb{J}\Rightarrow D_{edge_i}} \tag{28}$$

Our co-location approach involves co-locating these tasks firmly on available resources. We detail the procedures of the 3 components of EdgeColla as follows.

3.2.1. Resource and execution time estimation

As $\mathbb{J}$ arrives into the system, their collective resource requirement $\widetilde{T}^{\langle c,m\rangle\prime}_{\text{total}}$ and execution time $\widetilde{E}^{\text{total}}_{ex\prime}$ are estimated. The CL process in Equations (3) ~ (5) generates a global model. A set of runtime parameters $\boldsymbol{f}_{\text{mt}}(\omega, \epsilon, \gamma)$, where ω is the number of instances, ϵ is the type of tasks, and γ is the dependency depth, is fed into the global model $\Theta^{\star}_{G}$ to produce estimation values. Once the estimation values are produced, they are used in the dispatching stage.

3.2.2. Dispatching

Our policy is to dispatch $\mathbb{J}$ to the closest `member cluster` $D_{edge_{i\star}}$ with matching or sufficient resources $D^{\langle c,m\rangle}_{edge_{i\star}}$ and flight time $f_{i\star}$, such that $\widetilde{T}^{\langle c,m\rangle\prime}_{\text{total}} \le D^{\langle c,m\rangle}_{edge_{i\star}}$ and $\widetilde{E}^{\text{total}}_{ex\prime} \le f_{i\star}$. The *closest* heuristic given in Equation (28) is to further minimize the overall response time of $\mathbb{J}$. *Closest* or *Nearest* is a popular task offloading heuristic in distributed systems, since IoT and other end devices often need to communicate only with the closest or nearest clusters and cloud servers. Existing studies, e.g., [6,7,18,43], adopted the *closest* principle as the task offloading policy. Holistic dispatching of $\mathbb{J}$ treats each $J \in \mathbb{J}$ as a high-priority job. Algorithm 2 describes the dispatching procedure.

With the collective estimated values of $\mathbb{J}$ and all `member clusters` $\in \mathbb{EF}$, and available resources $D^{\langle c,m\rangle}_{edge_i}$ are obtained through the CP. The dispatcher selects the closest `member cluster` $D_{edge_{i\star}}$ having sufficient resources (line 3). It dispatches $\mathbb{J}$ to $D_{edge_{i\star}}$ (line 4). If $\mathbb{J}$ cannot be dispatched to $D_{edge_{i\star}}$, then $\mathbb{J}$ is dispatched to the next $D_{edge_{i\star}}$ (line 6). If at any time t, the collective estimated values of $\mathbb{J}$ are greater than any `member cluster` $\in \mathbb{EF}$, i.e., $\sum_{J\in\mathbb{J}} \widetilde{T}^{\langle c,m\rangle\prime}_{J} > D^{\langle c,m\rangle}_{edge_i}$ and/or $\sum_{J\in\mathbb{J}} \widetilde{E}_{ex_J\prime} > f_i, \forall D_{edge_i} \in \mathbb{EDGE}$, then $\mathbb{J}$ cannot be composed as a whole. The dispatcher can allow fractionally dispatching each $J \in \mathbb{U}$ to the closest `member cluster` (line 10 ~ 22). Note that fractionally dispatching each $J \in \mathbb{U}$ to the closest `member cluster` would still allow inter-dependent tasks within each $J \in u$ to execute faster.

Recall that a multi-Job $\mathbb{J}$ is composed of jobs from multiple users. The reason behind this multi-Job $\mathbb{J}$ formation is that it improves the efficiency of dispatching and scheduling of ready jobs at a time, given that their collective resource demand and execution time estimation does not exceed the resource capacity of the closest `member cluster`. Therefore, we can define the size of a multi-Job $\mathbb{J}$ in terms of the total execution time and resource requirement estimation using constraints (23) and (24).

3.2.3. Co-location

At `member cluster` $D_{edge_\star}$, our co-location algorithm uses $I_i^{\langle c,m\rangle}$ and $\sum_{J\in\mathbb{J}} \widetilde{T}^{\langle c,m\rangle\prime}$ to provide efficient co-location, such that fewer nodes are used for execution in the EF system. Specifically, the gang scheduling approach is adopted alongside our bin-packing optimization to co-schedule and co-locate $\mathbb{J}$ at a time. Bin-packing is one of the most popular packing problems. The goal is to minimize the number of nodes used, as given in optimization (29). Unlike other approaches, such as First-Fit-Bin-Packing (FFBP) [44], it requires the next J_i to be placed on an active node, otherwise; it is placed on a new node. Our approach scans all $J \in \mathbb{J}$ and maps J_i to active nodes for full utilization. All $J \in \mathbb{J}$ are co-located firmly on active nodes, so that resource wastage is avoided and fewer nodes are used to execute all jobs concurrently. Hence our co-location strategy is to find the solution to the following problem:

$$\textbf{Minimize} \sum_{I_i\in D_{edge_\star}} I_i \tag{29}$$

$$\textit{subject to } \mathbb{J} \Rightarrow D_{edge_\star}, \exists \tag{30}$$

$$\sum_{J\in\mathbb{J}} \Gamma[J, I_i] \cdot \widetilde{T}_J^{\langle c,m\rangle\prime} \le I_i^{\langle c,m\rangle}, \quad \forall c, m \tag{31}$$

where

$$\Gamma[J, I_i] = \begin{cases} 1, & \text{if } J \Rightarrow I_i \\ 0, & \text{otherwise} \end{cases} \tag{32}$$

Constraint (30) is the multi-job $\mathbb{J}$ deployment constraint of guaranteeing that $\mathbb{J}$ is dispatched to a cluster such that dependent tasks within each $J \in \mathbb{J}$ can communicate and execute faster. As we have stated previously that $\mathbb{J}$ cannot be dispatched as a whole to a cluster, the dispatcher can allow fractionally dispatching each $J \in \mathbb{J}$ to closest `member cluster`. Constraint (31) indicates that the total estimated resource requirements of co-located jobs $\sum_{i=1}^{N} \widetilde{T}_i^{\langle c,m\rangle\prime}$ cannot exceed $I_i^{\langle c,m\rangle}$, the node's available resources. Constraint (32) means that if job J_i is placed on node I_i, then $\Gamma[J_i, I_i] = 1$; otherwise, $\Gamma[J_i, I_i] = 0$. This is to guarantee that each $J \in \mathbb{J}$ is placed in exactly one node. To solve this multi-job packing problem, we have adopted the Solving Constraint Integer Programs (SCIP) solver, which is currently one of the fastest Mathematical Programming (MP) solvers for this problem.

Algorithm 3 co-locates multi-dependent tasks firmly on nodes, such that for any given job, resource wastage is avoided and fewer nodes are used for execution. It takes the multi-task/job resource demand and nodes available resources as input, then scans all $J \in \mathbb{J}$ and maps them to active nodes for full utilization.

Note that in existing EC systems, the dispatcher and scheduler need to understand the characteristics of both the applications, e.g., dependencies, resource requirements, and edge resources in terms of availability. Specifically, the scheduler should understand the resource requirements of each sub-application, resource availability of each node, node availability, etc., and assumes the responsibility for executing the applications on the nodes so that the desired objectives are achieved. Therefore, we utilize an ML linear regression model to provide the estimated resource requirements and execution time of ready tasks, so as to dispatch these tasks to the closest `member cluster` having sufficient resources and to quickly execute them. With this in mind, we aim to avoid the execution delays due to the insufficient resources of the deployed `member cluster` or due to tasks waiting in queues. However, if the estimation values are not accurate, and **depending on the level of inaccuracy**, the dispatcher and scheduler in Algorithms 2 and 3, respectively, would perform poorly, i.e., the overall execution of tasks would be delayed due to the insufficient resources or long queues. For

Table 3
Integrated-edge resource capacities.

Edge Deployments	Edge Devices and Total Weight for Aerial Deployments	CPU Capacity (Cores)	Mem Capacity	Flight Time
Aerial Edge 1	Lenovo SE350 + HIVECELL + Xavier NX + Dell 3000 ≈13 kg	30	274 GiB	Sufficient
Aerial Edge 2	Snowcone + Huawei AR502H + INTELLIEDGE G700(x2) ≈14 kg	22	38 GiB	Sufficient
Aerial Edge 3	{Dell 3000 + Dell 5000 + aiSage + dynaEdge}(x4) ≈20 kg	48	112 GiB	Sufficient
Ground Edge 1	Stack Edge + DELL EMC + Snowball + ThinkSystem	132	3536 GiB	NA
Ground Edge 2	AWS Snowball (x3) + Dell EMC VxRail(x2)	256	6640 GiB	NA
Ground Edge 3	{HPE Edgeline + IBM Power Systems}(x6)	288	24 TiB	NA

this reason, we first investigate the accuracy of our trained linear regression model for estimating the resource requirements and execution time of multi-dependent tasks, using the NAEE method. We will discuss NAEE in Section 4.3.1 and Equation (36). This serves as the estimation accuracy measure for the trained linear regression model.

3.2.4. Connection with optimization objectives

As stated previously, our objectives are to maximize the actual edge cluster resource utilization and to minimize the overall execution time of task-dependent jobs. Algorithms 2 and 3 together achieve these objectives. By dispatching task-dependent jobs to the closest edge having sufficient resources and flight time (for drones), Algorithm 2 ensures that the actual resources assigned to the execution of jobs $D^{\langle c,m\rangle}_{edge_i U}$ are sufficient and the dependent tasks can be executed faster, ultimately leading to a smaller aggregate execution time $E^{\text{total}}_{ex\prime}$ and better cluster resource utilization. By intelligently packing dependent tasks tightly on nodes, Algorithm 3 is capable of fully utilizing available resources at edge clusters, ultimately leading to the actual resource assigned to the execution of jobs $D^{\langle c,m\rangle}_{edge_i U}$ as small as possible while guaranteeing it is sufficient for the multi-dependent jobs.

More specifically, the Actual Resource Usage (ARU) of the cluster for multi-job deployment $\mathbb{J}$ is given by

$$D^{\langle c,m\rangle}_{edge_i ARU} = \frac{D^{\langle c,m\rangle}_{edge_i U}}{D^{\langle c,m\rangle}_{edge_i}} \tag{33}$$

It can be seen that solving optimization (29) is directly linked to minimizing ARU (33). Let the actual CPU resource and the actual memory resource assigned for $\mathbb{J}$ be $D^{\langle c\rangle}_{edge_i U}$ and $D^{\langle m\rangle}_{edge_i U}$, respectively. Further denote the actual CPU consumed and the actual memory consumed in executing $\mathbb{J}$ as $\sum_{J\in\mathbb{J}} T^{\langle c\rangle\prime}$ and $\sum_{J\in\mathbb{J}} T^{\langle m\rangle\prime}$, respectively. Then the actual CPU utilization $\rho^{\langle c\rangle}_{edge_i}$ and the actual memory utilization $\rho^{\langle m\rangle}_{edge_i}$ are defined respectively by

$$\rho^{\langle c\rangle}_{edge_i} = \frac{\sum_{J\in\mathbb{J}} T^{\langle c\rangle\prime}}{D^{\langle c\rangle}_{edge_i U}} \tag{34}$$

$$\rho^{\langle m\rangle}_{edge_i} = \frac{\sum_{J\in\mathbb{J}} T^{\langle m\rangle\prime}}{D^{\langle m\rangle}_{edge_i U}} \tag{35}$$

Algorithms 2 and 3 are directly connected with minimizing $E^{\text{total}}_{ex\prime}$ as well as maximizing $\rho^{\langle c\rangle}_{edge_i}$ and $\rho^{\langle m\rangle}_{edge_i}$.

Proof. The proposed *EdgeColla* runs all tasks in containers, which helps to prevent lock-in situations among `host` and `member` clusters. A container packages up all the dependencies of an application, such that it runs quickly and reliably from one computing environment to another.[11] Hence, the proposed *Co-location* strategy can co-locate multiple containers on the same node **without resource contention** among the co-located containers. In this case, each container runs within its allocated resources, i.e., $T^{\langle c,m\rangle}$ in terms of CPU and memory resources. We aim to minimize the ARU within each cluster as expressed in Equation (33) by employing our *Dispatching* and *Co-location* strategies in Algorithms 2 and 3, respectively. Therefore, the actual resource utilization of a cluster is expressed in Equations (34) and (35) for CPU and memory, respectively. □

4. Performance evaluation

In this section, we describe our experimental setup including cluster resource configuration, the Alibaba cluster dataset, and the comparison baselines. We perform extensive experiments to compare EdgeColla against some existing schemes. We will also compare EdgeColla against existing individual cluster schemes. We show that EdgeColla can achieve the minimized actual execution time of multi-dependent tasks, high resource utilization, load balancing, use fewer cluster resources and avoid job losses in an integrated edge system.

4.1. Experimental setup

Cluster Resources: Our 𝔼𝔽 setup consists of 3 aerial or drone clusters and 3 ground or on-premise clusters, as summarized in Table 3. The aerial clusters consist of various portable edge devices with combined weights of up to 20 kg. For example, autonomous drones such as the Bell ATP70[12] have a payload capability of up to 31 kg and a flight time og up to 45 minutes. Therefore, we assume that the selected drones have sufficient flight time to execute ready multi-dependent job.

Multi-dependent Tasks: We employ the v-2018 version of the Alibaba cluster dataset, which records the activities of about 4000 machines in a period of 8 days. The entire dataset contains more than 14 million tasks with more than 12 million dependencies and more than 4 million jobs. Among these, we have deployed 209 jobs with a total of 931 tasks (including dependencies) for our experiments. The number of tasks within each job ranges from (26, 344], while the task dependency depth among the jobs ranges from (1, 16]. The multi-task dependencies in the dataset are valuable for our investigation. Researchers have thoroughly investigated the v-2018 version of Alibaba cluster dataset and used it for various task scheduling problems [45–47].

Comparison Baselines: We compare the scheduling approach of EdgeColla with the following 3 existing schemes and the random approach, fixing their dispatching policies to that of EdgeColla:

1. An approach that does not consider tasks' dependencies but schedules 50% of any given multi-dependent task by mainly focusing on task co-location. We refer to this approach as No Dependency and Full Packing (NDFP), which is similar to the approach in [19].
2. An approach that schedules up to 15% of any given multi-dependent task at a time but does not consider task co-location. We refer to this approach as Partial Dependency and No Packing (PDNP), which is similar to the approach in [20].
3. An approach that schedules up to 40% of any given multi-dependent tasks with task co-location. We consider this approach as a Partial Dependency and Full Packing (PDFP), which is similar to the approach in [21].

[11] https://www.docker.com/resources/what-container/.

[12] www.bellflight.com/products/bell-apt.

Table 4
Multi-job executions in integrated edge deployments, where the actual resource consumed for multi-job execution $T_{\text{total}}^{\langle c,m\rangle\prime}$ and actual execution time $E_{ex\prime}^{\text{total}}$ are taken from the original Alibaba dataset, while the estimated resource demand $\widetilde{T}_{\text{total}}^{\langle c,m\rangle\prime}$ and execution time $\widetilde{E}_{ex\prime}^{\text{total}}$ are calculated by Algorithm 1.

D_{edge_i}	$\mathbb{J}$	$\mathbb{C}$	$\widetilde{T}_{\text{total}}^{\langle c,m\rangle\prime}$	$T_{\text{total}}^{\langle c,m\rangle\prime}$	NAEE	$\widetilde{E}_{ex\prime}^{\text{total}}$ (s)	$E_{ex\prime}^{\text{total}}$ (s)	NAEE
`AerialEdge 1`	5	35	$\langle 23, 10.81\rangle$	$\langle 25.5, 10.48\rangle$	$\langle 0.10, 0.03\rangle$	1134.18	735	0.54
`AerialEdge 2`	13	26	$\langle 15.1, 7.48\rangle$	$\langle 17.7, 6.57\rangle$	$\langle 0.15, 0.14\rangle$	158.16	957	0.83
`AerialEdge 3`	8	49	$\langle 31.52, 15.38\rangle$	$\langle 36.5, 15.5\rangle$	$\langle 0.14, 0.01\rangle$	1456.1	994	0.46
`GroundEdge 1`	34	159	$\langle 101.17, 49.13\rangle$	$\langle 116.2, 48.05\rangle$	$\langle 0.13, 0.02\rangle$	4204.51	3680	0.14
`GroundEdge 2`	68	318	$\langle 202.35, 98.25\rangle$	$\langle 232.4, 96.1\rangle$	$\langle 0.13, 0.02\rangle$	8409.03	7360	0.14
`GroundEdge 3`	81	344	$\langle 217.45, 105.73\rangle$	$\langle 250.1, 102.67\rangle$	$\langle 0.13, 0.03\rangle$	8567.2	8317	0.03

4. The Random approach schedules a single task individually and assumes a node can only execute a task at a time.

4.2. Device mobility and communication

In a broader EC scenario, such as the EC-enabled Internet of Vehicles (IoV), edge clusters are deployed on Road Side Units (RSU) and directly in vehicles to facilitate faster application executions [48,49]. The RSUs and in-vehicle edge deployments can be added as new members to existing integrated edge systems. Therefore, vehicles without sufficient resources can offload their tasks to the closest RSU or other available edge deployments, and after the execution of $J \in \mathbb{J}$, the final result is immediately transmitted back to the vehicle. However, the fundamental challenge is how a moving vehicle, whose initial location coordinate are $\{x, y\}$, can receive its final execution results at any current location $\{x', y'\}$. Moreover, given the current location of the vehicle, more than one routing path may exist from the RSU to the vehicle. Therefore, the routing path with the best transmission performance can be determined as the optimal one for the final result transmission. To this end, we propose an Integrated Edge-assisted Routing (IER) mechanism [48], whose goal is to find the fastest route to efficiently forward execution results to the vehicle at its current location. Specifically, our IER leverages the cooperation among participating EC deployments, i.e., `host` and `members` to quickly forward the execution results to the target vehicle.

4.3. Deployment results and performance comparison

Our investigation focuses on CPU and memory usage/utilization, task deployment, scheduling, and execution time. The results obtained by EdgeColla, PDFP, NDFP, PDNP, and Random are compared.

4.3.1. Resource and execution time estimation accuracy

As detailed in the previous section, to implement the proposed CL-based intelligent multi-task dispatching and co-location strategy, we train a linear regression model on a training dataset. In the real-time application experiments, the trained model is used to estimate the resource requirements and execution time (Algorithm 1). The estimated resource requirements and execution time[13] are then employed to aid our intelligent dispatching and co-location strategy (Algorithms 2 and 3). Clearly, the accuracy of Algorithm 1 impacts the achievable performance of our EdgeColla. Therefore, we first investigate the accuracy of our trained linear regression model.

The multi-job execution information across federated edge deployments, obtained according to the Alibaba dataset, are listed in Table 4, where the estimated resource demand $\widetilde{T}_{\text{total}}^{\langle c,m\rangle\prime}$ and the estimated execution time $\widetilde{E}_{ex\prime}^{\text{total}}$ are calculated using Algorithm 1, while the actual resources consumed for multi-job execution $T_{\text{total}}^{\langle c,m\rangle\prime}$ and the actual execution time $E_{ex\prime}^{\text{total}}$ are taken from the original data. The Normalized Absolute Estimate Error (NAEE) is defined as

$$\text{NAEE} = \frac{\left|\text{estimated value} - \text{actual value}\right|}{\text{actual value}} \tag{36}$$

NAEEs are listed in Table 4 for the resource consumed and execution time, and serve as the estimation accuracy measure for the trained multi-output linear regression model. The average NAEE across 6 deployments is 0.13 for CPU resources, 0.04 for memory resources, and 0.36 for execution time. From Tables 3 and 4, it can be seen that $\widetilde{T}_{\text{total}}^{\langle c,m\rangle\prime} < D_{edge_i}^{\langle c,m\rangle}$ and $T_{\text{total}}^{\langle c,m\rangle\prime} < D_{edge_i}^{\langle c,m\rangle}$. In other words, each edge has sufficient resources to execute the multi-dependent jobs deployed to it. This further indicates the suitability or accuracy of the trained ML model to provide the necessary information for our intelligent dispatching and co-location strategy.

4.3.2. Performance comparisons across integrated edge-enabled CL clusters

We applied our EdgeColla to orchestrate 209 jobs across 6 integrated edge clusters and compare its performance with those of the benchmark schemes. We first investigated multi-job $\mathbb{J}$ scheduled across the 6 integrated clusters, as depicted in Fig. 5. It can be observed that both EdgeColla (FDFP) and NPFP are able to deploy 100% of all the jobs in $\mathbb{J}$. PDFP and PDNP are slightly inferior and could not deploy 100% of the jobs in $\mathbb{J}$ on some clusters. Specifically, PDFP only achieves 77% of the multi-job deployments on `AerialEdge 2`, while PDNP only achieves 83% and 77% on `AerialEdge 1` and `AerialEdge 2`, respectively. The Random approach could barely schedule 50% of $\mathbb{J}$ to 5 clusters and the percentage of its scheduled jobs is much lower compared to other schemes. Because it deploys a task randomly to any available node, this results in longer delays for dependent tasks, resource underutilization and inability to execute all jobs. Therefore, we only show the results and performance comparisons for the multi-dependent jobs/tasks that are deployed or scheduled successfully in the integrated edge system.

Fig. 6 compares the actual resource usage $D_{edge_i ARU}^{\langle c,m\rangle}$ of EdgeColla with those of the 3 baseline schemes and the Random approach. It can be seen that EdgeColla consumes the fewest resources across the integrated clusters with NPFP as the very close second best, while Random is the worst and PDNP as the second worst. PDFP ranks in the middle, in terms of resource usage across the integrated clusters. The actual resource utilization (CPU resource utilization $\rho_{edge_i}^{\langle c\rangle}$ and memory resource utilization $\rho_{edge_i}^{\langle m\rangle}$) comparisons are shown in Figs. 7 and 8, respectively. Again, EdgeColla and NDFP are superior to PDFP, PDNP, and Random, and they achieve the highest and close second highest resource utilization across the integrated clusters, respectively, while PDNP and Random achieve the second lowest and lowest resource utilization across the integrated clusters, respectively.

Two other key metrics are the actual task/job scheduling time $\sum_{J\in\mathbb{J}}\sum_{z=1}^{m}\sum_{i=1}^{k_z} S_{ch_{z_i}}/k_z$, where m is the number of scheduling units, k_z is the number of tasks within the z-th scheduling unit, and more importantly, the actual execution time of multi-dependent jobs/tasks $E_{ex\prime}^{\text{total}}$. Figs. 9 and 10 compare the actual task/job scheduling time and task/job execution time of EdgeColla with those of the 4 benchmarks, respectively. The results show that EdgeColla is the best, NDFP is the second best, and PDFP is the third best, while Random is the worst and PDNP is the second worst, in terms of both scheduling time and

[13] As the drone edges involved have sufficient flight time, the estimated execution time is not required in selecting drones.

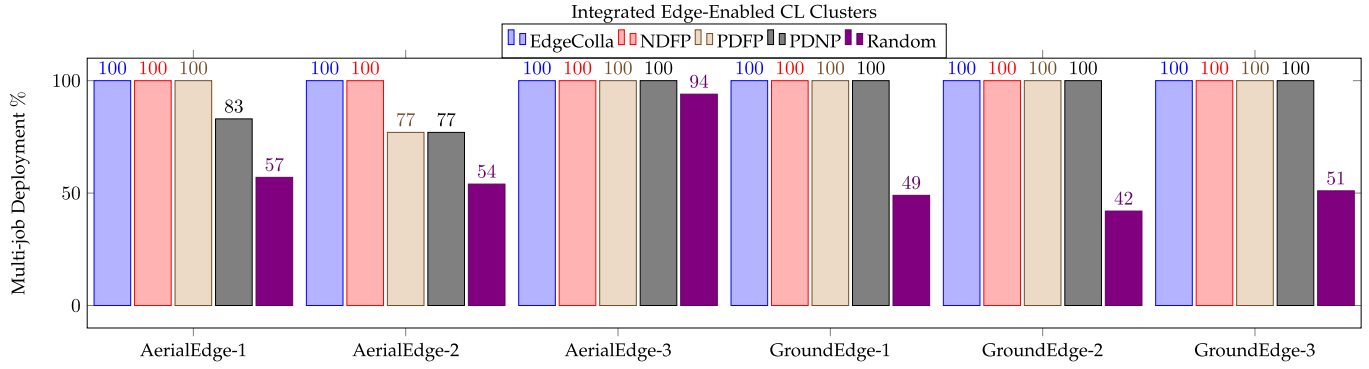


Fig. 5. Multi-job dispatching and co-location across integrated edge-enabled CL clusters.

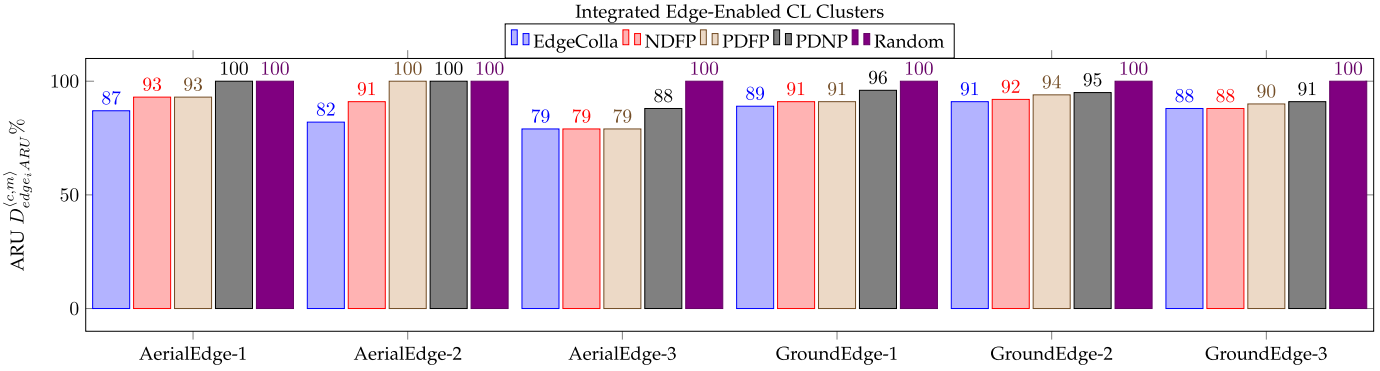


Fig. 6. Actual resource usage across integrated edge-enabled CL clusters.

execution time. The superior performance of EdgeColla over the other benchmarks is overwhelmingly clear.

4.3.3. *Performance comparisons in individual clusters*

Figs. 6–10 show the performance of the schemes in terms of the multi-job deployment, actual resource usage, and resource utilization, task scheduling and execution time across the integrated clusters. We now delve into the individual clusters to examine the performance of all the schemes.

`AerialEdge-1` is a drone attached with edge devices `Lenovo SE350`, `HIVECELL`, `Xavier NX`, and `Dell 3000`, with a total resource capacity of 30 Cores and 274 GiB memory, respectively. The entire weight of the devices is ≈13 kg. We deploy 5 jobs with a total of 35 tasks, where the job has a task dependency depth γ (2, 16]. Utilizing the gang scheduling strategy, EdgeColla co-shedules and co-locates all the 5 jobs at a time in nodes to minimize the overall used nodes. These jobs are tightly co-located, which enables dependent tasks to communicate and share data effectively. As a result, EdgeColla achieves the fastest scheduling time and execution time compared to NDFP, PDFP, PDNF, and the Random approach. In addition, EdgeColla only uses 87% of cluster resources to execute the jobs. In the same cluster, NPFP, PDFP, and PDNP utilize 93%, 93%, and 100% of the cluster resources, respectively. The Random approach uses all the cluster resources as well. It is observed that EdgeColla is 5 times and 3 times faster than the second-best NPFP in both the scheduling time and execution time, respectively. EdgeColla is more than 13 times and 5 times faster than PDFP as well as more than 18 times and 6 times faster than PDNP in the scheduling time and execution time, respectively. EdgeColla is 50 times and 21 times faster than the Random approach in the scheduling time and execution time, respectively.

Like `AerialEdge-1`, `AerialEdge-2` is also a drone and a small-capacity cluster. It is made up of AWS `Snowcone`, `Huawei AR502H`, and `Intelliedge G700(x2)` portable edge devices with a total weight of ≈14 kg, and total resource capacity of 22 Cores and 38 GiB memory, respectively. Here, we deployed $\mathbb{J} = 13$, where each $J \in \mathbb{J}$ has a task dependency in the range of (1, 7]. The total number of tasks in $\mathbb{J}$ is 26. We ensure that the attached edge resources are fully utilized by co-locating the jobs tightly on them. As discussed earlier, the application containers provide isolation to co-located tasks, thereby eliminating interference and resource contentions in the cluster. A single node is capable of running several containerized tasks, given that available resources are sufficient. In this cluster, EdgeColla consumes 9% fewer resources than NDFP, and 18% fewer resources than PDFP, PDNP, and Random. EdgeColla also gains 10% higher CPU utilization over NDFP, and 18% higher CPU utilization over PDFP, PDNP, and Random, as well as 2% higher memory utilization than NDFP, and 4% higher memory utilization than PDFP, PDNP and Random. More significantly, EdgeColla is 2.4, 11 and 26 times faster in the scheduling time than NDFP, PDFP, and PDNP, respectively, while it is 3, 5, and 8 times faster in the execution time than NDFP, PDFP, and PDNP, respectively. Note that for Random, PDFP, and PDNP, the results of the actual resource usage, resource utilization, scheduling and execution time are 54%, 77% and 77% of the multi-dependent jobs that can be scheduled on `AerialEdge-2`, respectively.

`AerialEdge-3` is the last of the drone clusters. It is attached with `Dell 3000`, `Dell 5000`, `aiSage`, and `dynaEdge(x4)` portable edge devices. It has a high load capacity of ≈20 kg compared to `AerialEdge-1` and `AerialEdge-2`. It also has a higher resource capacity of 48 Cores and 112 GiB memory, respectively compared to the previous drone clusters. In this cluster, we deploy $\mathbb{J} = 8$ and a total of 49 tasks, where each $J \in \mathbb{J}$ has a task dependency depth γ ranging from (2, 16]. In this cluster, EdgeColla, NDFP, and PDFP achieve reduced $D^{\langle c,m \rangle}_{edge_i ARU}$ by 11% and 21% compared with PDNP and Random, respectively. EdgeColla, NDFP, and PDFP achieve 9% and 20% higher CPU utilization as well as 2% and 4% higher memory utilization compared to PDNP and Random, respectively. In terms of scheduling, EdgeColla is about 4, 14, 45, and 96 times faster than NDFP, PDFP, PDNP, and

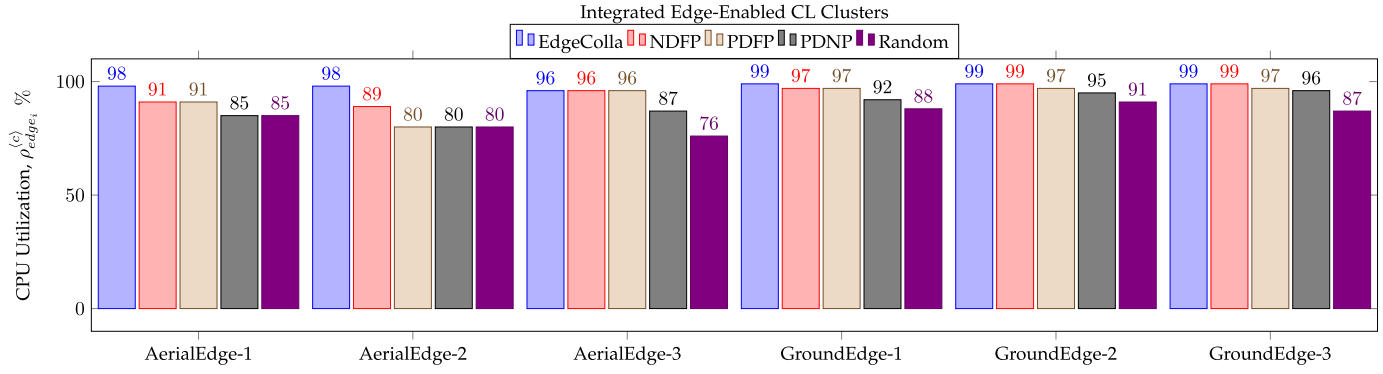


Fig. 7. CPU utilization across integrated edge-enabled CL clusters.

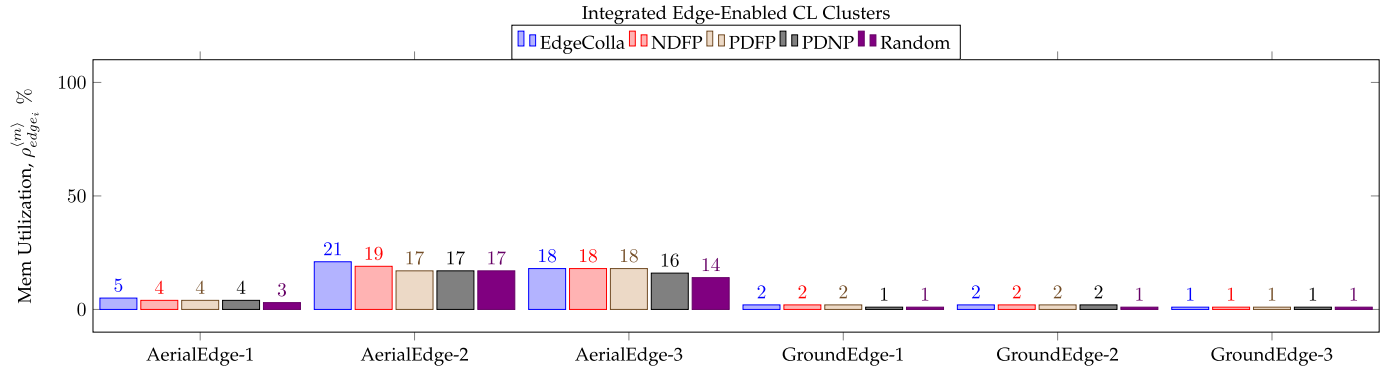


Fig. 8. Memory utilization across integrated edge-enabled CL clusters.

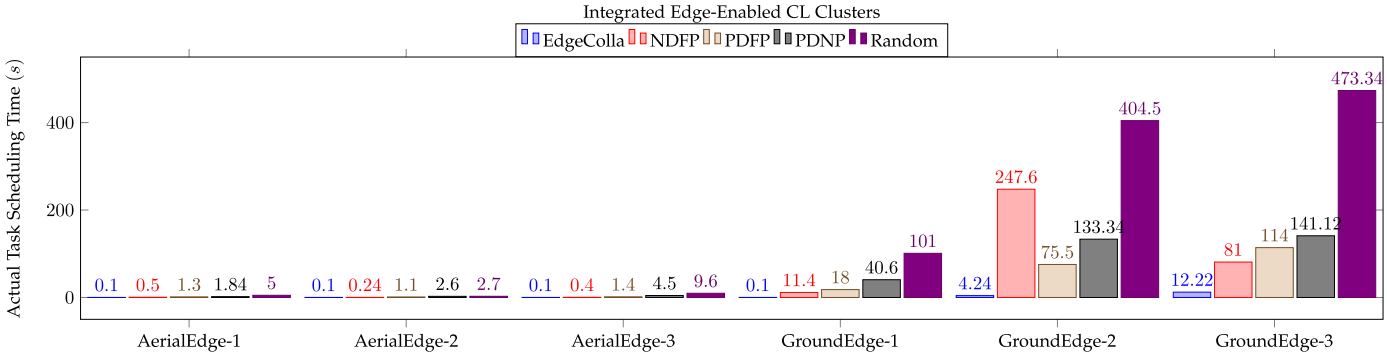


Fig. 9. Actual multi-job scheduling time across integrated edge-enabled CL clusters.

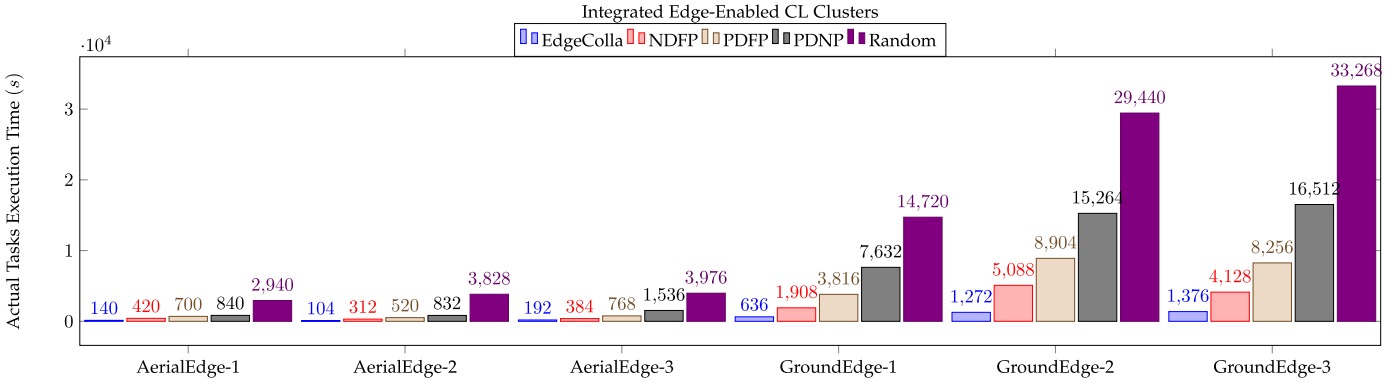


Fig. 10. Actual multi-job execution time across integrated edge-enabled CL clusters.

Random, respectively. It achieves 2, 4, 8, and 20.7 times faster execution time than NDFP, PDFP, PDNP, and Random, respectively. Not surprisingly, Random has the worst scheduling time and execution time performance.

`GroundEdge-1` is an on-premise cluster. Generally, on-premise clusters are higher than drone clusters in terms of resource capacities. This cluster particularly is a memory-intensive cluster. It is made of `Stack Edge`, `DELL EMC`, `AWS Snowball`, and `Lenovo ThinkSystem` on-premise edge devices, with a resource capacity of 132 Cores and a memory capacity of 3536 GiB, respectively. Here, we deploy $\mathbb{J} = 34$ with a total of 159 tasks. Each $J \in \mathbb{J}$ has a dependency depth γ ranging from $(2,\ 16]$. It is observed that EdgeColla consumes the fewest resources at 89%, followed by NDFP and PDFP at 91% each. PDNP consumes 96% of the resources, while the Random approach uses all the available resources. EdgeColla also achieves 2%, 2%, 7%, and 11% higher CPU utilization over NDFP, PDFP, PDNP, and Random, respectively. Note that the clusters `GroundEdge 1`, `GroundEdge 2`, and `GroundEdge 3` are memory-intensive clusters, i.e., they have huge memory capacities. Therefore, the jobs can only consume a few such capacities, as shown in Fig. 8. It is worth noting that Random can only schedule 49% of the tasks. By contrast, EdgeColla, NDFP, PDFP, and PDNP all schedule 100% of the jobs. In terms of scheduling time, EdgeColla is approximately 114, 180, and 406 times faster than NDFP, PDFP, and PDNP, respectively. In terms of execution time, EdgeColla is about 3, 6, and 12 times faster than NDFP, PDFP, and PDNP, respectively. In this cluster, Random can only schedule 49% of all the tasks within the jobs and it has the worst performance for scheduling time and execution time.

`GroundEdge-2` and `GroundEdge-3` are the largest on-premise clusters in terms of resource capacities. We deploy a combined $\mathbb{J} = 149$ in these 2 clusters. The combined number of tasks deployed in both clusters is 662. The task dependency depth γ of each $J \in \mathbb{J}$ is in the range of $(2,\ 16]$. Random can only deploy 41% and 51% of the tasks in these high-capacity on-premise clusters, respectively. In `GroundEdge-2`, EdgeColla uses 1%, 3%, and 4% fewer resources, compared with NDFP, PDFP, and PDNP, respectively. EdgeColla and NDFP also achieve 2% and 4% higher CPU utilization over PDFP and PDNP, respectively. All the schemes, except Random, achieve the same memory utilization. In terms of scheduling time, EdgeColla is approximately 58, 18, 34, and 95 times faster than NDFP, PDFP, PDNP, and Random, respectively. In terms of execution time, EdgeColla is about 4, 7, 12, and 23 times faster than NDFP, PDFP, PDNP, and Random, respectively. In `GroundEdge-3`, EdgeColla and NDFP use 2% and 3% fewer resources than PDFP and PDNP, respectively. EdgeColla and NDFP also achieve 2% and 3% higher CPU utilization than PDFP and PDNP, respectively. In terms of memory utilization, all the 5 schemes achieve the same utilization. In terms of scheduling time, EdgeColla is 6.6 times faster than NDFP as well as 9.3 and 12 times faster than PDFP and PDNP, respectively. In terms of execution time, EdgeColla is 3, 6, and 12 times faster than NDFP, PDFP, and PDNP, respectively. EdgeColla is 38.7 times faster and 24 times faster than Random in the scheduling time and execution time, respectively.

4.4. Discussions

Overall, EdgeColla has demonstrated better performance in an integrated EC system. It has consistently outperformed baseline schemes (NDFP, PDFP, PDNP and Random) by achieving faster scheduling time and excution time, and using fewer resources. Utilizing fewer resources can allow for more tasks to be executed, thereby improving the overall throughput of EC systems. Effective multi-task dispatching of EdgeColla across the integrated clusters provides overall system load balancing, thereby eliminating any resource overload problem. The performance of EdgeColla can be attributed to its effective dispatching policy, gang-deployment and co-location of multi-dependent jobs, which allows inter-dependent tasks within each job to communicate and share data faster. Such fast execution is crucial for EC applications to perform better. The existing schemes do not consider task's dependencies or multi-task co-location, leading to edge resource wastage and underutilization, as well as causing execution delay.

5. Conclusions

This paper has presented an intelligent multi-dependent task dispatching and co-location scheme called EdgeColla. Specifically, we derived a CL-based multi-dependent task resource requirements and execution time estimation method for an integrated edge system through the control panel, such that multi-dependent tasks are intelligently dispatched to the *closest* edge cluster having sufficient resources. To guarantee the optimal usage of cluster resources, we further utilize a variant bin-packing optimization approach through gang-scheduling multi-dependent tasks to co-schedule and co-locate tasks firmly on available nodes, so as to avoid resource wastage. Our experimental results demonstrated that EdgeColla is capable of minimizing the actual completion time of multi-dependent tasks using minimum resources, and we conducted extensive experiments to compare the performance of our EdgeColla with several existing approaches using the real-world Alibaba cluster dataset, which provides information on task dependencies in an integrated edge system.

CRediT authorship contribution statement

Uchechukwu Awada: Writing – original draft, Methodology, Formal analysis. **Jiankang Zhang:** Writing – review & editing, Validation, Supervision, Resources, Funding acquisition, Conceptualization. **Sheng Chen:** Writing – review & editing, Validation, Methodology. **Shuangzhi Li:** Validation, Resources, Investigation, Funding acquisition. **Shouyi Yang:** Writing – review & editing, Supervision, Resources, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgements

The financial support of the National Natural Science Foundation of China under grants 61901416 and 61571401 (part of the Natural Science Foundation of Henan under grant 242300420269), the Young Elite Scientists Sponsorship Program of Henan under grant 2024HYTP026, and the Innovative Talent of Colleges and the University of Henan Province under grant 18HASTIT021 are gratefully acknowledged.

References

[1] U. Bokhari Mohammad, Q. Makki, Y.K. Tamandani, A survey on cloud computing, in: V.B. Aggarwal, V. Bhatnagar, K. Mishra, Durgesh (Eds.), Big Data Analytics, Springer, Singapore, 2018, pp. 149–164.

[2] U. Awada, J. Zhang, Edge federation: a dependency-aware multi-task dispatching and co-location in federated edge container-instances, in: 2020 IEEE International Conference on Edge Computing (EDGE), IEEE, 2020, pp. 91–98.

[3] H. Guo, J. Liu, Uav-enhanced intelligent offloading for Internet of things at the edge, IEEE Trans. Ind. Inform. 16 (4) (2020) 2737–2746.

[4] Z. Yu, Y. Gong, S. Gong, Y. Guo, Joint task offloading and resource allocation in uav-enabled mobile edge computing, IEEE Int. Things J. 7 (4) (2020) 3147–3159.

[5] X. Cao, G. Tang, D. Guo, Y. Li, W. Zhang, Edge federation: towards an integrated service provisioning model, IEEE/ACM Trans. Netw. 28 (3) (2020) 1116–1129.

[6] U. Awada, J. Zhang, S. Chen, S. Li, Air-to-air collaborative learning: a multi-task orchestration in federated aerial computing, in: 2021 IEEE 14th International Conference on Cloud Computing (CLOUD), IEEE, 2021, pp. 671–680.

[7] U. Awada, J. Zhang, S. Chen, S. Li, Airedge: a dependency-aware multi-task orchestration in federated aerial computing, IEEE Trans. Veh. Technol. 71 (1) (2022) 805–819.
[8] G. Faraci, C. Grasso, G. Schembra, Fog in the clouds: UAVs to provide edge computing to iot devices, ACM Trans. Internet Technol. 20 (3) (2020) 1–26.
[9] J. Ren, H. Wang, T. Hou, S. Zheng, C. Tang, Federated learning-based computation offloading optimization in edge computing-supported Internet of things, IEEE Access 7 (2019) 69194–69201.
[10] R. Yu, P. Li, Toward resource-efficient federated learning in mobile edge computing, IEEE Netw. 35 (1) (2021) 148–155.
[11] Q. Wu, K. He, X. Chen, Personalized federated learning for intelligent iot applications: a cloud-edge based framework, IEEE Open Journal of the Computer Society 1 (2020) 35–44.
[12] R. Urgaonkar, S. Wang, T. He, M. Zafer, K. Chan, K.K. Leung, Dynamic service migration and workload scheduling in edge-clouds, Perform. Eval. 91 (2015) 205–228.
[13] L. Tong, Y. Li, W. Gao, A hierarchical edge cloud architecture for mobile computing, in: IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications, IEEE, 2016, pp. 1–9.
[14] Z. Han, H. Tan, S.H.-C. Jiang, X. Fu, W. Cao, F.C. Lau, Scheduling placement-sensitive bsp jobs with inaccurate execution time estimation, in: IEEE INFOCOM 2020 - IEEE Conference on Computer Communications, IEEE, 2020, pp. 1053–1062.
[15] C. Anderson, Docker [software engineering], IEEE Softw. 32 (3) (2015) 102–c3.
[16] Q. Ren, K. Liu, L. Zhang, Multi-objective optimization for task offloading based on network calculus in fog environments, Digital Communications and Networks 8 (5) (2022) 825–833.
[17] U. Awada, A. Barker, Resource efficiency in container-instance clusters, in: Proceedings of the Second International Conference on Internet of Things, Data and Cloud Computing, ACM, 2017, pp. 1–5.
[18] U. Awada, A. Barker, Improving resource efficiency of container-instance clusters on clouds, in: Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, IEEE, 2017, pp. 929–934.
[19] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, A. Akella, Multi-resource packing for cluster schedulers, in: Proceedings of the 2014 ACM Conference on SIGCOMM, ACM, 2014, pp. 455–466.
[20] Z. Hu, J. Tu, B. Li, Spear: Optimized dependency-aware task scheduling with deep reinforcement learning, in: 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS), IEEE, 2019, pp. 2037–2046.
[21] R. Grandl, S. Kandula, S. Rao, A. Akella, J. Kulkarni, Graphene: packing and dependency-aware scheduling for data-parallel clusters, in: Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation, USENIX Association, USA, 2016, pp. 81–97.
[22] T.-P. Pham, J.J. Durillo, T. Fahringer, Predicting workflow task execution time in the cloud using a two-stage machine learning approach, IEEE Trans. Cloud Comput. 8 (1) (2020) 256–268.
[23] F. Nadeem, D. Alghazzawi, A. Mashat, K. Faqeeh, A. Almalaise, Using machine learning ensemble methods to predict execution time of e-science workflows in heterogeneous distributed systems, IEEE Access 7 (2019) 25138–25149.
[24] M.H. Hilman, M.A. Rodriguez, R. Buyya, Task runtime prediction in scientific workflows using an online incremental learning approach, in: 2018 IEEE/ACM 11th International Conference on Utility and Cloud Computing (UCC), IEEE, 2018, pp. 93–102.
[25] W. Xiao, R. Bhardwaj, R. Ramjee, M. Sivathanu, N. Kwatra, Z. Han, P. Patel, X. Peng, H. Zhao, Q. Zhang, F. Yang, L. Zhou, Gandiva: introspective cluster scheduling for deep learning, in: Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation, USENIX Association, USA, 2018, pp. 595–610.
[26] S. Venkataraman, Z. Yang, M. Franklin, B. Recht, I. Stoica, Ernest: Efficient performance prediction for large-scale advanced analytics, in: Proceedings of the 13th Usenix Conference on Networked Systems Design and Implementation, USENIX Association, 2016, pp. 363–378.
[27] Y. Peng, Y. Bao, Y. Chen, C. Wu, C. Guo, Optimus: an efficient dynamic resource scheduler for deep learning clusters, in: Proceedings of the Thirteenth EuroSys Conference, ACM, 2018, pp. 1–14.
[28] C. Delimitrou, C. Kozyrakis, Quasar: resource-efficient and qos-aware cluster management, SIGPLAN Not. 49 (4) (2014) 127–144.
[29] Z. Guo, K. Yu, N. Kumar, W. Wei, S. Mumtaz, M. Guizani, Deep-distributed-learning-based poi recommendation under mobile-edge networks, IEEE Int. Things J. 10 (1) (2023) 303–317.
[30] R.-A. Cherrueau, A. Lebre, D. Pertin, F. Wuhib, J.M. Soares, Edge computing resource management system: a critical building block! Initiating the debate via OpenStack, in: USENIX Workshop on Hot Topics in Edge Computing (HotEdge 18), USENIX Association, 2018.
[31] V.S. Marco, B. Taylor, B. Porter, Z. Wang, Improving spark application throughput via memory aware task co-location: a mixture of experts approach, in: Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference, ACM, 2017, pp. 95–108.
[32] Y. Li, D. Sun, B.C. Lee, Dynamic colocation policies with reinforcement learning, ACM Trans. Archit. Code Optim. 17 (1) (2020) 1–25.
[33] C. Shu, Z. Zhao, Y. Han, G. Min, H. Duan, Multi-user offloading for edge computing networks: a dependency-aware and latency-optimal approach, IEEE Int. Things J. 7 (3) (2020) 1678–1689.
[34] J. Liu, H. Shen, Dependency-aware and resource-efficient scheduling for heterogeneous jobs in clouds, in: 2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), IEEE, 2016, pp. 110–117.
[35] J. Lee, H. Ko, J. Kim, S. Pack, Data: dependency-aware task allocation scheme in distributed edge clouds, IEEE Trans. Ind. Inform. 16 (12) (2020) 7782–7790.
[36] Y. Liu, S. Wang, Q. Zhao, S. Du, A. Zhou, X. Ma, F. Yang, Dependency-aware task scheduling in vehicular edge computing, IEEE Int. Things J. 7 (6) (2020) 4961–4971.
[37] Z. Ji, L. Chen, N. Zhao, Y. Chen, G. Wei, F.R. Yu, Computation offloading for edge-assisted federated learning, IEEE Trans. Veh. Technol. 70 (9) (2021) 9330–9344.
[38] J. Konečný, H.B. McMahan, F.X. Yu, P. Richtarik, A.T. Suresh, D. Bacon, Federated learning: strategies for improving communication efficiency, in: NIPS Workshop on Private Multi-Party Machine Learning, Curran Associates, Inc., 2016, pp. 5–10.
[39] Y. Chen, X. Sun, Y. Jin, Communication-efficient federated deep learning with layer-wise asynchronous model update and temporally weighted aggregation, IEEE Trans. Neural Netw. Learn. Syst. 31 (10) (2020) 4229–4238.
[40] Z. Chen, W. Liao, K. Hua, C. Lu, W. Yu, Towards asynchronous federated learning for heterogeneous edge-powered Internet of things, Digital Communications and Networks 7 (3) (2021) 317–326.
[41] Y. Li, T. Wang, Y. Wu, W. Jia, Optimal dynamic spectrum allocation-assisted latency minimization for multiuser mobile edge computing, Digital Communications and Networks 8 (3) (2022) 247–256.
[42] U. Awada, J. Zhang, S. Chen, S. Li, S. Yang, Edgedrones: co-scheduling of drones for multi-location aerial computing missions, J. Netw. Comput. Appl. 215 (2023) 103632.
[43] H. Tan, Z. Han, X.-Y. Li, F.C. Lau, Online job dispatching and scheduling in edge-clouds, in: IEEE INFOCOM 2017 - IEEE Conference on Computer Communications, IEEE, 2017, pp. 1–9.
[44] S. Rampersaud, D. Grosu, Sharing-aware online virtual machine packing in heterogeneous resource clouds, IEEE Trans. Parallel Distrib. Syst. 28 (7) (2017) 2046–2059.
[45] J. Guo, Z. Chang, S. Wang, H. Ding, Y. Feng, L. Mao, Y. Bao, Who limits the resource efficiency of my datacenter: an analysis of alibaba datacenter traces, in: 2019 IEEE/ACM 27th International Symposium on Quality of Service (IWQoS), IEEE, 2019, pp. 1–10.
[46] H. Wu, W. Zhang, Y. Xu, H. Xiang, T. Huang, H. Ding, Z. Zhang, Aladdin: optimized maximum flow management for shared production clusters, in: 2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS), IEEE, 2019, pp. 696–707.
[47] F. Li, B. Hu, Deepjs: job scheduling based on deep reinforcement learning in cloud data center, in: Proceedings of the 2019 4th International Conference on Big Data and Computing, ACM, 2019, pp. 48–53.
[48] U. Awada, J. Zhang, S. Chen, S. Li, S. Yang, Resource-aware multi-task offloading and dependency-aware scheduling for integrated edge-enabled iov, J. Syst. Archit. 141 (2023) 102923.
[49] W. Huang, Z. Zeng, N.N. Xiong, S. Mumtaz, Joet: sustainable vehicle-assisted edge computing for iot devices, J. Syst. Archit. 131 (2022) 102686.