Machine Learning Driven Latency Optimization for Internet of Things Applications in Edge Computing

Uchechukwu AWADA[1], ZHANG Jiankang[2], CHEN Sheng[3,4], LI Shuangzhi[1], YANG Shouyi[1]

(1. Zhengzhou University, Zhengzhou 450001, China;
2. Bournemouth University, Poole BH12 5BB, UK;
3. University of Southampton, Southampton SO17 1BJ, UK;
4. Ocean University of China, Qingdao 266100, China)

DOI: 10.12142/ZTECOM.202302007

https://kns.cnki.net/kcms/detail/34.1294.TN.20230516.1317.002.html, published online May 17, 2023

Manuscript received: 2023-03-11

Abstract: Emerging Internet of Things (IoT) applications require faster execution time and response time to achieve optimal performance. However, most IoT devices have limited or no computing capability to achieve such stringent application requirements. To this end, computation offloading in edge computing has been used for IoT systems to achieve the desired performance. Nevertheless, randomly offloading applications to any available edge without considering their resource demands, inter-application dependencies and edge resource availability may eventually result in execution delay and performance degradation. We introduce Edge-IoT, a machine learning-enabled orchestration framework in this paper, which utilizes the states of edge resources and application resource requirements to facilitate a resource-aware offloading scheme for minimizing the average latency. We further propose a variant bin-packing optimization model that co-locates applications firmly on edge resources to fully utilize available resources. Extensive experiments show the effectiveness and resource efficiency of the proposed approach.

Keywords: edge computing; execution time; IoT; machine learning; resource efficiency

Citation (Format 1): AWADA U, ZHANG J K, CHEN S, et al. Machine learning driven latency optimization for Internet of Things applications in edge computing [J]. ZTE Communications, 2023, 21(2): 40 – 52. DOI: 10.12142/ZTECOM.202302007

Citation (Format 2): U. Awada, J. K. Zhang, S. Chen, et al., "Machine learning driven latency optimization for Internet of Things applications in edge computing," *ZTE Communications*, vol. 21, no. 2, pp. 40 – 52, Jun 2022. doi: 10.12142/ZTECOM.202302007.

1 Introduction

The Internet of Things (IoT) describes physical devices that are connected to the Internet or networks for the purpose of exchanging and sharing data. IoT enables direct fusion of physical devices into computer systems, resulting in efficiency, more reliable services and economic benefits without human intervention. However, most IoT devices have limited or no computing capability to meet some application-specific requirements. For example, emerging IoT technologies such as the smart city[1], healthcare-IoT[2], Internet of Vehicles (IoV)[3 – 5], connected and autonomous vehicles (CAVs)[6], and industry 4.0[7], require substantial resources to execute their applications. In addition, most of these applications are structured as a collection of loosely-coupled services that communicate with one another and are often latency-sensitive. A conventional approach is to offload these applications to a cloud computing (CC)[8] data center for execution. CC provides an on-demand availability of compute resources over multiple locations, each of which is a data center. However, a CC data center could be hundreds or thousands of miles away from the data sources, thereby jeopardizing the application performance through longer response time. A recent innovative distributed computing paradigm referred to as edge computing (EC)[9] brings computation and storage resources closer to the locations where they are needed, to reduce response time and save bandwidth. This enabling architecture deploys computation and storage resources at the edge of a network, and even beyond the edge of the network. It is important to note that EC computational resources are also limited compared to CC resources, but EC benefits IoT systems by deploying computing resources closer to end devices, thus reducing network traffic and latency to enable real-time insights. To this end, existing research works have exploited

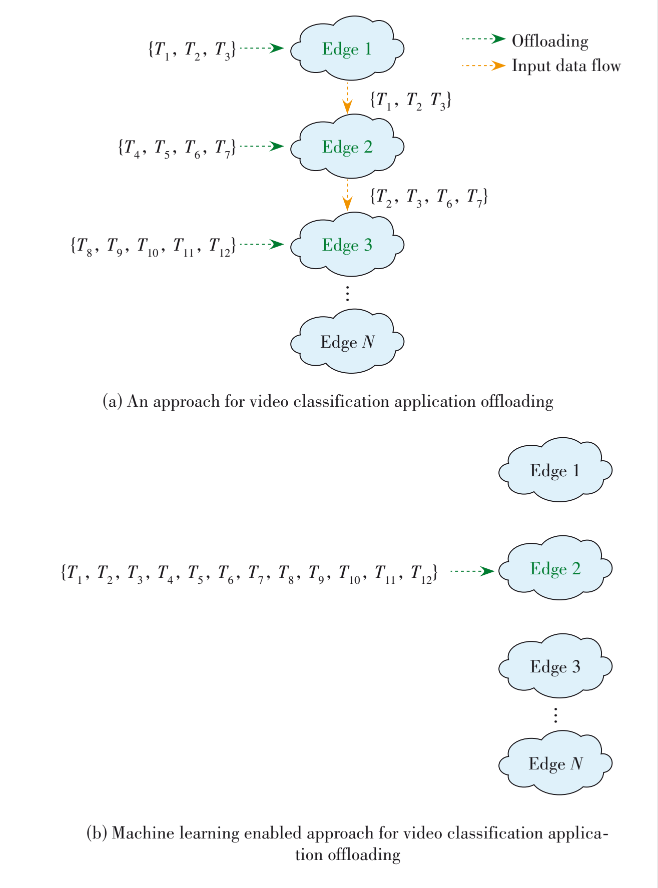
This work is supported by the National Natural Science Foundation of China under Grant Nos. 61571401 and 61901416 (part of the China Postdoctoral Science Foundation under Grant No. 2021TQ0304) and the Innovative Talent Colleges and the University of Henan Province under Grant No.18HASTIT021.

EC for task offloading in various IoT systems[3–5, 10–11]. Nevertheless, one fundamental challenge is where and how to offload and schedule complex applications so that their average latency is minimized and high resource efficiency is achieved. A common practice is to randomly offload applications or tasks individually to available edges without jointly considering task resource demands, task dependencies and edge resource availability. Such a disjointed approach would result in execution delays due to insufficient resource availability or tasks unable to communicate with their dependent tasks. Hence, it is not suitable for latency-sensitive tasks.

For example, the video classification application shown in Fig. 1(a) consists of 12 sub-applications $T_1,\cdots,T_{12}$, where T_1, T_2 and T_3 are independent tasks, whereas T_4 and T_5 require inputs from T_1 to be able to complete their executions. Similarly, T_6, T_7 and T_8 depend on the completion of T_4, T_5 and T_2, respectively. These make the execution of complex IoT applications very challenging. It is naturally important to offload and schedule such applications, to minimize their average latency. For instance, suppose each sub-application or tasks $T_1,\cdots,T_n$ of the application in Fig. 1(a) are randomly offloaded to different EC deployments, and then each dependent task would require the execution result(s) or input data from other task(s) to be transmitted back to its host edge deployment to complete its execution, as shown in Fig. 2(a). This transfer of input data is referred to as an input data flow, and such transmission would incur additional delay, thereby further affecting the average latency, given the rate and number of transmissions that could occur.

More specifically, assuming the video classification application in Fig. 1(a) is to be executed, the work in Ref. [5] proposed an approach as shown in Fig. 2(a), which offloads tasks T_1, T_2 and T_3 to Edge 1, tasks T_4, T_5, T_6 and T_7 to Edge 2, and the remaining tasks T_8, T_9, T_{10}, T_{11} and T_{12} to Edge 3. Since these tasks are interdependent tasks, the execution result of task T_1 needs to be transmitted from Edge 1 to Edge 2, to serve as the input data to tasks T_4 and T_5, while the execution results of tasks T_6 and T_7 need to be transmitted from Edge 2 to edge Edge 3, to serve as the input data to task T_{10}. Finally, the execution results of tasks T_2 and T_3 need to be transmitted $\langle c_1, m_1\rangle$ from Edge 1 to Edge 3 to complete the video classification application execution.

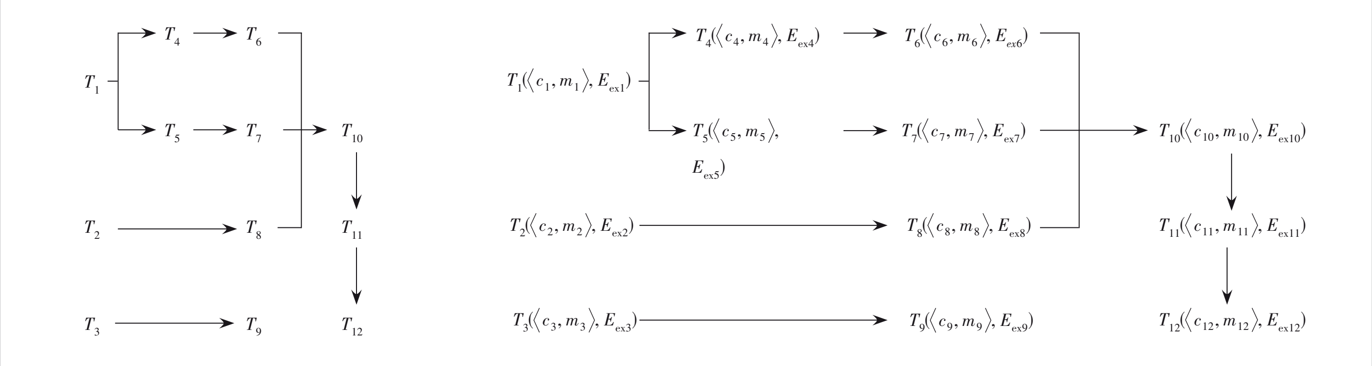
In this paper, we show that machine learning (ML) techniques enable effective IoT task offloading and scheduling in edge computing systems. We propose an ML linear regression model to predict or estimate the resource requirements and



(a) An approach for video classification application offloading

(b) Machine learning enabled approach for video classification application offloading

▲**Figure 2. Application offloading strategies**


T_1
T_4
T_6
T_5
T_7
T_{10}
T_2
T_8
T_{11}
T_3
T_9
T_{12}
$T_1(\langle c_1, m_1\rangle, E_{ex1})$
$T_4(\langle c_4, m_4\rangle, E_{ex4})$
$T_6(\langle c_6, m_6\rangle, E_{ex6})$
$T_5(\langle c_5, m_5\rangle, E_{ex5})$
$T_7(\langle c_7, m_7\rangle, E_{ex7})$
$T_{10}(\langle c_{10}, m_{10}\rangle, E_{ex10})$
$T_2(\langle c_2, m_2\rangle, E_{ex2})$
$T_8(\langle c_8, m_8\rangle, E_{ex8})$
$T_{11}(\langle c_{11}, m_{11}\rangle, E_{ex11})$
$T_3(\langle c_3, m_3\rangle, E_{ex3})$
$T_9(\langle c_9, m_9\rangle, E_{ex9})$
$T_{12}(\langle c_{12}, m_{12}\rangle, E_{ex12})$

(a) Video classification application

(b) Video classification application, with each sub-application's CPU and memory resource requirements denoted as $\langle c, m\rangle$ and execution time denoted as E_{ex}

▲**Figure 1. Directed acyclic graphs (DAG) of representative application** execution time of an application, as shown in Fig. 1(b), and intelligently offload them to an edge with sufficient resource availability, as shown in Fig. 2(b). This approach eliminates the need of input data flow, as sub-applications can communicate and share data quickly. However, upon arrival of an application in a suitable edge, the application may perform poorly if the sub-applications are scheduled naively, e.g., in an edge deployment that can only execute one task at any time, where each task is scheduled individually. Therefore, we further propose a variant bin-packing optimization that gang-schedules[12 – 13] and co-locates applications firmly on EC resources to fully utilize available resources. We aim to schedule and execute all the tasks by considering dependencies and resource demands, such that the actual scheduling and execution time is minimized. In summary, to achieve our Edge-IoT implementation, we address the following critical issues:

- We investigate a situation whereby multiple IoT systems can intelligently offload their complex applications to an edge deployment with sufficient resource availability to meet the resource-level demands of the applications, thus facilitating a resource-aware offloading scheme by enabling faster interactions among the applications to maximize their performance.
- Specifically, we derive a multi-task ML resource requirement and execution time estimation, so as to aid the selection of edge deployment with suitable resource availability.
- To guarantee optimal usage of edge resources and faster execution of tasks, 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 show that Edge-IoT is capable of minimizing the response time of IoT applications using minimum resources, and conduct extensive experiments to compare the performance of our Edge-IoT with several existing approaches using real-world data-trace from Alibaba Cluster Trace Program, which provides information on task dependencies.

2 Related Work

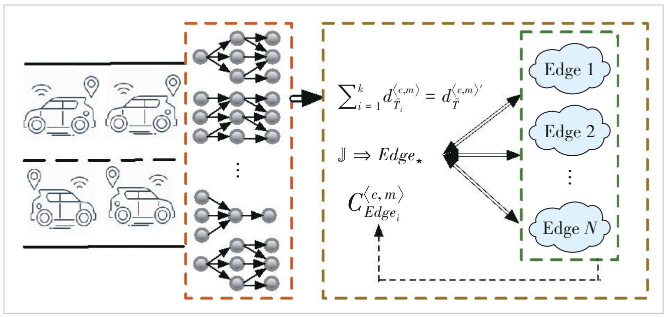
Edge computing has been proven to make the IoT smarter by implementing smart connections and operation of IoT devices[14]. Emerging IoT technologies, such as the smart city[1], healthcare-IoT[2], Internet of Vehicles (IoV)[3 – 5], connected and autonomous vehicles (CAVs)[6], and industry 4.0[7], are utilizing EC for data analysis, processing and monitoring within their networks to improve both the efficiency and response speed. There are a huge number of existing works that have addressed the use of EC for IoT applications. For example, in Ref. [15], the authors studied multi-user IoT application offloading for a mobile edge computing (MEC) system and both the resources of computation and communication were cooperatively allocated. The proposed system focuses on minimizing both the weighted overhead of local IoT devices and the offload measured by the delay and energy consumption. The authors in Ref. [16] formulated two novel optimization problems for delay-sensitive IoT applications, i. e., the total utility maximization problems under both static and dynamic offloading task request settings, to maximize the accumulative user satisfaction on the use of the services provided by an MEC system and show the non-deterministic polynomial time (NP)-hardness of the defined problems. Aiming to maximize the number of IoT devices through jointly optimizing the unmanned aerial vehicle (UAV) trajectory and service indicator as well as resource allocation and computation offloading, the authors in Ref. [17] formulated the optimization problem as a mixed integer nonlinear programming (MINLP) problem, where the chosen IoT devices would complete their computation tasks on time under given energy budgets and co-channel interference was taken into account. In Ref. [18], the authors studied the service home identification problem of service provisioning for multi-source IoT applications in an MEC network, by identifying a service home (cloudlet) of each multi-source IoT application for its data processing, querying and storage. They considered two novel service home identification problems. The work in Ref. [19] presented a joint optimization objective to evaluate the unavailability level, communication delay and resource wastage while allocating the same batch of IoT applications to multiple edge clouds. Then, the authors proposed an approach to minimizing the joint optimization objective under the condition of certain communication delays. In Ref. [20], the authors investigated the issue of joint cooperative edge caching and recommender systems to achieve additional cache gains by the soft caching framework. To measure the cache profits, they formulated the optimization problem as an Integer Linear Programming (ILP) problem, which is NP-hard.

The above methods leverage EC to offload IoT applications. They promise efficiency and better performance, but lack the consideration of a learning-based resource-aware offloading scheme with joint optimization of task resource demands and edge deployment resource availability. Therefore, we propose a joint optimization solution that guarantees faster offloading and execution of IoT applications in edge computing systems.

3 System Model and Problem Formulation

3.1 System Model

We consider an urban vehicular network environment where the IoV applications are offloaded from vehicles to EC deployments across various EC-enabled roadside units (RSUs), EC-enabled base stations (BSs), etc. We focus on V2I application offloading as illustrated in Fig. 3, where each vehicle is equipped with a powerful wireless interface that can be used to connect with RSUs, BSs, etc. We also consider the possibility that each vehicle is equipped with in-vehicle edge devices or deployment. For example, an in-vehicle EC deploy-


$\sum_{i=1}^{k} d_{\tilde{T}_i}^{\langle c,m \rangle} = d_{\tilde{T}}^{\langle c,m \rangle'}$
$\mathbb{J} \Rightarrow Edge_{\star}$
$C_{Edge_i}^{\langle c,m \rangle}$
Edge 1
Edge 2
Edge N

▲Figure 3. An example architecture of Internet of Vehicles (IoV) multi-task offloading

ment may not be as large as the deployments of RSUs, while those of the RSUs may not be as large as the deployments of BSs, etc., in terms of resource capacity. Therefore, IoV applications can be packaged in containers, i.e., Docker container provides a task offloading solution for isolation, portability and lightweight from devices to edge clusters, or to deploy it to the closest edge deployment with sufficient resource availability whenever it is needed. For such applications, let $\langle c, m \rangle$ represent the CPU and memory requirements.

Let $\mathbb{E} = \{Edge_1, \cdots, Edge_M\}$ represent the set of individual participating edge deployments (i. e., in-vehicle, RSU, BS, etc.), as a cluster of container-instances (such as an edge device with virtualized container-optimized nodes). With the resource availability of each participating edge deployment $C_{Edge_i}^{\langle c,m \rangle}$, an informed decision on multi-task offloading can be made. Let $\mathbb{V} = \{\mathcal{V}_1, \cdots, \mathcal{V}_M\}$ represent the index set of vehicles. A vehicle $\mathcal{V}_q$ can choose to execute its ready application locally in its in-vehicle edge device installation if there is sufficient resource availability or it is offloaded to the closest edge deployment $Edge_{i^\star} \in \mathbb{E}$ with sufficient resource availability. Let $\vartheta\left[\mathcal{V}_q(t)\right]$ denote the offloading decision variable, which is measured by

$$\vartheta\left[\mathcal{V}_q(t)\right] = \begin{cases} 1, & \text{tasks are offloaded,} \\ 0, & \text{tasks are processed locally.} \end{cases} \tag{1}$$

A multi-task set $\mathbb{C} = \{T_1, \cdots, T_N\}$ from the vehicles at time t requires much CPU and memory for execution. Such resource requirement, along with its execution time, is first predicted or estimated by a linear regression ML model. The multi-task features, $f_{\mathrm{mt}}(\omega, \epsilon, \gamma)$ where ω is the number of instances, ϵ is the type of tasks, and γ is the dependency depth, are fed into the model $\Theta^\star$ to estimate the values of the resource requirement and execution time according to

$$f_{\mathrm{mt}} \cdot \Theta^\star = \left[\tilde{E}_{\mathrm{ex}_1} \tilde{T}_1^{\langle c,m \rangle} \tilde{E}_{\mathrm{ex}_2} \tilde{T}_2^{\langle c,m \rangle} \cdots \tilde{E}_{\mathrm{ex}_N} \tilde{T}_N^{\langle c,m \rangle}\right], \tag{2}$$

where $\tilde{T}_i^{\langle c,m \rangle}$ and $\tilde{E}_{\mathrm{ex}_i}$ are the estimated resource requirement (in terms of CPU and memory $\langle c, m \rangle$) and estimated execution time for task i, respectively. With these estimated values, a suitable edge deployment can be selected and multi-dependent tasks can be intelligently scheduled with the aim of minimizing their actual response time, while maximizing available resources. Assuming that $\boldsymbol{f}_{\mathrm{mt}} \in \boldsymbol{R}^{1 \times d}$ is a d-dimensional vector (tensor), the predictor $\boldsymbol{\Theta}$ is a $(d \times \epsilon)$-dimensional parameter matrix. We use historical data from previously executed tasks/jobs based on Keras to train the predictor $\boldsymbol{\Theta}$. Keras is a library that wraps TensorFlow complexity into a simple and user-friendly application programming interface (API). Dataset $\mathcal{DS} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^{n}$ contains d-dimensional tensors of data features $\boldsymbol{x}_i \in \boldsymbol{R}^{1 \times d}$ and ϵ-dimensional tensors of labels (the actual execution times) $\boldsymbol{y}_i \in \boldsymbol{R}^{1 \times \epsilon}$. The learning problem is to solve the following optimization:

$$\boldsymbol{\Theta}^\star = \arg \min_{\boldsymbol{\Theta} \in \mathbf{R}^{d \times \epsilon}} \frac{1}{2n} \sum\nolimits_{i=1}^{n} \| \mathbf{x}_i \boldsymbol{\Theta} - \mathbf{y}_i \|_2^2 + \frac{\lambda}{2} \| \boldsymbol{\Theta} \|_F^2, \tag{3}$$

where λ is the regularization parameter and $\|\cdot\|_F$ denotes the Frobenius norm. Optimization (3) is solved using gradient-descent, where the model is updated iteratively until convergence, i. e., $\boldsymbol{\Theta}^{t+1} = \boldsymbol{\Theta}^t - \eta(\frac{1}{n} \boldsymbol{g}(\boldsymbol{\Theta}^t) + \lambda \boldsymbol{\Theta}^t)$, in which η is the learning rate, $\mathbf{g}(\boldsymbol{\Theta}) = \frac{1}{n} \boldsymbol{X}^T(\boldsymbol{X}\boldsymbol{\Theta} - \boldsymbol{Y})$ denotes the gradient of the loss function, $\boldsymbol{X} = [\boldsymbol{x}_1^T \cdots \boldsymbol{x}_n^T]^T$ and $\boldsymbol{Y} = [\mathbf{y}_1^T \cdots \boldsymbol{y}_n^T]^T$ are the feature set and label set, respectively. To guarantee the accuracy of the proposed model, we introduce the normalized absolute estimate error (NAEE), defined as:

$$\mathrm{NAEE} = \frac{\text{lestimated value} - \text{actual valuel}}{\text{actual value}}, \tag{4}$$

for both the resource requirement and execution time estimation, which serves as the estimation accuracy measure for the trained linear regression model.

At time t, while $\vartheta\left[\mathcal{V}_q(t)\right] = 0$, the multi-task set $\mathbb{C} \in \mathcal{V}_q$ is decided to perform local execution procedure in the vehicle $\mathcal{V}_q$; while $\vartheta\left[\mathcal{V}_q(t)\right] = 1$, $\mathbb{C} \in \mathcal{V}_q$ is otherwise to be offloaded to the edge deployment ($Edge_{i^\star}$) with sufficient resources closest to $\mathcal{V}_q$. Multi-task set $\mathbb{C}$ is a loosely coupled inter-dependent application, as shown in Fig. 1, where each task $T \in \mathbb{C}$ has two resource requirements: CPU and memory, as the total number of estimated resources needed for its execution is denoted as $d_{\tilde{T}}^{\langle c,m \rangle}$. For each task $T \in \mathbb{C}$, let E_{sh}, E_{st} and E_{cp} denote its scheduling time, starting time and completion time, respectively. Therefore, the execution time of a task is thus:

$$E_{\mathrm{ex}} = E_{\mathrm{cp}} - E_{\mathrm{st}}. \tag{5}$$

Existing offloading strategies (i.e., Refs. [4], [5], [21], etc.,) allow subtasks of an application or a job to be offloaded separately across different edge deployments, thus creating additional delay in the application's response time, as explained in Section 1. For example, when a vehicle in such an approach begins to offload its tasks, the delay includes three parts: 1) the time for offloading subtasks from the vehicle to different edge deployments, given as E_{of}, 2) the time for transmitting the results of executed subtasks (known as input data flow) from one edge deployment to another edge deployment, given as E_{sub}, and 3) the time for transmitting the final result from EC deployment to the vehicle, given as E_{rst}. Therefore, the response time of the vehicle's job is given as:

$$E_{\text{rsp}} = \sum\nolimits_{T \in \mathbb{C}} \left(E_{\text{of}} + E_{\text{sub}} + E_{\text{sh}} + E_{\text{ex}} \right) + E_{\text{rst}}. \tag{6}$$

In this paper, we aim to offload or dispatch a set of applications $\mathbb{C}$ belonging to a parked or moving vehicle $\mathcal{V}_q$ directly to a single and the closest edge deployment $Edge_{i^\star}$ having sufficient resource capacity or availability to accommodate the tasks such that E_{of} is minimized, E_{sub} is avoided, as well as the overall E_{sh} and E_{ex} are minimized, namely,

$$\mathbb{C} \Rightarrow Edge_\star. \tag{7}$$

Hence, the response time of the vehicle's job changes to:

$$E_{\text{rsp}} = E_{\text{of}} + \sum\nolimits_{T \in \mathbb{C}} \left(E_{\text{sh}} + E_{\text{ex}} \right) + E_{\text{rst}}. \tag{8}$$

Once $\mathbb{C}$ has been offloaded to $Edge_\star$, Edge-IoT utilizes the gang-scheduling[12 – 13] strategy to co-schedule all the applications at a time in $Edge_\star$. Given a cluster of container instances or nodes $I_i \in Edge_\star$, let $I_{Edge_\star}^{\langle c,m \rangle}$ denote each node's resource capacity or availability. In a real scenario where multi-vehicle set $\mathcal{V} \in \mathbb{V}$ offload multi-job tasks at t, these applications are offloaded as a multi-job set $\mathbb{J}$, i.e., $\mathbb{J} \Rightarrow Edge_\star$, where its collective estimated resource demand denoted as $\sum\nolimits_{i=1}^{k} d_{\widetilde{T}_i}^{\langle c,m \rangle} = d_{\tilde{T}}^{\langle c,m \rangle'}$. Hence, we can offload $\mathbb{J}$ to $Edge_\star$ with suitable resource availability. Therefore, the aggregate scheduling time and execution time of multi-job set $\mathbb{J}$ is given as:

$$\sum\nolimits_{J \in \mathbb{J}} \sum\nolimits_{i=1}^{k} \frac{E_{\text{sh}_i}}{k} = E_{\text{sh}}{}', \tag{9}$$

$$\sum\nolimits_{J \in \mathbb{J}} \sum\nolimits_{i=1}^{k} \frac{E_{\text{ex}_i}}{k} = E'_{\text{ex}}. \tag{10}$$

The estimated resource utilization of the edge for multi-job tasks is thus

$$\tilde{\mathcal{U}}_{Edge_i}^{\langle c,m \rangle} = \frac{\sum\nolimits_{J \in \mathbb{J}} d_{\tilde{T}}^{\langle c,m \rangle'}}{C_{Edge_i}^{\langle c,m \rangle}} . \tag{11}$$

Similarly, $\tilde{\mathcal{U}}_{Edge_i}^{\langle c,m \rangle}$ includes CPU utilization $\tilde{\mathcal{U}}_{Edge_i}^{\langle c \rangle}$ and memory utilization $\tilde{\mathcal{U}}_{Edge_i}^{\langle m \rangle}$, which are defined respectively by

$$\tilde{\mathcal{U}}_{Edge_i}^{\langle c \rangle} = \frac{\sum\nolimits_{J \in \mathbb{J}} d_{\tilde{T}}^{\langle c \rangle'}}{C_{Edge_i}^{\langle c \rangle}} , \tag{12}$$

$$\tilde{\mathcal{U}}_{Edge_i}^{\langle m \rangle} = \frac{\sum\nolimits_{J \in \mathbb{J}} d_{\tilde{T}}^{\langle m \rangle'}}{C_{Edge_i}^{\langle m \rangle}} , \tag{13}$$

where $\sum\nolimits_{J \in \mathbb{J}} d_{\tilde{T}}^{\langle c \rangle'}$ and $\sum\nolimits_{J \in \mathbb{J}} d_{\tilde{T}}^{\langle m \rangle'}$ are the total collective estimated CPU and memory, respectively. After completing the multi-job executions, the final execution results are immediately and deterministically transmitted back to the vehicles.

3.2 Problem Formulation

The basic notations adopted are described in Table 1. The objectives are to minimize the response time, E_{rsp} in Eq. (8) for all $J \in \mathbb{J}$ and to maximize the computation or cluster resource utilization $\mathcal{U}_{Edge_i}^{\langle c,\text{m} \rangle}$ in Eq. (11), subject to certain constraints. The response time E_{rsp} in Eq. (8) comprises the dispatching or offloading time E_{of}, the scheduling time E'_{sh}, the execution time E'_{ex}, and the transmission time of final execution results E_{rst}. The closest computation offload-

▼Table 1. Notations

Notation	Description	Notation	Description
$\mathbb{E}$	A set of edge deployments	$\mathcal{V}, \mathbb{V}$	A vehicle, a set of vehicles
T	Individual application or task	I_i	Container-instance or node in a cluster
$\langle c, m \rangle$	CPU and memory resources	$I_i^{\langle c,m \rangle}$	Resource capacity or availability of a node
$\mathbb{C}$	A set of containerized applications	$C_{Edge_i}^{\langle c,m \rangle}$	Resource capacity/availability in an edge
$d_T^{\langle c,m \rangle}$	Application resource requirements	$U_{Edge_i}^{\langle c,m \rangle}$	Resources used for execution
$Edge_i$	Individual edge deployment or cluster	$U_{Edge_i}^{\langle c \rangle}, U_{Edge_i}^{\langle m \rangle}$	CPU, memory resource used for execution
$Edge_\star$	Closest edge deployment or cluster	$RU_{Edge_i}^{\langle c,m \rangle}$	Actual resources usage of jobs
$RU_{Edge_i}^{\langle c \rangle}, RU_{Edge_i}^{\langle m \rangle}$	Actual CPU, memory resources usage	$E_{\text{st}}, E_{\text{cp}}$	Application/task start, completion time
E_{ex}	Application or task execution time	$\mathcal{U}_{Edge_i}^{\langle c,m \rangle}$	Cluster resource utilization
$\mathcal{U}_{Edge_i}^{\langle c \rangle}, \mathcal{U}_{Edge_i}^{\langle m \rangle}$	Cluster CPU, memory resource utilization	$J, \mathbb{J}$	A job, a set of jobs

ing policies are jointly adopted in E_{of}, thus enabling faster offloading time.

1) Constraints

The collective resource demand or request of multi-job set $\mathbb{J}$ at any given time t cannot exceed the collective resource capacity or available in the selected EC deployment:

$$\sum_{J \in \mathbb{J}} d_{\tilde{T}}^{\langle c,m \rangle'} \leqslant C_{Edge_\star}^{\langle c,m \rangle}, \quad \forall_{c,m}, \tag{14}$$

and the unused or inactive nodes $I_i \in Edge_\star$ would be shut down. All the nodes are in active or inactive states. An active node is a node that is running and currently considered for allocation or has at least a job being started, executing or completing. An inactive node is a node that is not running and is not currently considered for allocation or has no job. These two states can be expressed as follows:

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

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

2) Optimization formulation

Hence, maximizing utilization of the selected edge deployment or cluster depends on application orchestration:

$$\text{Maximize} \quad \tilde{\mathcal{U}}_{Edge_i}^{\langle c,m \rangle} = \frac{\sum_{J \in \mathbb{J}} d_{\tilde{T}}^{\langle c,m \rangle'}}{C_{Edge_i}^{\langle c,m \rangle}}, \tag{16}$$

$$\text{subject to} \quad \mathbb{J} \Rightarrow Edge_\star, \quad \exists, \tag{17}$$

$$\beta\left(I_i\right) \in \{0,1\}, \quad \exists, \tag{18}$$

$$\sum_{J \in \mathbb{J}} d_{\tilde{T}}^{\langle c,m \rangle'} \leqslant C_{Edge_\star}^{\langle c,m \rangle}, \quad \forall_{c,m}. \tag{19}$$

The constraints in Eqs. (17) to (19) indicate the dispatching of multi-job set $\mathbb{J}$ to the closest edge having sufficient resource capability or availability. More specifically, Eq. (17) is the constraint for $\mathbb{J}$ offloading, guaranteeing that $\mathbb{J}$ is dispatched to a cluster such that dependent tasks within each $J \in \mathbb{J}$ can communicate and execute faster. Condition (18) guarantees that active nodes ($\beta\left(I_i\right) = 1$) are used for execution and that inactive nodes ($\beta\left(I_i\right) = 0$) are shut down. The constraint in Eq. (19) guarantees that $d_{T}^{\langle c,m \rangle'}$ of $\mathbb{J}$ does not exceed $C_{Edge_i}^{\langle c,m \rangle}$ any selected cluster. The details of our multi-job dispatching principle will be discussed in Section 4.1 and Algorithm 1. We aim to minimize the number of active nodes used for execution by co-locating jobs tightly on each node to maximize resource utilization. The details of our co-location strategy will be discussed in Section 4.2 and Algorithm 2.

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

$$\text{Minimize} \quad \sum_{J \in \mathbb{J}} \sum_{i=1}^{k} \frac{E_{sh_i}}{k} = E_{sh}', \tag{20}$$

$$\text{subject to} \quad \mathbb{J} \Rightarrow Edge_\star, \quad \forall_{c,m}. \tag{21}$$

$$\text{Minimize} \quad \sum_{J \in \mathbb{J}} \sum_{i=1}^{k} \frac{E_{ex_i}}{k} = E_{ex}', \tag{22}$$

$$\text{subject to} \quad \mathbb{J} \Rightarrow Edge_\star, \quad \forall_{c,m}. \tag{23}$$

The constraints in Eqs. (21) and (23) guarantee that $\mathbb{J}$ is dispatched to the same 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 Section 4.1 and Algorithm 1.

4 Edge-IoT Algorithm Framework

The proposed Edge-IoT solution in this paper is focused on offloading and scheduling. The offloading strategy is based on the orchestration of ready multi-job tasks to the closest edge deployment with sufficient available resources to accommodate the tasks, as expressed in Eq. (17), while the scheduling strategy involves packing or co-location of these tasks tightly on container instances to fully utilize the available resources. These components aim at providing optimal performance for vehicular multi-task execution in EC systems such that the optimizations in Eqs. (16), (20) and (22) are achieved.

4.1 Offloading Policy

When sets of vehicular multi-job tasks $\mathbb{J} = J_1,\cdots,J_N$ are ready to be offloaded, our policy is to offload them to the closest edge $Edge_\star$ with sufficient resource capacity or availability, i.e., $\mathbb{J} \Rightarrow Edge_\star$, while $\sum_{J \in \mathbb{J}} d_{\tilde{T}}^{\langle c,m \rangle'} \leqslant C_{Edge_\star}^{\langle c,m \rangle}$. For the rationale of this strategy, consider the Ericsson Connected Vehicle Platform (CVP), which serves about 5.5 million active vehicles across more than 150 countries. Assuming that there are 0.1% of these vehicles at a location $\mathcal{L}$ and at time t deciding to offload their multiple tasks i.e., $\vartheta\left[\mathcal{V} \in \mathbb{V}\right] = 1$, we would see a total load of 4 000 requests. Executing these loads would require an edge deployment with 40 nodes or container instances if we assume that a container instance can co-locate 100 containerized tasks. To serve these vehicles efficiently, it is better to dispatch these tasks as units to a closest edge deployment, i.e., $\mathbb{J} \Rightarrow Edge_\star$, having sufficient resource capacity or availability. The closest heuristic given in Eq. (17) is to minimize the offloading time E_{of} and to further minimize the overall response time E_{rsp}. Algorithm 1 describes the offloading procedure.

Algorithm 1. Edge-IoT: multi-job offloading

```
Input: $\mathbb{J}$ arrived at time $t$; $Edge_i \in \mathbb{E}$; $\sum_{J \in \mathbb{J}} d^{\langle c,m\rangle'}_{\tilde{T}}$
Output: Offload $\mathbb{J}$ to $Edge_\star$ with matching $C^{\langle c,m\rangle}_{Edge_\star}$ such that
        $\mathbb{J} \Rightarrow Edge_\star$
for $Edge_i \in \mathbb{E}$ do
    if $\sum_{J \in \mathbb{J}} d^{\langle c,m\rangle'}_{\tilde{T}} \leqslant C^{\langle c,m\rangle}_{Edge_i}$ then
        $\mathbb{J} \Rightarrow Edge_i = Edge_\star$
    else
        Offload $\mathbb{J}$ to next $Edge_\star$
    end if
end for
if $\mathbb{J}$ cannot be offloaded as a whole then
    for $Edge_i \in \mathbb{E}$ do
        for $J \in \mathbb{J}$ do
            if $\sum_{J \in \mathbb{J}} d^{\langle c,m\rangle'}_{\tilde{T}} \leqslant C^{\langle c,m\rangle}_{Edge_i}$ then
                $J \Rightarrow Edge_i = Edge_\star$
            else
                Dispatch $J$ to next $Edge_\star$
            end if
        end for
    end for
end if
```

4.2 Scheduling Policy

Once $\mathbb{J}$ is offloaded to $Edge_\star$, our scheduling algorithm uses the resource availability $I_i^{\langle c,m\rangle}$ of each container-instance in $Edge_\star$, and the resource demand $d^{\langle c,m\rangle'}_{\tilde{T}}$ of each $J \in \mathbb{J}$ to provide efficient co-location such that fewer container-instances are used for execution in $Edge_\star$. Specifically, the gang scheduling approach is adopted alongside our bin-packing optimization to co-schedule and co-locate all $J \in \mathbb{J}$ 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 in Eq. (31). Unlike other approaches, such as the first fit bin packing problem (FFBPP)[22], it requires the next J_i to be placed on the active node; otherwise, it is placed on a new node. Our scheduling strategy co-locates multi-dependent tasks firmly on nodes (Algorithm 2) such that for any given job, resource wastage is avoided and fewer nodes are used for execution. It takes the resource demand of multi-job tasks and resource availability of nodes as input, then scans all $J \in \mathbb{J}$ and maps them to active nodes in full utilization. Our approach scans all $J \in \mathbb{J}$ and maps J_i to active nodes in full utilization (Line 2 in Algorithm 2). 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 (Lines 4–9 in Algorithm 2).

Algorithm 2. Edge-IoT: multi-job co-location

```
Input: $\mathbb{J}$ offloaded to $Edge_\star$, resource demand of each $J \in \mathbb{J}$:
       $d^{\langle c,m\rangle'}_{\tilde{T}}$, resource availability of each node $I_i \in Edge_\star$: $I_i^{\langle c,m\rangle}$
Output: $\mathbb{J}$ is co-located, such that
        Minimize $\sum_{I_i \in Edge_\star} I_i \equiv$ Minimize $RU^{\langle c,m\rangle}_{Edge_\star}$
1: for $I_i \in Edge_\star$ do
2:     if $\beta\left(I_i\right) = 1$ then
3:         $I_i^{\langle c,m\rangle} = \langle c,m\rangle$, i.e., initial resource available
4:         for $J \in \mathbb{J}$ do
5:             if $\Gamma\left[J, I_i\right] = 0$ and $d^{\langle c,m\rangle'}_{\tilde{T}} \leqslant I_i^{\langle c,m\rangle}$ then
6:                 $J \Rightarrow I_i$
7:                 $\Gamma\left[J, I_i\right] = 1$
8:                 $I_i^{\langle c,m\rangle} = I_i^{\langle c,m\rangle} - d^{\langle c,m\rangle'}_{\tilde{T}}$
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
```

Hence, for every $\mathbb{J}$ offloaded to $Edge_\star$, our co-location strategy is to find the solution to the problem:

$$\text{Minimize} \sum_{I_i \in Edge_\star} I_i \equiv \text{Minimize}\, RU^{\langle c,m\rangle}_{Edge_\star} = \frac{U^{\langle c,m\rangle}_{Edge_\star}}{C^{\langle c,m\rangle}_{Edge_\star}} , \tag{24}$$

$$\text{subject to } \mathbb{J} \Rightarrow Edge_\star, \big| \exists , \tag{25}$$

$$\sum_{J \in \mathbb{J}} \Gamma\left[J, I_i\right] \cdot d^{\langle c,m\rangle'}_{\tilde{T}} \leqslant I_i^{\langle c,m\rangle}, \quad \forall c, m , \tag{26}$$

where

$$\Gamma\left[J, I_i\right] = \begin{cases} 1, & \text{if } J \Rightarrow I_i, \\ 0, & \text{otherwise.} \end{cases} \tag{27}$$

We aim to minimize the number of nodes used for executing $\mathbb{J}$, which is equivalent to minimizing the actual resource usage in $Edge_\star$, given as $RU^{\langle c,m\rangle}_{Edge_\star}$, which is the ratio of the resources used for execution $U^{\langle c,m\rangle}_{Edge_\star}$ over the edge' s resource capacity $C^{\langle c,m\rangle}_{Edge_i}$. The metric $RU^{\langle c,m\rangle}_{Edge_\star}$ includes the actual CPU resource usage $RU^{\langle c\rangle}_{Edge_\star}$ and the actual memory resource usage $RU^{\langle m\rangle}_{Edge_\star}$, which are defined respectively as

$$RU^{\langle c\rangle}_{Edge_\star} = \frac{U^{\langle c\rangle}_{Edge_\star}}{C^{\langle c\rangle}_{Edge_\star}} , \tag{28}$$

$$RU^{\langle m\rangle}_{Edge_\star} = \frac{U^{\langle m\rangle}_{Edge_\star}}{C^{\langle m\rangle}_{Edge_\star}} , \tag{29}$$

where $U^{\langle c\rangle}_{Edge_\star}$ and $U^{\langle m\rangle}_{Edge_\star}$ are the used CPU and memory resources, respectively, while $C^{\langle c\rangle}_{Edge_\star}$ and $C^{\langle m\rangle}_{Edge_\star}$ are the edge's CPU and memory resource capacity, respectively. Then the actual CPU utilization $\rho^{\langle c\rangle}_{\mathcal{DR}_i}$ and the actual memory utilization $\rho^{\langle m\rangle}_{\mathcal{DR}_i}$ are defined respectively by

$$\mathcal{U}^{\langle c\rangle}_{Edge_i} = \frac{\sum_{J\in\mathbb{J}} d_T^{\langle c,m\rangle'}}{U^{\langle c\rangle}_{Edge_\star}}, \tag{30}$$

$$\mathcal{U}^{\langle m\rangle}_{Edge_i} = \frac{\sum_{J\in\mathbb{J}} d_T^{\langle c,m\rangle'}}{U^{\langle c\rangle}_{Edge_\star}}. \tag{31}$$

Algorithms 1 and 2 are directly connected with minimizing E_{sh}', minimizing E_{ex}' as well as maximizing $\tilde{\mathcal{U}}^{\langle c,m\rangle}_{Edge_i}$. Therefore, Eq. (25) is the constraint for multi-job set $\mathbb{J}$ deployment, guaranteeing that $\mathbb{J}$ is offloaded to the closest cluster such that dependent tasks within each $J\in\mathbb{J}$ can communicate and execute faster. As we have stated previously that if $\mathbb{J}$ cannot be dispatched as a whole to a cluster, the dispatcher will allow fractional dispatching of each $J\in\mathbb{J}$ to the closest member edge. The constraint in Eq. (26) indicates that the total estimated resource requirements of co-located jobs $d_{\tilde{T}}^{\langle c,m\rangle'}$ cannot exceed $I_i^{\langle c,m\rangle}$, the node resource availability. The condition in Eq. (27) means that $\Gamma\left[J_i, I_i\right] = 1$ if job J_i is placed on the node I_i; otherwise, $\Gamma\left[J_i, I_i\right] = 0$. 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.

4.3 Connection with Optimization Objectives

Our objectives are to minimize the total response time of multiple IoV applications as stated in Eqs. (20) and (22) and maximize the edge cluster resource utilization in Eq. (26). Algorithms 1 and 2 together achieve these objectives. By offloading multi-job tasks to an edge having sufficient resource availability, Algorithm 1 ensures that any edge deployment selected has sufficient resources $C^{\langle c,m\rangle}_{Edge_\star}$ needed for multi-job execution such that the dependent tasks can be executed faster, ultimately leading to a smaller aggregate scheduling time E_{sh}' and execution time E_{ex}'. By intelligently packing dependent tasks tightly on nodes, Algorithm 2 is capable of fully utilizing available resources at EC clusters, ultimately leading to the resource assigned for the execution of jobs $U^{\langle c,m\rangle}_{Edge_\star}$ to be fewer while guaranteeing it is sufficient for multi-job tasks. More specifically, the resource usage (RU) of the cluster for multi-job tasks is given in Eqs. (28) and (29).

5 Experiment Setup

Our experiment setup consists of six edge deployments distributed across RSUs, BSs and vehicles, as summarized in Table 2. These platforms consist of large resource capacity EC devices. The input data flow time, final result transmission time, vehicle's speed, and road area were drawn from a uniform distribution range of $(0.2, 0.4]$ s, $(0.4, 4]$ s, $(40, 80]$ km/h and [2 km × 2 km], respectively[23]. Therefore, we conduct extensive experiments with orchestrated sets of multi-dependent tasks with heterogeneous resource requests across the EC resources. For each deployment, we compare the performance of our Edge-IoT with the existing state of the art.

As for applications, the v-2018 version of Alibaba cluster trace is used, which records the activities of about 4 000 machines in a period of eight days. The entire trace contains more than 14 million tasks with more than 12 million dependencies and more than four million jobs, among which we deploy a total of 48 jobs with total of 204 tasks (including dependencies) for our experiments. The task dependency depth among the jobs is in the range of (1, 17]. Table 3 lists the details of our multi-job sets.

5.1 Heuristics and Baselines

In our experiments, we assume that all tasks are of high priority. The proposed Edge-IoT utilizes the closest heuristic and adopts the gang-scheduling strategy and a variant bin-packing optimization to efficiently co-schedule and co-locate multi-job tasks in a cluster or edge to minimize the overall response time. We consider Edge-IoT as a full dependency and full packing (FDFP) approach.

We compare the scheduling approach of Edge-IoT with the following three existing schemes, fixing their dispatching policy to that of Edge-IoT, as follows:

1) Full dependency and partial packing (FDPP)[5] is an ap-

▼Table 2. Edge deployments and their resource capacities

Edge Deployment	Edge Device	CPU Capacity	Memory Capacity/GiB
Edge 1	Acer aiSage (x2)	12 Cores	4
Edge 2	AWS Snowcone (x10)	20 Cores	40
Edge 3	Huawei AR502H Series (x6)	24 Cores	12
Edge 4	HIVECELL (x6)	36 Cores	48
Edge 5	NVIDIA Jetson Xavier NX (x3)	36 Cores	24
Edge 6	INTELLIEDGE G700 (x5)	48 Cores	80

▼ Table 3. Multi-job execution, where the actual resources consumed for multi-job execution $d_T^{\langle c,m\rangle}$ are taken from the original Alibaba data and the estimated resource demands $d_{\tilde{T}}^{\langle c,m\rangle'}$are calculated by linear regression model

Multi-Job $\mathbb{J}$	$\mathbb{C}$	T	$d_{\tilde{T}}^{\langle c,m\rangle'}$	$d_T^{\langle c,m\rangle'}$	NAEE
1	5	22	$\langle 1\,195.24, 4.35\rangle$	$\langle 1\,135, 3.77\rangle$	$\langle 0.1, 0.15\rangle$
2	7	29	$\langle 1\,501.5, 5.81\rangle$	$\langle 1\,325, 4.23\rangle$	$\langle 0.13, 0.37\rangle$
3	9	38	$\langle 2\,011.55, 7.57\rangle$	$\langle 1\,820, 5.76\rangle$	$\langle 0.1, 0.3\rangle$
4	12	52	$\langle 2\,762.25, 10.4\rangle$	$\langle 2\,560, 8.2\rangle$	$\langle 0.1, 0.26\rangle$
5	15	63	$\langle 3\,369.68, 12.58\rangle$	$\langle 3185, 10.17\rangle$	$\langle 0.1, 0.23\rangle$

NAEE: normalized absolute estimate error proach that executes subtasks of a job locally in the vehicle and offloads subtasks to the cloud server and the remaining tasks to the RSU for execution at the same time.

2) Full dependency and no packing (FDNP)-1[3] is an approach that offloads all tasks of a job to the same EC deployment, but assumes that at any EC deployment, a node can only execute one task at a time, and FDNP-1 schedules one task at a time. Therefore, unscheduled tasks must wait in a queue until resources become available for the next task(s). Such a queue is constructed based on the application priority, where it keeps multiple applications in decreasing order of their priority.

3) FDNP-2[4] is an approach that offloads different subtasks of a job to different EC deployments, where each node at the selected EC deployment can only schedule and execute one task at a time, and the task with the highest priority is first selected for scheduling.

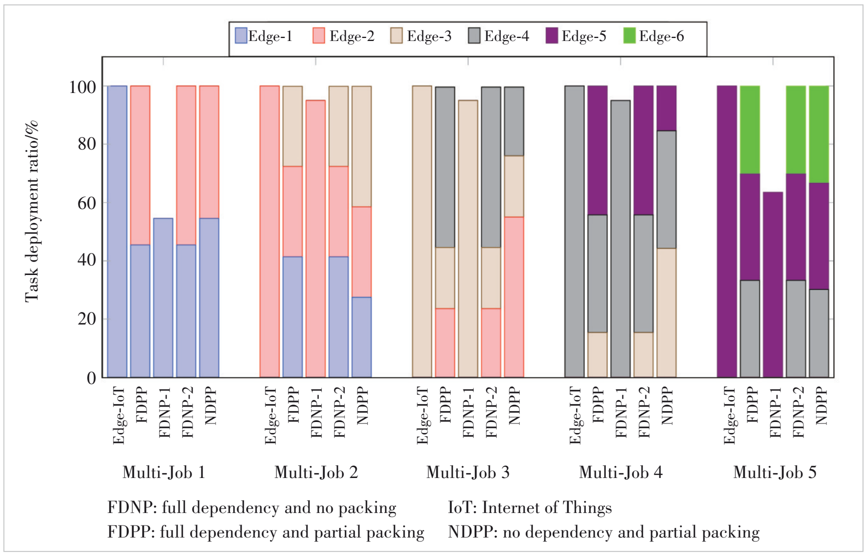
4) No dependency and partial packing (NDPP)[23] is an approach that offloads different multi-job subtasks to available EC deployment, by considering the completion deadline of each task. However, this approach does not respect inter-task dependencies, but co-locates tasks on a node.

5.2 Comparison of Offloading and Execution Results

The investigation focuses on the IoV multi-task response time, which includes the multi-job offloading, resource utilization/usage, scheduling, execution and response time. The multi-job execution information across the edge deployments, obtained according to Alibaba data, are listed in Table 3, where the actual resources consumed for the multi-job execution $d_T^{\langle c,m\rangle'}$ are taken from the original data. NAEE defined in Eq. (4) and listed in Table 3 for resource consumed serves as the estimation accuracy measure for the trained linear regression model. The average NAEE across six deployments is 0.12 for CPU and 0.23 for memory. Note that we only focus on the resource demand estimation for multi-job tasks, as the execution time estimation is not required to select suitable on-premise edge deployments given in Table 2. The results obtained by Edge-IoT (FDFP), FDPP, FDNP-1, FDNP-2 and NDPP are compared.

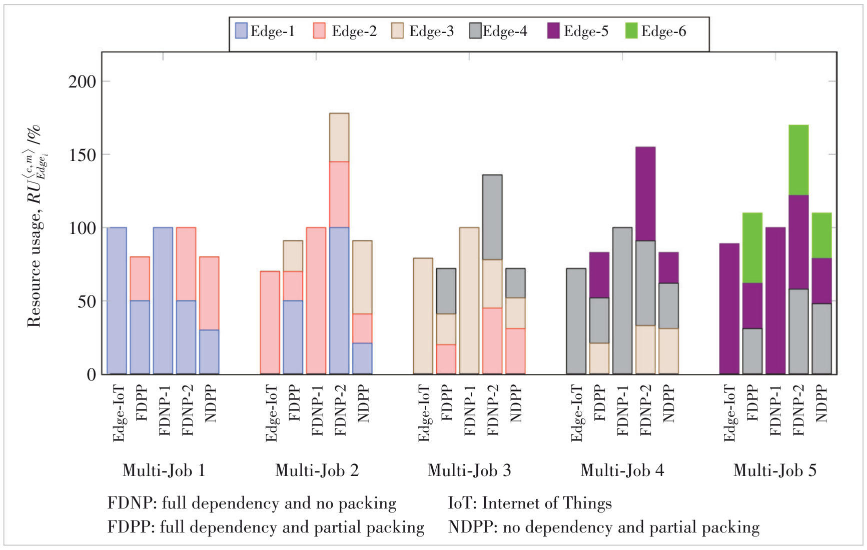
1) Resource usage and resource utilization

Fig. 4 shows the task deployment ratio of Edge-IoT with four baseline schemes. It can be seen that for each multi-job task offloaded, Edge-IoT is able to deploy its constituent tasks to a single edge. This is because Edge-IoT selects the closest edge with sufficient resource availability to accommodate all the tasks, and co-locates them tightly in each node. Recall that some of the baseline schemes, i.e., FDNP-1 and FDNP-2, do not co-locate tasks on each node, but assume each node can only execute one task at a time. Therefore, FDNP-1 can neither offload all its subtasks nor execute them at a time, given the number of nodes at each edge. For example, Multi-Job 1 that consists of five jobs is deployed and co-located on edge Edge-1 by Edge-IoT, and in turn, allows for faster input data flow transmissions. For the same Multi-Job 1, FDPP, FDNP-2 and NDPP deploy the jobs across two edge deployments. Although FDPP and NDPP can partially co-locate tasks at each of the edges, the three schemes incur additional execution delays due to input data flow transmissions across the two edge



FDNP: full dependency and no packing
FDPP: full dependency and partial packing
IoT: Internet of Things
NDPP: no dependency and partial packing

▲Figure 4. Tasks deployment ratio across the edge deployments

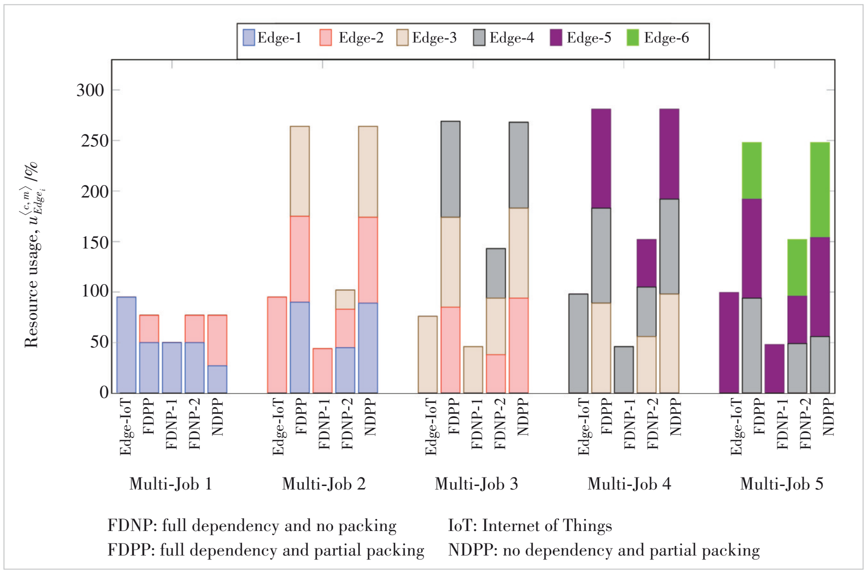


FDNP: full dependency and no packing
FDPP: full dependency and partial packing
IoT: Internet of Things
NDPP: no dependency and partial packing

▲Figure 5. Average resource usage across the edge deployments

deployments. On the other hand, FDNP-1 is not able to deploy all the jobs on edge Edge-1, because it executes a task on each node at a time. Hence, it can only execute several tasks at a time, given the number of nodes available in the edge cluster, while the remaining tasks wait in a queue. Fig. 5 shows the average resource usage of the multi-job tasks deployed by Edge-IoT with those of the four baseline schemes across the edge clusters. It can be seen that Edge-IoT consumes the fewest resources by using a single edge for each multi-job task, while FDNP-2 uses the highest resources (up to three edge deployments) for the same multi-job task. The average resource utilization comparison is shown in Fig. 6. Again, Edge-IoT achieves the highest resource utilization compared with the four baseline schemes. We now examine the performance of Edge-IoT compared with the baseline schemes for each multi-job offloaded (as shown in Table 3) in detail.

- Multi-Job 1: Edge-IoT dispatches 100% of the tasks in a single-hop offloading to Edge-1. It first optimizes the deployment by gang-scheduling and co-locating as many tasks in a node as possible to fully utilize the available resources in the node. These tasks are tightly packed on nodes using the packing algorithm, which uses all of Edge-1 resources to execute the tasks, and achieves 95% resource utilization. For the same Multi-Job 1, some of the baseline schemes such as FDPP, FDNP-2 and NDPP offload the tasks across two edge clusters (Edge-1 and Edge-2), using up to two times more resources than Edge-IoT. FDNP-1 schedules one task on a node at a time using a single edge deployment (Edge-1). Thus, it uses all available resources (100%) at the edge deployment and keeps the unscheduled tasks on a task queue until resources become available. Overall, Edge-IoT achieves better resource usage and utilization compared to the four baseline schemes, as shown in Figs. 5 and 6.
- Multi-Job 2: This multi-job task consists of seven jobs with a total of 29 tasks, where each job has a task dependency in the range of (1, 5]. Edge-IoT optimizes the deployment to ensure that the resources are fully utilized. Containers provide isolation to running applications, making it possible to co-locate multiple applications on the same node without any interference. A single container-optimized node can execute more containerized applications, given that there are sufficient available resources. For scheduling, Edge-IoT deploys all the tasks at a time on edge cluster Edge-2, using 70% of the resources, while with three edge deployments, FDPP, FDNP-2 and NDPP use 50%, 20% and 21% on Edge-1, 100%, 45% and 33% on Edge-2, and 21%, 20% and 50% on Edge-3. Edge-IoT and FDNP-1 utilize 95% and 55% of resources, respectively. Although FDNP-1 uses all available resources in the cluster, it achieves low resource utilization due to its inability to co-locate tasks on nodes, which results in resource under-utilization. Again Edge-IoT outperforms all the four baseline schemes in terms of task deployment ratio, resource usage and utilization.
- Multi-Job 3: Edge-IoT offloads all tasks of Multi-Job 3 to edge Edge-3. This edge deployment is made up of six Huawei AR502H Series edge devices, with CPU and memory capacity of 24 vCPU and 12 GiB, respectively. The multi-job task consists of nine jobs, with a total of 38 tasks, where each job has a task dependency range (1, 8]. Edge-IoT improves resource usage by using a single edge and up to three times fewer resources compared with the four baseline schemes, as can be seen from Fig. 5. It also achieves 76% resource utilization in a single cluster. On the other hand, with three edge deployments, FDPP and NDPP achieve 85% and 89% resource utilization on Edge-2; 94% and 94% on Edge-3; and 89% and 85% on Edge-4). FDNP-1 and FDNP-2 perform worst with the highest resource consumption and the lowest resource utilization.
- Multi-Job 4 and Multi-Job 5: These multi-job tasks are offloaded by Edge-IoT to Edge-4 and Edge-5, respectively. Among all the schemes, Edge-IoT uses the least resources for each multi-job execution across the two edge clusters. Specifically, Edge-IoT consumes 72% and 89% of resources at Edge-4 and Edge-5, respectively. It also achieves the highest resource utilization of 98% and 99% across the two clusters, compared to the four baseline schemes. FDPP consumes 21%, 31% and 31% of resources across Edge-3, Edge-4 and Edge-5, and NDPP consumes 31%, 31% and 21% ofresources across Edge-4, Edge-4 and Edge-6. FDNP-1 consumes all available resources at Edge-3 and Edge-4 for Multi-Job4 and Multi-Job5, respectively, while recording the lowest resource utilization at each cluster. FDNP-2 consumes the second highest resources and achieves the



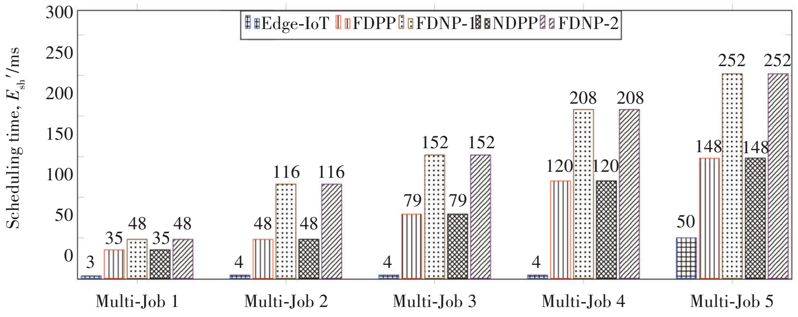
FDNP: full dependency and no packing
FDPP: full dependency and partial packing
IoT: Internet of Things
NDPP: no dependency and partial packing

▲Figure 6. Average resource utilization across the edge deployments second lowest resource utilization for the same multi-job task execution.

2) Multi-Task Scheduling, Execution and Response Time

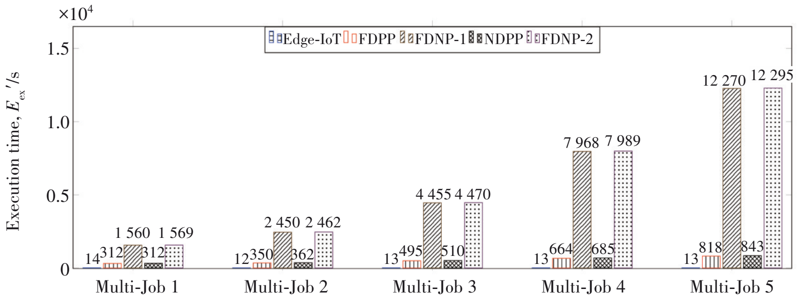
The aggregate job scheduling time E_{sh}' defined in Eq. (9), which is the time for placing multi-job tasks on the nodes in a cluster, is an important performance metric to assess the integrated edge clusters. Another important performance metric is the aggregate job execution time E_{ex}' defined in Eq. (10). The response time E_{rsp}'defined in Eq. (8) is even more important. Figs. 7, 8 and 9 compare the scheduling time, execution time and response time, respectively, attained by the five schemes.

It can be seen that the scheduling time is typically very small, and the execution time and response time by contrast are significantly larger. Across the edge clusters, Edge-IoT consistently achieves the fastest scheduling, execution and response, compared to the other four benchmark strategies. Note that we focus on the scheduling time, execution time and result transmission time components of the response time. This is because the offloading time E_{of}' is relatively small due to our offloading policy which ensures that jobs are offloaded to the closest edge cluster and within a single-hop offloading. Specifically, for Multi-Job 1, Edge-IoT achieves a very fast scheduling, which is 11.6 times faster than FDPP and NDPP, and 16 times faster than FDNP-1 and FDNP-2. For Multi-Job 2 scheduling, Edge-IoT achieves significantly shorter scheduling time than the four benchmark strategies, i. e., Edge-IoT is 12 times faster than FDPP and NDPP, and 29 times faster than FDNP-1 and FDNP-2. For Multi-Job 3, FDNP-1 and FDNP-2 attain the lowest scheduling time, while FDPP and NDPP attain the second lowest scheduling time. Edge-IoT achieves the best performance with up to 38 times faster than the other four schemes. For Multi-Job 4 and Multi-Job 5, Edge-IoT again achieves the fastest scheduling, followed by FDPP and NDPP, while FDNP-1 and FDNP-2 have the worst scheduling performance.



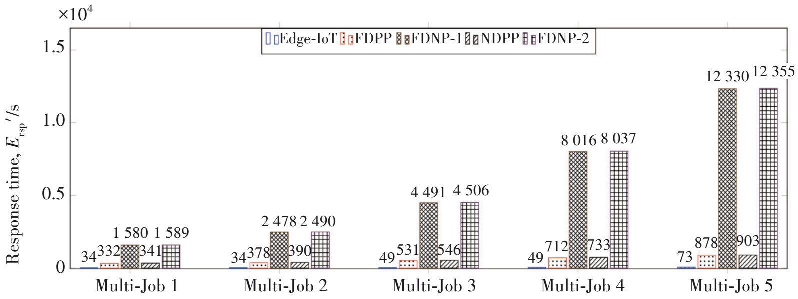
FDNP: full dependency and no packing IoT: Internet of Things
FDPP: full dependency and partial packing NDPP: no dependency and partial packing

▲Figure 7. Task scheduling time across edge deployments



FDNP: full dependency and no packing IoT: Internet of Things
FDPP: full dependency and partial packing NDPP: no dependency and partial packing

▲Figure 8. Task execution time across edge deployments



FDNP: full dependency and no packing IoT: Internet of Things
FDPP: full dependency and partial packing NDPP: no dependency and partial packing

▲Figure 9. Task response time across edge deployments

In terms of the execution time, it is important to note that the input data flow time also contributes to the total execution time of a job. FDPP, FDNP-2 and NDPP incur additional time due to their approaches of task offloading across multiple clusters, which leads to input data flows (which is in the range of (0.2, 0.4] s) across the clusters. Edge-IoT is 111.4, 22.3, 112 and 23 times faster than FDNP-1, FDPP, FDNP-2 and NDPP, respectively, for executing Multi-Job 1, while for Multi-Job 2 execution, it is approximately 204, 29, 205 and 30 times faster, respectively. Similarly, for Multi-Job 3, Multi-Job 4 and Multi-Job 5 executions, Edge-IoT achieves approximately up to 943.8, 63, 945.7 and 64.8 times shorter execution time than FDNP-1, FDPP, FDNP-2 and NDPP, respectively. The significant advantage of Edge-IoT in terms of the aggregate job execution time can be explained as follows. It deploys sets of multi-job tasks as a unit through the gang scheduling strategy in a single-edge deployment. These applications are deployed and executed concurrently. By contrast, the benchmark approaches schedule and execute the given DAGs individually and in parts across multiple edge deployments, resulting in input data flow transmission delays and longer time to execute the overall tasks.

Recall that the response time of a job defined in Eq. (8) is the addition of its offloading time, scheduling time, execution time and final result transmission time. Therefore, the ultimate aim is to minimize the response time of IoV applications offloaded to EC. Fig. 9 compares the response time of Edge-IoT and the four benchmark schemes. Edge-IoT outperforms the four benchmark schemes by achieving shorter response time for all the multi-job tasks, and up to 169, 12, 169.2 and 12.4 times faster than FDNP-1, FDPP, FDNP-2 and NDPP, respectively.

6 Conclusions

Edge-IoT, a machine learning-enabled IoT application orchestration in an EC system proposed in this paper, has demonstrated superior QoS in resource management and IoT multi-task orchestration in edge clusters. Unlike Edge-IoT, the existing methods do not deploy all the ready tasks at a time or in a single edge cluster or do not respect task dependencies, leading to more edge resource usage and cluster under-utilization as well as causing longer task execution time. This paper has presented Edge-IoT to improve edge resource efficiency and performance. We have utilized a resource-aware offloading strategy that selects the closest edge cluster suitable for a given job, and a container-based bin packing optimization strategy that packs or co-locates tasks tightly on nodes to fully utilize available resources. To evaluate our approach, we have illustrated use cases of real-world CPU and memory-intensive tasks from Alibaba cluster trace, which records the activities of both long-running containers (for Alibaba's e-commerce business) and batch jobs across eight days. We have compared our approach with the state-of-the-art dependency-aware IoV task orchestration baseline strategies. Our proposed algorithm achieves both the highest edge cluster resource utilization and the minimum scheduling, execution and response time for IoV multi-job tasks compared to the baseline strategies. The gains achieved by Edge-IoT as observed from our experiments include faster response time of the overall tasks and improved usage of edge resources.

References

[1] KHAN L U, YAQOOB I, TRAN N H, et al. Edge-computing-enabled smart cities: a comprehensive survey [J]. IEEE Internet of Things journal, 2020, 7(10): 10200 – 10232. DOI: 10.1109/JIOT.2020.2987070

[2] AMIN S U, HOSSAIN M S. Edge intelligence and Internet of Things in healthcare: a survey [J]. IEEE access, 2020, 9: 45 – 59. DOI: 10.1109/ACCESS.2020.3045115

[3] LIU Y J, WANG S G, ZHAO Q L, et al. Dependency-aware task scheduling in vehicular edge computing [J]. IEEE Internet of Things journal, 2020, 7(6): 4961 – 4971. DOI: 10.1109/JIOT.2020.2972041

[4] SHEN Q Q, HU B J, XIA E J. Dependency-aware task offloading and service caching in vehicular edge computing [J]. IEEE transactions on vehicular technology, 2022, 71(12): 13182 – 13197. DOI: 10.1109/TVT.2022.3196544

[5] REN H, LIU K, JIN F, et al. Dependency-aware task offloading via end-edge-cloud cooperation in heterogeneous vehicular networks [C]//25th International Conference on Intelligent Transportation Systems (ITSC). IEEE, 2022: 1420 – 1426. DOI: 10.1109/ITSC55140.2022.9922334.

[6] LIU S S, LIU L K, TANG J, et al. Edge computing for autonomous driving: opportunities and challenges [J]. Proceedings of the IEEE, 2019, 107(8): 1697 – 1716. DOI: 10.1109/jproc.2019.2915983

[7] MAHMUD R, TOOSI A N, RAMAMOHANARAO K, et al. Context-aware placement of industry 4.0 applications in fog computing environments [J]. IEEE transactions on industrial informatics, 2020, 16(11): 7004 – 7013. DOI: 10.1109/TII.2019.2952412

[8] OTHMAN M M, EL-MOUSA A. Internet of Things & cloud computing Internet of Things as a service approach [C]//11th International Conference on Information and Communication Systems (ICICS). IEEE, 2020: 318 – 323. DOI: 10.1109/ICICS49469.2020.239503

[9] REN J, ZHANG D Y, HE S W, et al. A survey on end-edge-cloud orchestrated network computing paradigms: transparent computing, mobile edge computing, fog computing, and cloudlet [J]. ACM computing surveys, 2020, 52(6): 1 – 36. DOI: 10.1145/3362031

[10] HWANG J, NKENYEREYE L, SUNG N, et al. IoT service slicing and task offloading for edge computing [J]. IEEE Internet of Things journal, 2021, 8 (14): 11526 – 11547. DOI: 10.1109/jiot.2021.3052498

[11] ALMUTAIRI J, ALDOSSARY M. A novel approach for IoT tasks offloading in edge-cloud environments [J]. Journal of cloud computing, 2021, 10(1): 1 – 19. DOI: 10.1186/s13677-021-00243-9

[12] AWADA U, ZHANG J K, CHEN S, et al. Air-to-air collaborative learning: a multi-task orchestration in federated aerial computing [C]//14th International Conference on Cloud Computing (CLOUD). IEEE, 2021: 671 – 680. DOI: 10.1109/CLOUD53861.2021.00086

[13] AWADA U, ZHANG J K, CHEN S, et al. AirEdge: a dependency-aware multi-task orchestration in federated aerial computing [J]. IEEE transactions on vehicular technology, 2022, 71(1): 805 – 819. DOI: 10.1109/TVT.2021.3127011

[14] TU Y F, DONG Z J, YANG H Z. Key Technologies and application of edge computing [J]. ZTE communications, 2017, 15(2): 26-34. DOI: 10.3969/j.issn.1673-5188.2017.02.004

[15] LI X W, ZHAO L, YU K P, et al. A cooperative resource allocation model for IoT applications in mobile edge computing [J]. Computer communications, 2021, 173: 183 – 191. DOI: 10.1016/j.comcom.2021.04.005

[16] LI J, LIANG W F, XU W Z, et al. Maximizing user service satisfaction for delay-sensitive IoT applications in edge computing [J]. IEEE transactions on parallel and distributed systems, 2022, 33(5): 1199 – 1212. DOI: 10.1109/TPDS.2021.3107137

[17] ZHAN C, HU H, LIU Z, et al. Multi-UAV-enabled mobile-edge computing for time-constrained IoT applications [J]. IEEE Internet of Things journal, 2021, 8 (20): 15553 – 15567. DOI: 10.1109/JIOT.2021.3073208

[18] LI J, LIANG W F, XU W Z, et al. Service home identification of multiple-source IoT applications in edge computing [J]. IEEE transactions on services computing, 2023, 16(2): 1417 – 1430. DOI: 10.1109/TSC.2022.3176576

[19] LIU J L, LIU C H, WANG B, et al. Optimized task allocation for IoT application in mobile-edge computing [J]. IEEE Internet of Things journal, 2022, 9 (13): 10370 – 10381. DOI: 10.1109/JIOT.2021.3091599

[20] HAN S N, LI X H, SUN C, et al. RecCac: Recommendation-empowered cooperative edge caching for internet of things [J]. ZTE communications, 2021, 19 (2): 2 – 10. DOI: 10.12142/ZTECOM.202102002

[21] LIU C H, LIU K, GUO S T, et al. Adaptive offloading for time-critical tasks in heterogeneous Internet of vehicles [J]. IEEE Internet of Things journal, 2020, 7 (9): 7999 – 8011. DOI: 10.1109/JIOT.2020.2997720

[22] RAMPERSAUD S, GROSU D. Sharing-aware online virtual machine packing

in heterogeneous resource clouds [J]. IEEE transactions on parallel and distributed systems, 2017, 28(7): 2046–2059. DOI: 10.1109/TPDS.2016.2641937

[23] HONG Z C, CHEN W H, HUANG H W, et al. Multi-hop cooperative computation offloading for industrial IoT-edge-cloud computing environments [J]. IEEE transactions on parallel and distributed systems, 2019, 30(12): 2759–2774. DOI: 10.1109/TPDS.2019.2926979

Biographies

Uchechukwu AWADA is currently working toward a PhD degree at the School of Information Engineering, Zhengzhou University, China. His current research interests include edge computing, cloud computing, aerial computing, distributed systems, IoT, IoV and wireless communications. He is a student member of the ACM.

ZHANG Jiankang (jzhang3@bournemouth.ac.uk) is a senior lecturer at Bournemouth University, UK. Prior to joining Bournemouth University, he was a senior research fellow at the University of Southampton, UK. Dr. ZHANG was a lecturer from 2012 to 2013 and then an associate professor from 2013 to 2014 at Zhengzhou University, China. His research interests are in the areas of aeronautical communications, aeronautical networks, evolutionary algorithms and edge computing.

CHEN Sheng received his BE degree from the East China Petroleum Institute, China in 1982 and his PhD degree from City, University of London, UK in 1986, both in control engineering. In 2005, he was awarded the higher doctoral degree, Doctor of Sciences (DSc), from the University of Southampton, UK. From 1986 to 1999, He held research and academic appointments at the Universities of Sheffield, Edinburgh and Portsmouth, all in UK. Since 1999, he has been with the School of Electronics and Computer Science, the University of Southampton, where he holds the post of Professor in Intelligent Systems and Signal Processing. His research interests include adaptive signal processing, wireless communications, modeling and identification of nonlinear systems, neural network and machine learning, intelligent control system design, and evolutionary computation methods and optimization. He has published over 600 research papers. He has 18 500+ Web of Science citations with h-index of 59, and 36 700+ Google Scholar citations with h-index of 81. Dr. CHEN is a Fellow of the United Kingdom Royal Academy of Engineering, a Fellow of the Asia-Pacific Artificial Intelligence Association, and a Fellow of IET. He is one of the original ISI' s highly cited researchers in engineering (March 2004). He is named a 2023 Electronics and Electrical Engineering Leader in the UK by Research.com.

LI Shuangzhi received his BS and PhD degrees from the School of Information Engineering, Zhengzhou University, China in 2012 and 2018, respectively. From 2015 to 2017, he was a visiting student with the Department of Electrical and Computer Engineering, McMaster University, Canada. He is currently a lecturer with the School of Information Engineering, Zhengzhou University, China. His research interests include noncoherent space-time coding and ultra-reliable low-latency communications.

YANG Shouyi received his PhD degree from the Beijing Institute of Technology, China in 2002. He is currently a full professor with the School of Information Engineering, Zhengzhou University, China. He has authored or co-authored various articles in the field of signal processing and wireless communications. His current research interests include signal processing in communications systems, wireless communications, and cognitive radio.