Journal of Systems Architecture 141 (2023) 102923

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

Resource-aware multi-task offloading and dependency-aware scheduling for integrated edge-enabled IoV

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

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

ARTICLE INFO

Keywords:
Edge computing
IoV
Dependency-aware
Execution time
Resource efficiency
Co-location

ABSTRACT

Internet of Vehicles (IoV) enables a wealth of modern vehicular applications, such as pedestrian detection, real-time video analytics, etc., that can help to improve traffic efficiency and driving safety. However, these applications impose significant resource demands on the in-vehicle resource-constrained Edge Computing (EC) device installation. In this article, we study the problem of resource-aware offloading of these computation-intensive applications to the *Closest* roadside units (RSUs) or telecommunication base stations (BSs), where on-site EC devices with larger resource capacities are deployed, and mobility of vehicles are considered at the same time. Specifically, we propose an Integrated EC framework, which can keep edge resources running across various in-vehicles, RSUs and BSs in a single pool, such that these resources can be holistically monitored from a single control plane (CP). Through the CP, individual in-vehicle, RSU or BS edge resource availability can be obtained, hence applications can be offloaded concerning their resource demands. This approach can avoid execution delays due to resource unavailability or insufficient resource availability at any EC deployment. This research further extends the state-of-the-art by providing intelligent multi-task scheduling, by considering both task dependencies and heterogeneous resource demands at the same time. To achieve this, we propose **FedEdge**, a variant Bin-Packing optimization approach through *Gang-Scheduling* of multi-dependent tasks that *co-schedules* and *co-locates* multi-task tightly on nodes to fully utilize available resources. Extensive experiments on real-world data trace from the recent Alibaba cluster trace, with information on task dependencies and resource demands, show the effectiveness, faster executions, and resource efficiency of our approach compared to the existing approaches.

1. Introduction

Internet of Vehicles (IoV) is a network of modern smart vehicles aiming to connect and exchange data over the Internet. It is expected that IoV will be an enabler for Future Mobility [1]. This technology supports various kinds of communication patterns, for example, Vehicle-to-Vehicle (V2V), Vehicle-to-Infrastructure (V2I), and Vehicle-to-Sensor (V2S), it also enables a series of intelligent applications, such as personalized navigation, autonomous driving, online AR/VR gaming, etc., which commonly require significant amount of computing resources and often latency-sensitive [2,3]. To this end, Edge Computing (EC) [4,5], a new distributed computing paradigm that brings computation and data storage closer to the location where it is needed [6–8], to improve response times, has been recently introduced to efficiently execute latency-sensitive IoV applications [9–21]. Consequently, automakers have begun installing portable EC devices into vehicles to enable in-vehicle application execution and other complex resource-hungry use cases. In doing so, data transfer costs would not only be reduced significantly but would also enable vehicles to respond and adapt instantly in real time.

However, the resource constraints of the in-vehicle EC deployment may be insufficient to fulfill some computation-intensive application demands. For example, suppose a vehicle needs to execute some applications concurrently, where each application is structured in a microservice architectural style, consisting of a large number of

* Corresponding authors.
E-mail addresses: awada@gs.zzu.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).
URLs: https://orcid.org/0000-0002-2300-0586 (U. Awada), https://orcid.org/0000-0001-5316-1711 (J. Zhang), https://orcid.org/0000-0001-6882-600X (S. Chen), https://orcid.org/0000-0002-2801-5779 (S. Li), https://orcid.org/0000-0002-5149-5280 (S. Yang).

https://doi.org/10.1016/j.sysarc.2023.102923
Received 31 July 2022; Received in revised form 14 May 2023; Accepted 4 June 2023
Available online 9 June 2023

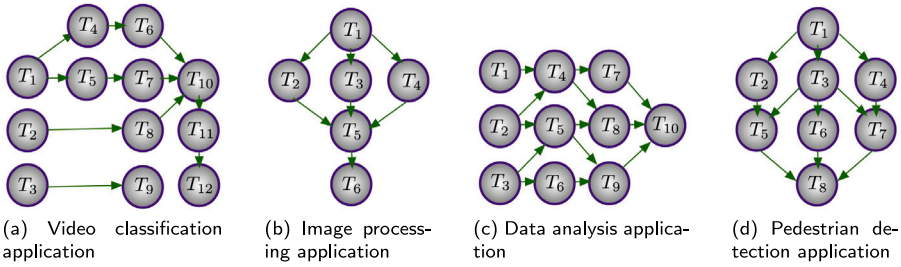

Fig. 1. DAG of representative applications.

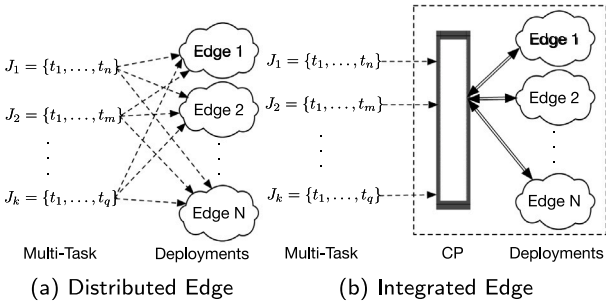

Fig. 2. (a) An example of random multi-tasks dispatching without considering their resource demand, dependencies and cluster resource status, and (b) An example of intelligent multi-tasks dispatching, where tasks resource demand, dependencies and cluster resource status are considered.

inter-dependent applications, requiring substantial resources for execution and are latency-sensitive, as shown in Fig. 1. The in-vehicle EC deployment may not have the resource capacity or availability to quickly execute such latency-sensitive applications. Although some existing schemes try to refuse, postpone, or queue these requests on the EC device, such approaches are not suitable for latency-sensitive applications. Hence, IoV systems have recently begun offloading such computation-intensive applications as follows; (i) to another vehicle with sufficient EC resource availability via vehicle-to-vehicle (V2V) communications [10,20], and (ii) to roadside units (RSUs) or telecommunication base stations (BSs), where on-site EC devices with larger resource capacities are deployed, via vehicle-to-infrastructure (V2I) communications [9,11–19,21]. After completing the executions, the results are immediately and deterministically communicated back to the vehicle. Nevertheless, to randomly offload applications to any available Vehicle, RSU or BS, as shown in Fig. 2(a), without any respect to their resource demands, inter-application dependencies and the Vehicle, RSU or BS resource availability, can also result in execution delay and performance degradation. For example, in Fig. 2(a), Job J_1 consists of inter-dependent tasks $t_1, \ldots, t_n$, where each task t_i is randomly dispatched or offloaded to a different EC deployment. Such a random and individual approach would incur a significant delay in the job's execution time. The reason for such delay is that each dependent task dispatched to a different EC deployment would require the execution result(s) or input data from other tasks to be transmitted back to its host EC deployment to complete its execution. This transfer of input data is referred to as an *input data flow*, and such transmission would incur additional delay, thereby further affecting the job's execution time given the rate and number of transmissions that could occur. Therefore, one fundamental problem is where and how to offload and schedule computation/latency-intensive multi-dependent tasks in such diverse EC deployments so that their collective execution time and response time are minimized, as well as optimal utilization of the EC resources is achieved.

For this reason, we wish to consider an approach which seamlessly integrates all participating edge resources running across N deployments (i.e., in-vehicles, RSUs, BSs, etc.) in a single pool as shown in Fig. 2(b), such that these resources can be holistically monitored from a control plane (CP), and multi-tasks can be effectively dispatched dynamically across the resources. This approach is called *Edge Federation* (EF) [22–28]. Recently introduced edge computing frameworks, i.e., KubeEdge,[1] MicroK8s,[2] etc, have the capabilities of integrating edge resources running across various platforms, run containerized tasks and eliminate provider lock-in situations. The EF setup consists of a `host` and `members`. Therefore, given N independent edge deployments, the CP is set up in one of the deployments regarded as the `host`, while the remaining $N-1$ deployments are regarded as the `members`, which can be added or removed from the CP. The EF system can be given as:

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

Through the CP, resource availability status, as well as running tasks status can be obtained from all the deployments (`host` and `members`), thus enabling informed decisions on optimal multi-task dispatching. EF can play as an agent for information dissemination through data caching and synchronization, without individual vehicle fetching from backhauls. Another benefit that EF brings is minimized latency by serving vehicles from the EC deployment closest to them [22–25]. The work presented in this paper differs considerably from prior works [23,24], which addressed the problem of multi-dependent tasks orchestration in a federated autonomous drone-enabled edge computing system, while considering the drones' flight time to avoid loss of jobs [29]. For example, in Fig. 2(b), all the tasks of Job J_1 can be dispatched to the same EC deployment with sufficient resource availability, thereby enabling the inter-dependent tasks to communicate and share input flow data quickly for faster execution. In addition, moving vehicles can offload their tasks to an edge deployment closest to them at their initial locations, then the results can be communicated back to the vehicles through the closest edge at their current locations [19–21,30]. However, given the current location of each vehicle, more than one routing path may exist for each vehicle. Therefore, the routing path with the best transmission performance can be determined as the optimal one for the final execution result transmission. To this end, we propose an EF-assisted routing (EFR), in which the goal is to find the fastest route to efficiently forward execution results to each vehicle at its current location. Specifically, our EFR leverages the cooperation among the participating EC deployments, i.e., `host` and `members` to quickly forward the execution results to the target vehicles. Furthermore, our EFR also adopts a forwarding policy: **direct single copy**, in which the results can only be transmitted from the source EC deployment to the vehicles without caching at the intermediate EC deployment(s), thus enabling faster transmission time.

[1] https://kubeedge.io/en/.
[2] https://microk8s.io/.

Table 1
Offloading orders and scheduling units of various schemes.

Scheme	Offloading order	Scheduling units
FedEdge	$\{T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8, T_9, T_{10}, T_{11}, T_{12}\} \rightarrow Edge_1$	1
FDPP	$\{T_1, T_2, T_3\} \rightarrow Edge_1$; $\{T_4, T_5, T_6, T_7\} \rightarrow Edge_2$; $\{T_8, T_9, T_{10}, T_{11}, T_{12}\} \rightarrow Edge_3$	3
FDNP-1	$\{T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_4 \rightarrow T_5 \rightarrow T_6 \rightarrow T_7 \rightarrow T_8 \rightarrow T_9 \rightarrow T_{10} \rightarrow T_{11} \rightarrow T_{12}\} \rightarrow Edge_1$	12
FDNP-2	$\{\{T_1, T_2\} \rightarrow \{T_3, T_4\} \rightarrow \{T_5, T_6\} \rightarrow \{T_7, T_8\} \rightarrow \{T_9, T_{10}\} \rightarrow \{T_{11}, T_{12}\}\} \rightarrow Edge_1, \ldots, Edge_6$	6
Random	$\{T\}_{i=1}^{12} \rightarrow Edge_i$	12

1.1. Challenge and motivation

Having dispatched inter-dependent applications concerning their resource demands to suitable EC deployment, one challenging issue is how to schedule such inter-dependent applications, to further minimize their collective scheduling time, and execution time and achieve high edge resource utilization. To address this problem, we further propose an intelligent multi-task scheduling approach called **FedEdge**, which considers both task dependencies and heterogeneous resource demands at the same time. FedEdge is a variant Bin-Packing optimization approach in which *Gang-Schedules* [31] multi-dependent tasks by *co-scheduling* and *co-locating* them tightly on nodes to fully utilize available resources. To avoid interference and resource contention among co-located tasks, we provide isolation to co-located tasks through containerization [22,32]. Containerization provides isolation to running tasks and enables tasks to be executed in any edge deployment regardless of the architecture or provider.

Task dependency awareness is critical for achieving optimal performance in cloud/EC dispatching and scheduling problems [33–37]. In Fig. 1, we show an example of multi-dependent tasks, where some of the tasks require input data from other tasks to complete their execution. Again, in Fig. 1(a), we have tasks T_1, T_2 and T_3, regarded as independent tasks, i.e., they do not depend or require input data flow from other tasks, and they can start execution once there are offloaded to EC deployment. Tasks T_4 and T_5 require input data from T_1 to enable them to complete their execution. Similarly, tasks T_6, T_7 and T_8 depend on the completion of task T_4, T_5 and T_2, respectively. Dispatching these tasks to the same edge would enable faster input data flow, thereby enabling faster execution compared to individual task execution across different edge clusters, as shown in Fig. 2(a). These complex inter-task dependencies with heterogeneous resource demands and diverse edge deployments with heterogeneous resource capacities make task offloading and resource management in edge/cloud computing a non-trivial task. Considering such demands and resource capacities is necessary to achieve effective task offloading and scheduling, ultimately to achieve optimal performance [38,39]. Hence, a key objective of our FedEdge is to reduce the collective execution time of such tasks and to improve cluster resource usage by considering the inter-task dependencies.

Therefore, given the n multi-dependent tasks $T_1, T_2, \ldots, T_n$, as shown in Fig. 1(a), FedEdge adopts the gang-scheduling [31] strategy and a variant bin-packing optimization to efficiently co-schedule and co-locate them in a cluster or edge. We consider FedEdge as a Full Dependency and Full Packing (FDFP) approach. Thus, the scheduling time can be expressed as:

$$\sum_{z=1}^{m} \sum_{i=1}^{k_z} E_{sh_{z_i}} / k_z, \tag{2}$$

where m is the number of scheduling units, k_z is the number of tasks within the zth scheduling unit having the tasks $\{T_{z_1}, T_{z_2}, \ldots, T_{z_{k_z}}\}$.

We illustrate the advantage of the offloading and scheduling approach in FedEdge over three other existing schemes as follows; (i) an approach which executes some tasks of a job locally in the vehicle, offloads some tasks to the cloud server and some to the RSU for execution at the same time. We refer to this approach as Full Dependency and Partial Packing (FDPP), and it is similar to the approach in [40]. However, this approach incurs additional waiting time or execution delay through the input data flows among the vehicle, RSU and Cloud. For example, assuming the video classification application in Fig. 1(a) is to be executed, this approach executes tasks T_1, T_2 and T_3 locally in the vehicles, dispatches tasks T_4, T_5, T_6 and T_7 to RSU, and dispatches the remaining tasks T_8, T_9, T_{10}, T_{11} and T_{12} to the Cloud. Since these tasks are inter-dependent tasks, it means that the execution result of task T_1 need to be transmitted from the vehicle to the RSU, to serve as 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 the RSU to the Cloud, to serve as input data to task T_{10}. Finally, the execution results of tasks T_2 and T_3 need to be transmitted from the vehicle to the Cloud to complete the job's execution. Unlike the FedEdge's scheduling approach which gang schedules all tasks of a job tightly on the same EC deployment, FDPP can only schedule subtasks dispatched to the vehicle, RSU and Cloud individually; (ii) an approach which offloads all tasks of a job to the same EC deployment, but assumes that at any EC deployment, only one node can execute one task at a time. Therefore, all tasks must wait in a queue until resources become available for the next task. Such a queue is constructed based on the application priority, where it keeps multiple applications in decreasing order of their priority. We refer to this approach as Full Dependency and No Packing (FDNP-1), and it is similar to the approach in [37]; (iii) an approach which offloads different subtasks of a job to different EC deployment, where each node at the EC deployment can only execute one task at a time and the task with the highest priority is first selected for scheduling. We consider this approach as a Full Dependency and No Packing (FDNP-2), and it is similar to the approach in [41]. Note that this approach would also incur additional waiting time or execution delay due to the input data flow from one EC deployment to another; (iv) and finally the Random approach, which offloads tasks randomly to any available EC deployment, and it does not consider both tasks' dependencies and task co-location. We refer to this approach as No Dependency and No Packing (NDNP). It is important to note that delay in dispatching and scheduling inter-dependent tasks directly impacts their response time. For the multi-dependent tasks of Fig. 1(a) with $n = 12$ tasks, Table 1 lists the scheduling orders and scheduling units for the schemes compared. FedEdge only needs one offloading and scheduling unit ($m = 1$) which has $k_1 = 12$ tasks and it also achieves the lowest execution time of $\frac{1}{k}\sum_{i=1}^{k} E_{ex_i}$. By contrast, Random has $m = 12$ offloading and scheduling units, each having a single task. Hence it has the highest execution time of $\sum_{i=1}^{m} E_{ex_i}$. Thus, FedEdge achieves the lowest scheduling and execution time. Note that FDPP, FDNP-1 and FDNP-2 dispatch and schedule individual or subsets of the tasks at a time.

1.2. Novelty and contribution

Motivated by the challenges introduced in Sections 1 and 1.1, in this paper, we study a novel multi-task dispatching and scheduling approach under an integrated EC-enabled IoV framework, where smart vehicles can offload their computation-intensive applications to the closest edge deployment, concerning tasks resource demand, thus enabling faster response time, as well as efficient resource utilization. Major novelties and contributions are summarized below:

- We investigate a situation whereby multiple resource running across various edge deployments (i.e., Vehicles, RUSs, BS, etc.) are integrated through a single control plane (CP), such that each edge deployment resource status can be obtained and utilized for an effective IoV application dispatching.

- Once these applications are dispatched, we introduce an intelligent multi-dependent tasks scheduling strategy that co-schedules through a gang-scheduling approach.
- To guarantee optimal usage of cluster resources, we further propose a variant bin-packing optimization approach that co-locates tasks firmly on available nodes, to avoid resource wastage.
- We further propose an EF-assisted routing, though the direct single copy (DSC) policy to efficiently forward the application's execution result to the target vehicle at its current location.
- We show that our approach is capable of minimizing the actual completion time of multi-dependent IoV tasks using minimum resources and minimizing the response time of IoV applications. We conduct extensive experiments to compare the performance of our approach with several existing approaches using real-world data-trace from Alibaba cluster data,[3] which provides information on task dependencies and resource demands.

1.2.1. Paper organization

The remaining parts of our paper are structured as follows. Section 2 presents related works on EC-enabled IoV resource allocation schemes. In Sections 3 and 4, we detail our proposed FedEdge for achieving high computation resource utilization and minimizing the response times of IoV applications deployed on integrated EC resources. In Section 5, we compare the performance of our proposed FedEdge against those of several existing schemes through extensive experiments. Finally, we conclude the paper in Section 6.

2. Related works

EC can efficiently augment a smart vehicle's performance by executing its complex and computation-intensive applications in its in-vehicle EC installation or by offloading them to the closest EC resources deployed at another vehicle, RSUs, BSs, etc. The objective of this work is to study the task offloading problem in IoV. This study aims to avoid naive offloading and scheduling strategies, which could result in resource wastage due to resource underutilization, delay due to insufficient resource availability, queuing, etc., at the edge clusters. It also aims to improve the quality of user experience through efficient task dispatching and faster execution, thereby enlarging the overall system capacity and benefits.

There is a huge number of existing works that have addressed numerous task-offloading problems in IoVs. For example, Liwang et al. [9] introduced a futures-based resource trading approach in EC-enabled IoV (EC-IoV), where a forward contract is used to facilitate resource trading-related negotiations between an EC deployment and a vehicle in a given future term. They formulated these futures-based resource trading as an optimization problem aiming to maximize the vehicle's and the EC deployment's expected utility. Chen et al. [10] proposed a distributed multi-hop task offloading decision model for task execution efficiency, which consists of two main parts: a candidate vehicle selection mechanism for screening the neighboring vehicles that can participate in offloading, and a task offloading decision algorithm design part for obtaining the task offloading solution. Zhang et al. [11] presented a V2V task offloading scheme, by jointly incorporating the budget constraint into the system design and then, they proposed a joint deep reinforcement learning (DRL) approach combined with the convex optimization algorithm to solve the problem. Sun et al. [12] proposed a resource allocation approach based on Federated Learning (FL), where they have tackled both the related delay and energy consumption challenges. Mu et al. [13] proposed a time-optimized, multi-task offloading model adopting the principles of Optimal Stopping Theory (OST) to maximize the probability of offloading to the optimal EC servers. Lu et al. [14] investigated how to analytically design an analytical offloading strategy for a multi-user mobile EC-based smart IoV, where there are multiple computational access points (CAPs) which can help compute tasks from the vehicular users. Ning et al. [15] constructed an energy-efficient scheduling framework for EC-enabled IoV to minimize the energy consumption of RSUs under task latency constraints, by jointly considering task scheduling among EC servers and downlink energy consumption of RSUs. In [16], the authors aim at the task offloading system of the IoV and considered the situation of multiple EC servers, and proposed a dynamic task offloading scheme based on deep reinforcement learning (DRL). Similarly, in [17], intending to address the dynamic computation offloading problem in IoV, the authors constructed a Markov decision process (MDP), and then they utilized the twin delayed deep deterministic policy gradient (TD3) algorithm to achieve the optimal offloading strategy. Pliatsios et al. [18] considered a network consisting of multiple vehicles connected to EC-enabled RSUs and proposed an approach that minimizes the total energy consumption of the system by jointly optimizing the task offloading decision, the allocation of power and bandwidth, and the assignment of tasks to EC-enabled RSUs. In [19] the authors proposed a latency-aware service migration method with decision theory, by modeling the network architecture, the transmission and the computation of the service requests. Then, they considered the high mobility of vehicles and analyzed the dynamic change of vehicle locations in other to transform the service migration problem into an uncertain decision optimization problem. Their main objective is to minimize the service latency and balance the workload on RSUs. Alam et al. [20] developed a cooperative three-layer generic decentralized vehicle-assisted EC (VEC) network, where vehicles in associated RSU and neighbor RSUs are in the bottom fog layer, EC servers are in the middle cloudlet layer, and cloud in the top layer. Then, they employed a multi-agent DRL-based Hungarian algorithm (MADRLHA) to solve dynamic task offloading in VEC. In [21], the authors aim at minimizing the total priority-weighted task processing delay for all the devices by offloading the tasks to the EC server or other vehicles, allocating the wireless channels of the BS, and allocating the computing resource of the EC server and the vehicles. Wu et al. [42] constructed an authentication scheme for edge computing-enabled IoV with drone assistance, which keeps the anonymity of vehicles during data transmissions. They aim to resist forgery attacks in the main process. Fontes et al. [43] reviewed recent studies that use artificial intelligence (AI) algorithms to schedule data offloading and manage distributed computational resources in mobile edge computing (MEC)-Enabled IoV Networks.

The above methods leverage EC to offload smart vehicle applications, they promise traffic efficiency and driving safety but lack the consideration of joint optimization of task resource demands and EC deployment resource status. Consequently, a resource-aware computation offloading strategy is proposed in this paper. On the other hand, it is also important to avoid any form of resource wastage, i.e., resource underutilization, because efficiently managing edge resources directly dictates service quality and performance [44]. 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 requirement of the co-running tasks, to determine how many of them can be co-located. To this end, a tasks co-location mechanism was proposed in [45], where it shows that by accurately estimating the resource level needed, a co-location scheme can effectively determine how many tasks can co-locate on the same host to improve the system throughput, by taking into consideration the memory and CPU requirements of co-running tasks. Intending to maximize resource utilization, the authors of [46] utilized reinforcement learning to co-locate interactive services with batched ML workloads. Our previous works [38,39] 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 [23–28] which

[3] https://github.com/alibaba/clusterdata.

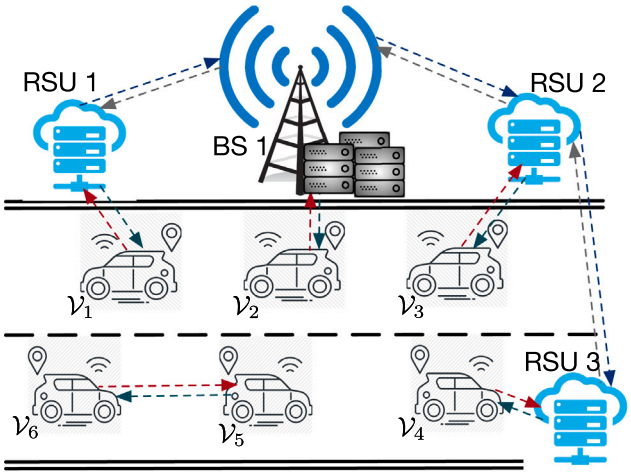

Fig. 3. An example architecture of dynamic integrated edge-enabled IoV multi-task offloading.

unifies distributed edge resources, such that they can be holistically managed. Our previous works [22–24] proposed a dependency-aware task scheduling in such a unified system, and illustrates how modern applications are usually structured with inter-task dependencies, where a task depends on the output from another task (s). A huge number of existing works, i.e., [33–37,47–49] have also 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.

Different from the aforementioned works, in this article, we consider multiple inter-dependent applications and propose a fine-grained offloading strategy to maximize the overall IoV system efficiency, which jointly considers tasks dependencies, unified service entities and distributed edge resources, such that they can be holistically managed from a single control plan (CP), where resource status information can be utilized for effective multi-task dispatching, co-scheduling and co-location. Our research extends existing schemes by proposing resource-aware multi-task dispatching in a unified EC-enabled IoV system, which includes parked and moving vehicles, all service entities and resources running across various EC platforms (i.e., in-vehicle EC deployment, RSUs, BS, etc.). We further propose a variant bin-packing optimization approach through gang-scheduling of multi-dependent tasks, which quickly co-schedules and co-locates tasks firmly on available nodes, to avoid resource wastage. We also propose an EF-assisted routing strategy for faster execution result transmission. We finally show that our scheme is capable of minimizing the response time of IoV multi-dependent tasks using minimum EC resources through extensive experiments and comparison.

3. System model and problem formulation

In this section, we introduce our proposed FedEdge for achieving high resource utilization and faster response times for IoV applications deployed on federated edge resources, then the problem formulation and algorithm framework.

3.1. System model

We consider an urban vehicular network environment where IoV applications are offloaded from vehicles to integrated EC deployments across various vehicles, RSUs, BSs, etc. Each vehicle is equipped with a powerful wireless interface that can be used to connect with other vehicles, RSUs, BSs, etc. Our work focuses on V2V and V2I offloading as illustrated in Fig. 3. Note that these EC deployments are made up of heterogeneous resources. For example, the in-vehicle EC deployment may not be as large as those of the RSUs, while those of the RSUs may not be as large as the BSs, etc, in terms of resource capacity. The most important feature of EC is the ability to provide storage and computing resources closer to where it is needed so that applications can process data and respond within a minimum time. One of the advantages of using an integrated edge system is the ability to cache and synchronize core data across all participating EC deployments, such that the same high efficiency can be achieved wherever it is needed. For example, a smart vehicle moving within a road segment at a constant speed v should be able to access the **closest** EC deployment and react immediately to changing road conditions, without first offloading its core data which could lead to increased latency. Therefore, data caching and synchronization is the best-fit solution for IoVs to fully exploit EC. For applications with small data sizes, it is possible to package the applications and database in containers,[4] and then to deploy it to the **closest** EC deployment whenever it is needed. For such applications, let $\langle \delta, c, m \rangle$ represent the size of data input, CPU and memory requirements, respectively.

Let $\mathbb{EF} = \{Edge_1, \ldots, Edge_M\}$ represent the integrated or federated EC deployments, where each individual participating EC deployment $Edge_i$ (i.e., vehicle, RSU, BS, etc.) is a cluster of container-instances i.e., edge device(s) with virtualized container-optimized nodes. Let $C_{Edge_i}^{\langle \delta,c,m \rangle}$ represent the resource availability of each participating edge $Edge_i$, and $\mathcal{L} = \{x_i, y_i\}$ to represent its location coordinate. Through the control panel (CP) of the $\mathbb{EF}$, $C_{Edge_i}^{\langle \delta,c,m \rangle}$ and $\mathcal{L}$ of each edge can be obtained to make an informed decision on multi-task offloading or dispatching. Let $\mathbb{V} = \{\mathcal{V}_1, \ldots, \mathcal{V}_M\}$ represent the index set of vehicles. We consider both parked and moving vehicles [17,19,20]. At time t, the location coordinate $\mathcal{L}_q(t)$ of a moving vehicle $\mathcal{V}_q$ is given as $\{x_q, y_q\}$, then at $t > 0$ the location coordinate $\mathcal{L}_q(t > 0)$ changes to $\{x_{q'}, y_{q'}\}$. The vehicle $\mathcal{V}_q$ can choose to execute its ready application locally in its in-vehicle EC device installation if there is sufficient resource availability or offload it to the closest EC deployment $Edge_{i^\star} \in \mathbb{EF}$ with sufficient resource availability. Let $\vartheta\left[\mathcal{V}_q(t)\right]$ denote the offloading decision variable, which is measured by

$$\vartheta\left[\mathcal{V}_q(t)\right] = \begin{cases} 1, & \text{tasks are offloaded,} \\ 0, & \text{tasks are processed locally.} \end{cases} \tag{3}$$

At time t, while $\vartheta\left[\mathcal{V}_q(t)\right] = 0$, the multi-task $\mathbb{C} \in \mathcal{V}_q$ is decided to perform local execution procedure in the vehicle $\mathcal{V}_q$, otherwise while $\vartheta\left[\mathcal{V}_q(t)\right] = 1$, $\mathbb{C} \in \mathcal{V}_q$ is to be offloaded to the EC deployment $(Edge_{i^\star})$ with sufficient resources closest to $\mathcal{V}_q$. A task $T \in \mathbb{C}$ can be dependent on other tasks (i.e., microservice or inter-dependent application, as shown in Fig. 1). Each task T has three resource requirements: storage, CPU and memory, as the total amount of resources needed for its execution, denoted as $d_T^{\langle \delta,c,m \rangle}$. For simplicity, we will focus on the CPU and memory requirements/capacity of all tasks and resources. That is, the storage is sufficient for the size of data input δ, and hence the requirement $\langle \delta, c, m \rangle$ is simplified as $\langle c, m \rangle$. 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:

$$E_{ex} = E_{cp} - E_{st}. \tag{4}$$

Recall that $\mathbb{EF}$ enables core data to be cached and synchronized across all participating EC deployments, thereby eliminating the need for core data transmission delay. However, existing offloading strategies (i.e., [40,41], etc.) allow subtasks of a job to be offloaded separately across different EC deployments, thus creating additional delay in the application's response time, as explained in Section 1.1. For example, when a vehicle in such an approach begins to offload its tasks, the delay includes three parts: (1) the time to offload subtasks from

[4] Application containers (i.e., Docker [32]) provide isolation, portability and lightweight tasks offloading solution from devices to edge clusters.

the vehicle to different EC deployments, given as E_{of}, (2) the time to transmit results of executed subtasks from one EC deployment to another EC deployment, given as E_{sub}, and (3) the time to transmit 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:

$$E_{rsp} = \sum_{T \in \mathbb{C}} \left(E_{of} + E_{sub} + E_{sh} + E_{ex} \right) + E_{rst}. \tag{5}$$

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

$$\mathbb{C} \Rightarrow Edge_{\star}, \tag{6}$$

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

$$E_{rsp} = E_{of} + \sum_{T \in \mathbb{C}} \left(E_{sh} + E_{ex} \right) + E_{rst}. \tag{7}$$

Note that for a parked vehicle, the final execution result is also transmitted back to the vehicle in a single-hop along the reverse offloading path. However, for a moving vehicle, a single-hop transmission of the final execution result might not be possible due to the mobility or change in location of the vehicle. Hence, an EF-assisted routing (EFR) to quickly forward the execution results to the vehicle at its current location is further proposed in this paper, by determining the optimal route for the transmissions to minimize E_{rst}. Recall that at time t, while $\vartheta\left[\mathcal{V}_q(t)\right] = 1$, it is decided that a vehicle $\mathcal{V}_q$ (i.e., parked vehicle or moving vehicle) is to offload its multi-task $\mathbb{C}$ to the closest edge $Edge_{\star}$ with sufficient resource availability. Hence, the coordinates of the vehicle's location $\mathcal{L} = \{x_q, y_q\}$, and the coordinates of all surrounding EC deployments, i.e., vehicles, RUSs, BSs, etc, are used to compute the distances matrix $\mathbb{DM}$, to select the closest edge with sufficient resource availability. $\mathbb{DM}$ is the distance between a vehicle and surrounding EC deployments. It can be obtained using the Manhattan Distance,[5] in which the distance between two locations $\mathcal{L}_z = \{x_z, y_z\}$ and $\mathcal{L}_j = \{x_j, y_j\}$ is given as:

$$dist^{\mathcal{L}_{z,j}} = |x_z - x_j| + |y_z - y_j|. \tag{8}$$

Thus, the distance matrix between a vehicle $\mathcal{V}_q$ and a set of M surrounding EC deployments is given as;

$$\mathbb{DM} = \begin{bmatrix} dist^{\mathcal{L}_{\mathcal{V}_q, Edge_1}} \\ dist^{\mathcal{L}_{\mathcal{V}_q, Edge_2}} \\ \vdots \\ dist^{\mathcal{L}_{\mathcal{V}_q, Edge_M}} \end{bmatrix}. \tag{9}$$

Therefore, the **closest** EC deployment is defined as

$$Edge_{\star} = \arg \min_{Edge_i \in \mathbb{EF}} \left(dist^{\mathcal{L}_{\mathcal{V}_q, Edge_i}} \right), \forall_{c,m}. \tag{10}$$

Once $\mathbb{C}$ has been dispatched to $Edge_{\star}$, FedEdge utilizes the gang-scheduling [31] strategy which co-schedules all ready applications at a time. Given a cluster of container instances or nodes $I_i \in Edge_{\star}$, let $I^{\langle c,m \rangle}_{Edge_{\star}}$ denote each node's resource capacity or availability. In a real scenario where multi-vehicle $\mathcal{V} \in \mathbb{V}$ offload multi-tasks with multi-dependency at t, these applications are dispatched as a multi-Job $\mathbb{J}$, i.e., $\mathbb{J} \Rightarrow Edge_{\star}$, where each Job J is a collection of each vehicle's multi-tasks, with collective resource demand denoted as $\sum_{i=1}^{k} d^{\langle c,m \rangle}_{T_i} = d^{\langle c,m \rangle\prime}_{T}$. Hence, we can dispatch $\mathbb{J}$ to $Edge_{\star}$ with suitable resource availability. Therefore, the aggregate scheduling time and execution time of a multi-job $\mathbb{J}$ is given as:

$$\sum_{J \in \mathbb{J}} \sum_{i=1}^{k} \frac{E_{sh_i}}{k} = E_{sh}\prime, \tag{11}$$

and

$$\sum_{J \in \mathbb{J}} \sum_{i=1}^{k} \frac{E_{ex_i}}{k} = E_{ex}\prime, \tag{12}$$

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

$$\mathcal{U}^{\langle c,m \rangle}_{Edge_i} = \frac{\sum_{J \in \mathbb{J}} d^{\langle c,m \rangle\prime}_{T}}{C^{\langle c,m \rangle}_{Edge_i}}. \tag{13}$$

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

$$\mathcal{U}^{\langle c \rangle}_{Edge_i} = \frac{\sum_{J \in \mathbb{J}} d^{\langle c \rangle\prime}_{T}}{C^{\langle c \rangle}_{Edge_i}}, \tag{14}$$

$$\mathcal{U}^{\langle m \rangle}_{Edge_i} = \frac{\sum_{J \in \mathbb{J}} d^{\langle m \rangle\prime}_{T}}{C^{\langle m \rangle}_{Edge_i}}, \tag{15}$$

where $\sum_{J \in \mathbb{J}} d^{\langle c \rangle\prime}_{T}$ and $\sum_{J \in \mathbb{J}} d^{\langle m \rangle\prime}_{T}$ are the total collective CPU and memory demands, respectively. After completing the multi-job executions, the final execution results are immediately and deterministically transmitted to the target vehicles at their current locations. Since there could be multiple routes or multi-hop transmissions to individual vehicles, we have leveraged the cooperation among the stationary EC deployments within the $\mathbb{EF}$, i.e., parked vehicles, RSUs, BSs, etc, and proposed an EF-assisted routing (EFR) to quickly forward the execution results to each vehicle by determining the optimal route for the transmissions. We also adopt a forwarding policy: **direct single copy (DSC)**, where the results can be transmitted from the host EC deployment to the vehicles without the intermediate EC deployment(s) keeping a copy. This policy guarantees faster transmission compared to other types of policies that make and keeps a copy of each result passing through the intermediate EC deployment(s). Although, DSC caches the original results only at the host EC deployment such that they can be transmitted to any other vehicle(s) upon request. Therefore, given a multi-Job $\mathbb{J} = J_1, \ldots, J_N$, executed at $Edge_{\star}$, the set of final execution results $\mathbb{R} = \mathcal{R}_1, \ldots, \mathcal{R}_N$ is destined for a set of vehicles $\mathbb{V} = \mathcal{V}_1, \ldots, \mathcal{V}_N$ at different location coordinates. For any result $\mathcal{R}_q$ to be transmitted from $Edge_{\star}$ to a vehicle $\mathcal{V}_q$ at location $\mathcal{L}_q(t > 0) = \{x_{q'}, y_{q'}\}$, there are assumed to be a maximum of K available routing paths between them. The number of hops of the kth routing path between $Edge_{\star}$ and target $\mathcal{V}_q$ is represented by $H_{Edge_{\star}, \mathcal{V}_q}$. Let $\mathcal{P}_k = \{Edge_{\star}, Edge_i, Edge_j, \ldots, \mathcal{V}_q\}$ be a sequence denoting the kth transmission path from $Edge_{\star}$ to target $\mathcal{V}_q$, the path is composed of segments denoted by defining a function:

$$l(\mathcal{P}_k) = \{(Edge_{\star}, Edge_i), (Edge_i, Edge_j), \ldots, (Edge_z, \mathcal{V}_q)\}, \tag{16}$$

where an ordered pair $a = (i, j)$, is the segment from i to j, $a[0] = i$, and $a[1] = j$. For each transmission path $\mathcal{P}_i \in K$, we first compute the total distance, given as:

$$dist^{total}_{\mathcal{P}_k} = \sum_{i=1}^{H_{Edge_{\star}, \mathcal{V}_q}} \sum_{j=i+1}^{H_{Edge_{\star}, \mathcal{V}_q}+1} dist^{\left(Edge_i, Edge_j\right)}. \tag{17}$$

our EFR aims to determine the optimal transmission path $\mathcal{OR}_i$ for each moving vehicle $\mathcal{V} \in \mathbb{V}$ and to apply the DSC policy, such that E_{rst} is minimized, namely:

$$\mathcal{OR}_i = \arg \min_{\mathcal{P}_i \in K} \left(dist^{total}_{\mathcal{P}_i} \right). \tag{18}$$

3.2. Problem formulation

The basic notations adopted are described in Table 2. The objectives are to minimize the response time, E_{rsp} of (7) for all $J \in \mathbb{J}$ and to maximize the computation or cluster resource utilization $\mathcal{U}^{\langle c,m \rangle}_{Edge_i}$ of (13), subject to certain constraints. The response time E_{rsp} of (7) comprises

[5] https://en.wikipedia.org/wiki/Taxicab_geometry.

Table 2
Notations.

Notation	Description
$\mathbb{EF}$	Integrated edge deployments
T	Individual application or task
$\langle \delta, c, m \rangle$	Storage, CPU and memory resources
$\mathbb{C}$	A set of containerized applications
$\mathbb{DM}$	Distance matrix
$d_T^{\langle c,m \rangle}$	Application or task resource requirements
$Edge_i$	Individual edge deployment or cluster
$Edge_\star$	Closest edge deployment or cluster
I_i	Container-instance or node in a cluster
$I_i^{\langle c,m \rangle}$	Resource capacity or availability of a node
$C_{Edge_i}^{\langle c,m \rangle}$	Resource capacity/availability in an edge
$U_{Edge_i}^{\langle c,m \rangle}$	Resources used for execution
$U_{Edge_i}^{\langle c \rangle}$, $U_{Edge_i}^{\langle m \rangle}$	CPU, memory resource used for execution
$RU_{Edge_i}^{\langle c,m \rangle}$	Actual resources usage of jobs
$RU_{Edge_i}^{\langle c \rangle}$, $RU_{Edge_i}^{\langle m \rangle}$	Actual CPU, memory resources usage
E_{st}, E_{cp}	Application/task start, completion time
E_{ex}	Application or task execution time
$dist^{\mathcal{L}_{i,j}}$	Distance between locations i and j
$\mathcal{U}_{Edge_i}^{\langle c,m \rangle}$	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
$\mathcal{V}$, $\mathbb{V}$	A Vehicle, A set of Vehicles

of the dispatching or offloading time E_{of}, in which the *Closest* and *gang-single-hop computation offloading* policies are jointly adopted, thus enabling faster offloading time; the scheduling time $E_{sh}\prime$ of (11); the execution time $E_{ex}\prime$ of (12); and the final execution results transmission time E_{rst}.

Constraints

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

$$\sum_{J \in \mathbb{J}} d_T^{\langle c,m \rangle \prime} \leq C_{Edge_\star}^{\langle c,m \rangle}, \quad \forall_{c,m}. \tag{19}$$

Second, unused or inactive nodes $I_i \in Edge_\star$ 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 is currently considered for allocation or has at least a job that is started, executed or completed. An *Inactive* node is a node that is not running and is not currently considered for allocation and not having at least a job that is being started, executed or completed. These two states can be expressed as follows:

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

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

Optimization formulation

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

$$\textbf{Maximize} \quad \mathcal{U}_{Edge_i}^{\langle c,m \rangle} = \frac{\sum_{J \in \mathbb{J}} d_T^{\langle c,m \rangle \prime}}{C_{Edge_i}^{\langle c,m \rangle}}, \tag{21}$$

$$subject\ to \quad \mathbb{J} \Rightarrow Edge_\star, \quad \exists, \tag{22}$$

$$\beta\left(I_i\right) \in \{0, 1\}, \quad \exists, \tag{23}$$

$$\sum_{J \in \mathbb{J}} d_T^{\langle c,m \rangle \prime} \leq C_{Edge_\star}^{\langle c,m \rangle}, \quad \forall_{c,m}. \tag{24}$$

The constraints (22) to (24) indicate the dispatching of multi-job $\mathbb{J}$ to the closest edge having sufficient resource capability or availability. More specifically, (22) is the $\mathbb{J}$ dispatching constraint, guaranteeing that $\mathbb{J}$ is dispatched to a cluster, such that dependent tasks within each $J \in \mathbb{J}$ can communicate and execute faster. The condition (23) guarantees that active nodes ($\beta\left(I_i\right) = 1$) should be used for execution, and inactive nodes ($\beta\left(I_i\right) = 0$) should be shut down. The constraint (24) guarantees that $d_T^{\langle c,m \rangle \prime}$ of $\mathbb{J}$ should not exceed $C_{Edge_i}^{\langle c,m \rangle}$ of any selected cluster. We shall discuss the details of our multi-job dispatching principle in Section 4.1 and Algorithm 1. Hence, we aim 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 Section 4.2 and Algorithm 2. On the other hand, the overall scheduling time and execution time can be minimized depending on orchestration:

$$\textbf{Minimize} \quad \sum_{J \in \mathbb{J}} \sum_{i=1}^{k} \frac{E_{sh_i}}{k} = E_{sh}\prime, \tag{25}$$

$$subject\ to \quad \mathbb{J} \Rightarrow Edge_\star, \quad \forall_{c,m}, \tag{26}$$

and

$$\textbf{Minimize} \quad \sum_{J \in \mathbb{J}} \sum_{i=1}^{k} \frac{E_{ex_i}}{k} = E_{ex}\prime, \tag{27}$$

$$subject\ to \quad \mathbb{J} \Rightarrow Edge_\star, \quad \forall_{c,m}. \tag{28}$$

The constraint (26) and (28) guarantees 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. Finally, the result transmission time for a moving vehicle can be minimized depending on routing:

$$\textbf{Minimize} \quad \sum_{J \in \mathbb{J}} E_{rst}, \tag{29}$$

$$subject\ to \quad \mathcal{OR}_i = \arg\min_{\mathcal{P}_i \in K} \left(dist_{\mathcal{P}_i}^{total}\right). \tag{30}$$

The constraint (30) guarantees that the shortest route is selected for the result transmission. The details of our result transmission policy are given in Section 4.3 and Algorithm 3

4. FedEdge algorithm framework

Our FedEdge solution is composed of three parts: offloading or dispatching, scheduling and result transmission (note that result transmission is only applied to a moving vehicle). The dispatching strategy is based on the orchestration of ready multi-task to the closest EC deployment with sufficient available resources to accommodate the tasks, as expressed in Eq. (10), while the scheduling strategy involves packing or co-location of these tasks tightly on container-instances to fully utilize the available resources. These three components aim at providing optimal performance for vehicular multi-task execution in an integrated EC system, such that the optimizations in Eqs. (11), (12) and (13) are achieved. FedEdge is developed considering the following policies:

1. Selecting the EC deployment with sufficient available resources and the least distance (i.e., single hop) to any vehicle ready to offload its multi-task. This is referred to as the Closest policy.
2. Dispatching these multi-tasks as a unit to the Closest EC deployment. This is referred to as the Single-hop Offloading policy.
3. After the multi-task execution, the execution result(s) is transmitted back to the vehicle. If the vehicle's location remains the same (i.e., a parked vehicle), the result(s) is transmitted back through a single-hop transmission, but if its location changes (i.e., a moving vehicle), the result(s) is transmitted by determining the optimal transmission path. In this case, FedEdge applies

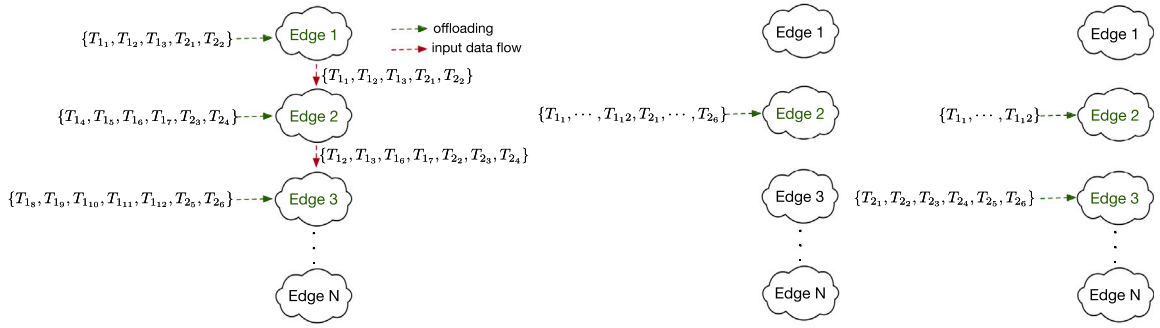

Fig. 4. Application dispatching strategies.

the direct single copy (DSC) policy, in which the result(s) is transmitted from the host EC deployment to the vehicle without caching at the intermediate EC deployment(s) within its path.

4.1. Offloading or dispatching policy

When sets of vehicular multi-tasks $\mathbb{J} = J_1, \ldots, J_N$ are ready to be offloaded, our policy is to dispatch them to the **closest** edge $Edge_\star$ with the sufficient resource capacity or availability, i.e., $\sum_{J\in\mathbb{J}} d_T^{\langle c,m\rangle\prime} \leq C_{Edge_i}^{\langle c,m\rangle}$. For the rationale of this strategy, consider Ericsson Connected Vehicle Platform[6] (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 multi-tasks i.e., $\vartheta[\mathcal{V} \in \mathbb{V}] = 1$, we would see a total load of 4000 requests. Executing these loads would require an EC 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 a unit to the closest EC deployment, i.e., $\mathbb{J} \Rightarrow Edge_\star$, having sufficient resource capacity or availability. The *closest* heuristic given in Equation Eq. (10) is to minimize the offloading time E_{of} and to further minimize the overall response time E_{rsp} 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. Existing works, e.g., [23,24,39], 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 1 describes the dispatching procedure.

With the collective resource demand of $\mathbb{J}$, each $Edge_i \in \mathbb{EF}$ resource status $C_{Edge_i}^{\langle c,m\rangle}$ is obtained from the CP, the dispatcher selects the closest having matching resources through the $\mathbb{DM}$ (line 2). It dispatches the $\mathbb{J}$ to $Edge_\star$ (line 3). If $\mathbb{J}$ cannot be dispatched to $Edge_\star$, then $\mathbb{J}$ is dispatched to the next $Edge_\star$ (line 5). If the collective $\mathbb{J}$ are greater than any $Edge_i \in \mathbb{EF}$, i.e., $\sum_{J\in\mathbb{J}} d_T^{\langle c,m\rangle\prime} > C_{Edge_i}^{\langle c,m\rangle}$, $\forall Edge_i \in \mathbb{EF}$, then $\mathbb{J}$ cannot be dispatched as a whole. The dispatcher can allow fractional dispatching of each $J \in \mathbb{J}$ to closest `member` (line 8∼18). Note that fractional dispatching of each $J \in \mathbb{J}$ to closest `member` would still allow inter-dependent tasks within each $J \in \mathbb{J}$ to execute faster.

To illustrate the effectiveness of our dispatching policy, we give a practical example. Suppose each sub-application or task $T_1, \ldots, T_n$ of the applications in Fig. 1(a) and (b) is randomly offloaded or dispatched to different EC deployments, then each dependent task would require the input data from other tasks to be transmitted back to its host edge deployment to complete its execution, as shown in Fig. 4(a). This transfer of input data 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 and image processing applications in Fig. 1(a) and (b), respectively, are to be executed, the work in [40] proposed an approach which offloads the tasks across multiple edge deployments, i.e., T_{1_1}, T_{1_2}, T_{1_3}, T_{2_1} and T_{2_2} to Edge 1, offloads tasks T_{1_4}, T_{1_5}, T_{1_6}, T_{1_7}, T_{2_3} and T_{2_4} to Edge 2, and offloads the remaining tasks T_{1_8}, T_{1_9}, $T_{1_{10}}$, $T_{1_{11}}$, $T_{1_{12}}$, T_{2_5} and T_{2_5} to Edge 3. Since these tasks are inter-dependent tasks, it means that the execution results of tasks T_{1_1} and T_{2_1} need to be transmitted from Edge 1 to Edge 2, to serve as input data to tasks T_{1_4}, T_{1_5}, T_{2_3} and T_{2_4}, while the execution results of tasks T_{1_6} and T_{1_7}; T_{2_2}, T_{2_3} and T_{2_4} need to be transmitted from edge Edge 2 to edge Edge 3, to serve as input data to tasks T_{10} and T_{2_5}, respectively. Finally, the execution results of tasks T_{1_2} and T_{1_3} need to be transmitted from edge Edge 1 to edge Edge 3 to complete the video classification application execution. This approach increases the time complexity and overall latency of the applications. To eliminate the need for input data flow, our dispatching policy dispatches the two applications as a multi-job $\mathbb{J}$ to a suitable edge having required resource availability, i.e., $\mathbb{J}$ is dispatched to Edge 2, as shown in Fig. 4(b). This approach would enable faster response time by enabling faster interactions among the dependent tasks, thus

Algorithm 1 FedEdge: Multi-Job Dispatching Policy

Input: $\mathbb{J}$ arrived at time t; $Edge_i \in \mathbb{EF}$; $\sum_{J\in\mathbb{J}} d_T^{\langle c,m\rangle\prime}$
Output: Dispatch $\mathbb{J}$ to $Edge_\star$ with matching $C_{Edge_\star}^{\langle c,m\rangle}$, such that $\mathbb{J} \Rightarrow Edge_\star$

1: **for** $Edge_i \in \mathbb{EF}$ **do**
2: **if** $\sum_{J\in\mathbb{J}} d_T^{\langle c,m\rangle\prime} \leq C_{Edge_i}^{\langle c,m\rangle}$ and $Edge_i = \text{argmin}_{Edge_i\in\mathbb{EF}}\left(dist^{\mathcal{L}_{\mathcal{V}\in\mathbb{V},Edge_i}}\right)$ **then**
3: $\mathbb{J} \Rightarrow Edge_i = Edge_\star$
4: **break**
5: **else**
6: Dispatch $\mathbb{J}$ to next $Edge_\star$
7: **end if**
8: **end for**
9: **if** $\mathbb{J}$ cannot be dispatched as a whole **then**
10: **for** $Edge_i \in \mathbb{EF}$ **do**
11: **for** $J \in \mathbb{J}$ **do**
12: **if** $d_T^{\langle c,m\rangle\prime} \leq C_{Edge_i}^{\langle c,m\rangle}$ and $Edge_i = \text{argmin}_{Edge_i\in\mathbb{EF}}\left(dist^{\mathcal{L}_{\mathcal{V}_i,Edge_i}}\right)$ **then**
13: $J \Rightarrow Edge_i = Edge_\star$
14: **break**
15: **else**
16: Dispatch J to next $Edge_\star$
17: **end if**
18: **end for**
19: **end for**
20: **end if**

[6] https://www.ericsson.com/en/connected-vehicles/platform.

Algorithm 2 FedEdge: Multi-job Co-location

Input: $\mathbb{J}$ dispatched to closest member $Edge_\star$, resource demand of each $J \in \mathbb{J}$: $d_T^{\langle c,m\rangle\prime}$, 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** $RU_{Edge_\star}^{\langle c,m\rangle}$

1: **for** $I_i \in Edge_\star$ **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 $d_T^{\langle c,m\rangle\prime} \le I_i^{\langle c,m\rangle}$ **then**
6: $J \Rightarrow I_i$
7: $\Gamma[J, I_i] = 1$
8: $I_i^{\langle c,m\rangle} = I_i^{\langle c,m\rangle} - d_T^{\langle c,m\rangle\prime}$
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**

Algorithm 3 FedEdge: Final Result Transmission

Input: The current locations of each vehicle $\{x_q\prime, y_q\prime\}$ and speed s, and the Distance matrix $\mathbb{DM}$
Output: The optimal transmission route $\mathcal{OR}_q$, such that **Minimize** E_{rst_q}

1: **for** $\mathcal{V}_q \in \mathbb{V}$ **do**
2: Compute the number of transmission paths K to $\mathcal{V}_q$
3: **for** $\mathcal{P}_q \in K$ **do**
4: Compute the total number of hops $\left[H_{Edge_\star,\mathcal{V}_q}\right]$
5: Compute the total distance $\left[dist_{\mathcal{P}_q}^{total}\right]$
6: **end for**
7: Compute and select the path that minimizes E_{rst}
8: i.e., $\mathcal{OR}_q = \text{argmin}_{\mathcal{P}_q \in K}\left[dist_{\mathcal{P}_q}^{total}\right]$
9: **end for**

enabling faster execution time. Nevertheless, if the edge deployments cannot accommodate $\mathbb{J}$ due to insufficient resource availability, each $J \in \mathbb{J}$ can be dispatched disjointly, i.e., J_1 can be dispatched to Edge 2, while J_2 can be dispatched to Edge 3, given there is sufficient resource availability at Edge 2 and Edge 3 to accommodate J_1 and J_2, respectively, as shown in Fig. 4(c). This fractional dispatching can also avoid input data flow, and enable faster interactions among the dependent tasks, thus enabling faster execution time.

4.2. Scheduling policy

Once $\mathbb{J}$ is dispatched 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 $d_T^{\langle c,m\rangle\prime}$ of each $J \in \mathbb{J}$ to provide efficient co-location, such that fewer container-instances or nodes are used for execution in $Edge_\star$. Specifically, the gang scheduling [31] 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 IoV task offloading in EC approaches, such as [37,41]which assumes that each node at the selected EC deployment can only execute one task at a time, would result in massive resource under-utilization and wastage. For example, the applications executed in Fig. 4 which contains a total of 26 tasks, will require 26 nodes for execution in such approaches, whereas in our Algorithm 2 which describes our scheduling strategy, can co-locate multi-dependent tasks firmly on nodes, such that for any given jobs, resource wastage is avoided and fewer nodes are used for execution. It takes the resource demand of multi-task/job 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). 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 (line 4~9). Hence, for every $\mathbb{J}$ offloaded to $Edge_\star$, our co-location strategy is to find the solution to the problem:

$$\textbf{Minimize} \sum_{I_i \in Edge_\star} I_i \equiv \textbf{Minimize} RU_{Edge_\star}^{\langle c,m\rangle} = \frac{U_{Edge_\star}^{\langle c,m\rangle}}{C_{Edge_\star}^{\langle c,m\rangle}}, \tag{31}$$

$$subject\ to\ \mathbb{J} \Rightarrow Edge_\star,\ \exists, \tag{32}$$

$$\sum_{J \in \mathbb{J}} \Gamma[J,\ I_i] \cdot d_T^{\langle c,m\rangle\prime} \le I_i^{\langle c,m\rangle}, \quad \forall c, m, \tag{33}$$

where

$$\Gamma[J,\ I_i] = \begin{cases} 1, & \text{if } J \Rightarrow I_i, \\ 0, & \text{otherwise.} \end{cases} \tag{34}$$

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

$$RU_{Edge_\star}^{\langle c\rangle} = \frac{U_{Edge_\star}^{\langle c\rangle}}{C_{Edge_\star}^{\langle c\rangle}}, \tag{35}$$

$$RU_{Edge_\star}^{\langle m\rangle} = \frac{U_{Edge_\star}^{\langle m\rangle}}{C_{Edge_\star}^{\langle m\rangle}}, \tag{36}$$

where $U_{Edge_\star}^{\langle c\rangle}$ and $U_{Edge_\star}^{\langle m\rangle}$ are the used CPU and memory resources, respectively, while $C_{Edge_\star}^{\langle c\rangle}$ and $C_{Edge_\star}^{\langle m\rangle}$ are the edge's CPU and memory resource capacity, respectively. Therefore, the constraint in Eq. (32) is the multi-job $\mathbb{J}$ deployment constraint, guaranteeing that $\mathbb{J}$ are dispatched 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 can allow fractional dispatching of each $J \in \mathbb{J}$ to the closest member edge. The constraint in Eq. (33) indicates that the total estimated resource requirements of co-located jobs $d_T^{\langle c,m\rangle\prime}$ cannot exceed $I_i^{\langle c,m\rangle}$, the node resource availability. The condition in Eq. (34) means that if job J_i is placed on the 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.

Note that in modern systems, the dispatcher and scheduler need to understand the characteristics of both the applications (i.e., dependencies, resource requirements, etc.) 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, the offloading and scheduling policies should be jointly designed to achieve the desired goals. For example, our scheduling policy which adopted bin-packing optimization can easily adapt to other types of offloading policies. However, if these offloading policies do not consider the dependencies among tasks, resource requirements and edge resource availability, the overall execution of

the tasks would be delayed due to input data flow from one edge deployment to another or insufficient resource availability. Likewise, other scheduling policies i.e., FDPP, FDNP-1, FDNP-2 and NDPP in our baselines (Section 5.2), would also incur the same execution delay from such offloading policies.

4.3. Execution result transmission policy

After the execution of $J \in \mathbb{J}$, the final result is immediately transmitted back to each vehicle. However, due to changes in the original location of a moving vehicle, more than one routing path may exist for each vehicle, therefore, the final result can be transmitted through intermediate EC deployments by considering the vehicle's current location $\{x_q\prime, y_q\prime\}$ speed s and the distance matrix given in (9). Therefore, our transmission policy is to determine the optimal transmission route such that a minimized transmission time is achieved, as given in Optimization (29). To this end, the goal of our EFR is to find the fastest route to efficiently forward execution results to each vehicle at its current location. Given other approaches in [50], which transmits to each vehicle in a store-and-forward manner with unknown delay, may cause latency-sensitive applications to face a high failure probability, hence our EFR adopts a forwarding policy: **direct single copy**, in which the results can only be transmitted from the source EC deployment to the vehicles without the intermediate EC deployment(s) keeping a copy, thus enabling faster transmission time. Algorithm 3 describes our result transmission procedure, where the number of paths to each vehicle is first computed (line 2), then the number of hops and total distance in that path is computed (lines 4 and 5). Finally, the shortest path with high transmission capability is selected. Note that the input data flow is different from the final result transmission. For instance, the input data flow is transmitted from one EC deployment to another within a single-hop transmission, while the job's final result transmission applies to a moving vehicle due to its change in the original location. Furthermore, the final result transmission might involve multi-hop transmissions, hence a routing plan is essential to achieve minimized transmission time. To solve this routing problem, we have adopted the CP-SAT solver.

4.4. Algorithms time complexity analysis

The dispatching algorithm has a runtime of $\mathcal{O}(1)$ in the Big-O notation when dispatching multi-job $\mathbb{J}$ to the closest edge deployment, where the edge has suitable resource availability to execute the jobs. In the worst case, the run time increases due to the dispatcher being unable to dispatch the jobs as a whole. Hence, the dispatcher can allow fractional dispatching of each $J \in \mathbb{J}$ to closest `members`, resulting in a run time of $\mathcal{O}(M)$ in the worst case, where M is the number of ready jobs. In Algorithm 2, the run time complexity is also $\mathcal{O}(N)$, where N is the number of nodes. It includes calculating the resource availability of each active node in the selected edge cluster and intelligently placing tasks tightly on each node. In Algorithm 3, the run time complexity is also $\mathcal{O}(K)$, where K is the number of vehicles. It includes calculating the number of hops and total distance to a given vehicle and then choosing the path with the least distance to the vehicle. To further improve the computational speeds of our algorithms, we have adopted various solvers, i.e., the SCIP solver and CP-SAT solver.

4.5. Connection with optimization objectives

As stated previously, our objectives are to minimize the total response time of multiple IoV applications, which consist of Optimizations (25), (27) and (29), and maximize the EC cluster resource utilization given in Optimization (21). Algorithms 1, 2 and 3 together achieve these objectives. By gang-dispatching the multi-jobs to an edge having sufficient resource availability, Algorithm 1 ensures that any EC deployment selected has sufficient resources $C^{\langle c,m\rangle}_{Edge_\star}$ needed for jobs execution, such that the dependent tasks can be executed faster, ultimately leading to a smaller aggregate scheduling time and execution time, i.e., $E_{sh}\prime$ and $E_{ex}\prime$ respectively. 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 $U^{\langle c,m\rangle}_{Edge_\star}$ as small as possible while guaranteeing it is sufficient for the multi-jobs. More specifically, the resource usage (RU) of the cluster for multi-job $\mathbb{J}$ deployment is given in Optimization (31). Finally, Algorithm 3 ensures that the optimal transmission route is used to transmit the execution results of jobs, as given in Optimization (29).

5. Performance evaluation

We evaluate our FedEdge on real-time Alibaba cluster data traces and compare its performance with three existing state-of-the-art.

5.1. Experiment setup

Computing Resources: Our $\mathbb{EF}$ setup consist of 3 RSUs and 3 BSs, as summarized in Table 3. These platforms consist of large resource capacity EC devices. For example, AWS Wavelength[7] embeds AWS compute and storage services within BSs, providing mobile EC infrastructure for developing and scaling ultra-low-latency applications. 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 [51]. Therefore, we conduct extensive experiments with orchestrated sets of multi-dependent tasks having heterogeneous resource requests across the EC resources. For each deployment, we compare the performance of our FedEdge with the existing state-of-the-art.

Applications: We employ the v-2018 version of Alibaba cluster trace, which records the activities of about 4000 machines in a period of 8 days. The entire trace contains more than 14 million tasks with more than 12 million dependencies and more than 4 million jobs. Among these, we deployed a total of 253 jobs with a total of 821 tasks (including dependencies) for our experiments. The task dependency depth among the jobs ranges from $(1,\ 17]$. Table 4 list the details of our Multi-Jobs. Task dependencies in Alibaba data trace are valuable for our investigation. Researchers have thoroughly investigated the v-2018 version of Alibaba cluster trace and used it for various task scheduling problems [52–54].

5.2. Heuristics and baselines

In our experiments, we assume that all tasks are of high priority. Our strategy called **FedEdge** utilizes the *closest* heuristic and adopts the gang-scheduling strategy and a variant bin-packing optimization to efficiently co-schedule and co-locate multi-task in a cluster or edge to minimize the overall response time. We consider FedEdge as a Full Dependency and Full Packing **(FDFP)** approach.

We compare the dispatching and scheduling approach of FedEdge with the following four existing schemes, as follows:

1. **FDPP** [40] is an approach which executes subtasks of a job locally in the vehicle, and offloads subtasks to the cloud server and others to the RSU for execution at the same time. We refer to this approach as Full Dependency and Partial Packing (FDPP).
2. **FDNP-1** [37] is an approach which 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 it 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

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

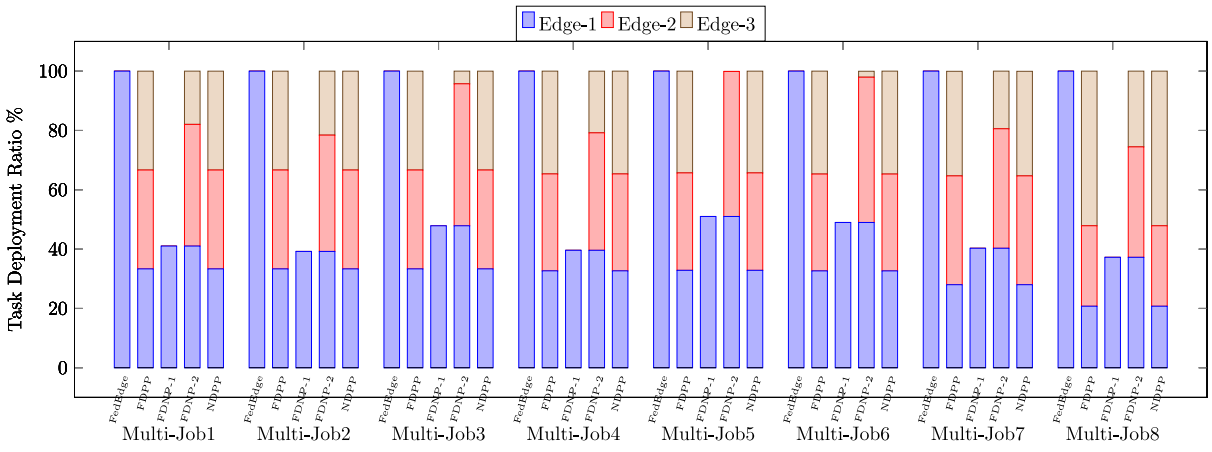

Fig. 5. Task deployment ratio across the federated edge clusters.

Table 3
Integrated-Edge resource capacities.

EC deployments	EC device(s) installation	CPU capacity (Cores)	Mem capacity (GiB/TiB)
RSU-1	Lenovo ThinkSystem(x1)	32	512 GiB
RSU-2	AWS Snowball(x1)	40	80 GiB
RSU-3	Dell EMC VxRail(x2)	112	6144 GiB
BS-1	IBM Power Systems(x4)	80	8 TiB
BS-2	HPE Edgeline(x5)	140	10 TiB
BS-3	IBM Power Systems(x1) + HPE Edgeline(x1)	48	4 TiB

Table 4
Multi-Job dispatched to integrated EC.

Multi-Job $\mathbb{J}$	Jobs	Tasks	Dependency	Cluster
`Multi-Job1`	13	39	(1, 8]	`RSU-1`
`Multi-Job2`	19	51	(1, 5]	`RSU-2`
`Multi-Job3`	40	117	(1, 8]	`RSU-3`
`Multi-Job4`	31	101	(1, 17]	`BS-1`
`Multi-Job5`	38	137	(1, 11]	`BS-2`
`Multi-Job6`	12	49	(1. 11]	`BS-3`
`Multi-Job7`	44	139	(1. 11]	`RSU-3`
`Multi-Job8`	56	188	(1. 11]	`BS-2`

priority, where it keeps multiple applications in decreasing order of their priority. We refer to this approach as Full Dependency and No Packing (FDNP).

3. **FDNP-2** [41] is an approach which 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. We consider this approach as a Full Dependency and No Packing (FDNP).
4. **NDPP** [55] is an approach which offloads different subtasks of a multi-Job to available EC deployment, by considering each task completion deadline. However, this approach does not respect inter-task dependencies but co-locates tasks on a node. We refer to this approach as No Dependency and Partial Packing (NDPP).

5.3. Deployment results and comparison

The investigation focuses on IoV multi-tasks response time, which includes offloading time, scheduling time, execution time and result transmission time. Since our offloading policy is the *Closest* and within the one-hop transmission, therefore the offloading time is relatively smaller compared to multi-hop offloading strategies, hence we focus our investigation on scheduling time, execution time and result transmission time. We use the cluster data trace from Alibaba to obtain task start time, completion time, resource requirements (CPU, Mem) and all task dependencies. The results obtained by FedEdge (FDFP), FDPP, FDNP-1, FDNP-2 and NDPP are compared. We also investigate the EC deployment CPU and memory usage/utilization.

5.3.1. Resource usage and resource utilization

Fig. 5 shows the task deployment ratio of FedEdge with the four baseline schemes. It can be seen that for each multi-job offloaded, FedEdge can deploy its constituent tasks to a single edge. This is because FedEdge 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, they either execute subtasks at a time or dispatch other subtasks to another edge, given the number of nodes at each edge. For example, `Multi-Job1` which consists of 13 jobs is deployed and co-located on `RSU-1` edge cluster by FedEdge, in turn, allows for faster input data flow transmissions. For the same `Multi-Job1`, FDPP, FDNP-2 and NDPP deploy the jobs across three edge clusters. Although FDPP and NDPP can partially co-locate tasks at each of the edge clusters, the three schemes incur additional execution delays due to input data flow transmissions across the three edge clusters. On the other hand, FDNP-1 is also able to deploy all the jobs on a single edge, but 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. Fig. 6 shows the average resource usage of the multi-jobs deployed by FedEdge with those of the four baseline schemes across the integrated edge clusters. It can be seen that FedEdge consumes the fewest resources by using a single edge for each multi-job, while FDNP-2 uses the highest resources (up to three clusters) for the same multi-job. The CPU and memory resource utilization comparisons are shown in Figs. 7 and 8, respectively. Again, FedEdge achieves the highest resource utilization compared with the four baseline schemes. We now examine the performance of FedEdge compared with the baseline schemes for each multi-job offloaded (as shown in Table 4) in detail. **`Multi-Job1:`** FedEdge dispatch 100% of the tasks in a single-hop offloading to `RSU-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

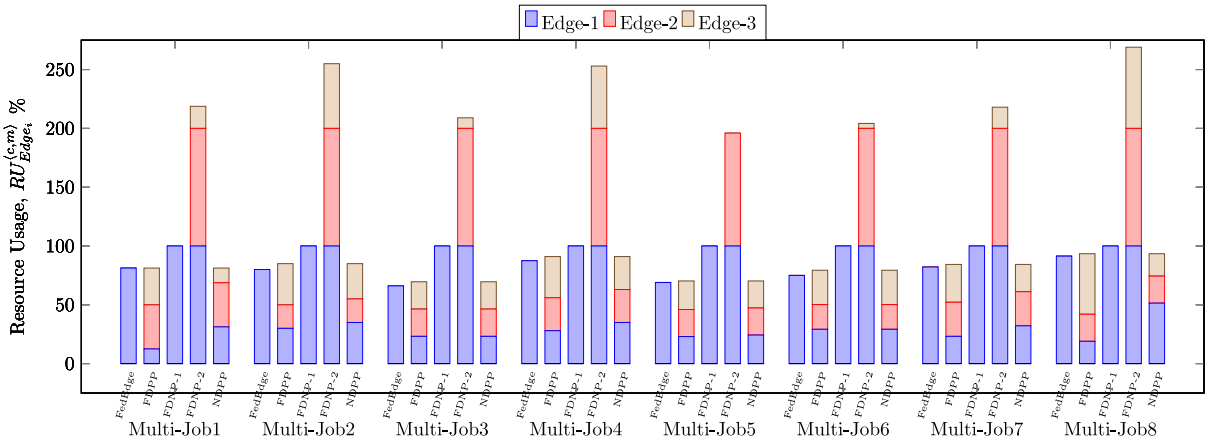

Fig. 6. Average resource usage across the federated edge clusters.

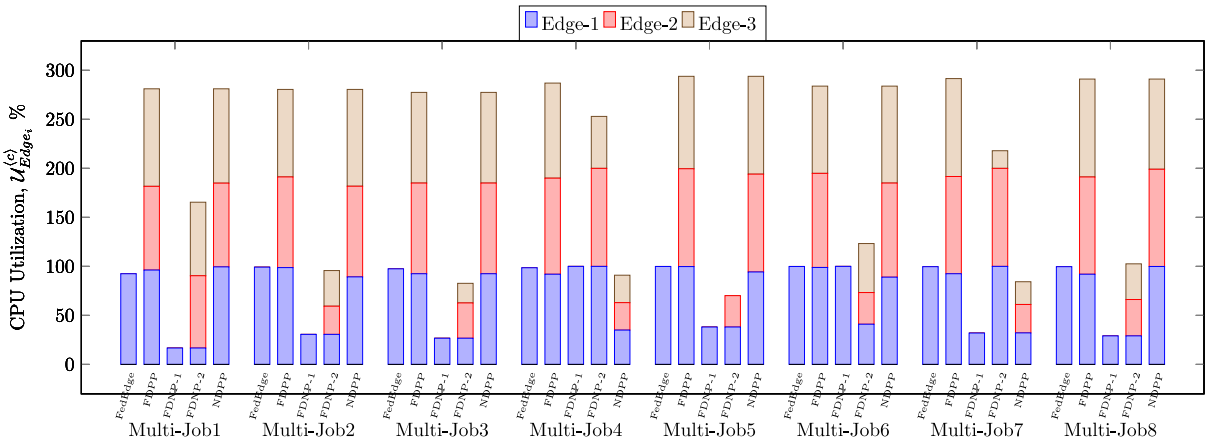

Fig. 7. CPU resource utilization across the federated edge resources.

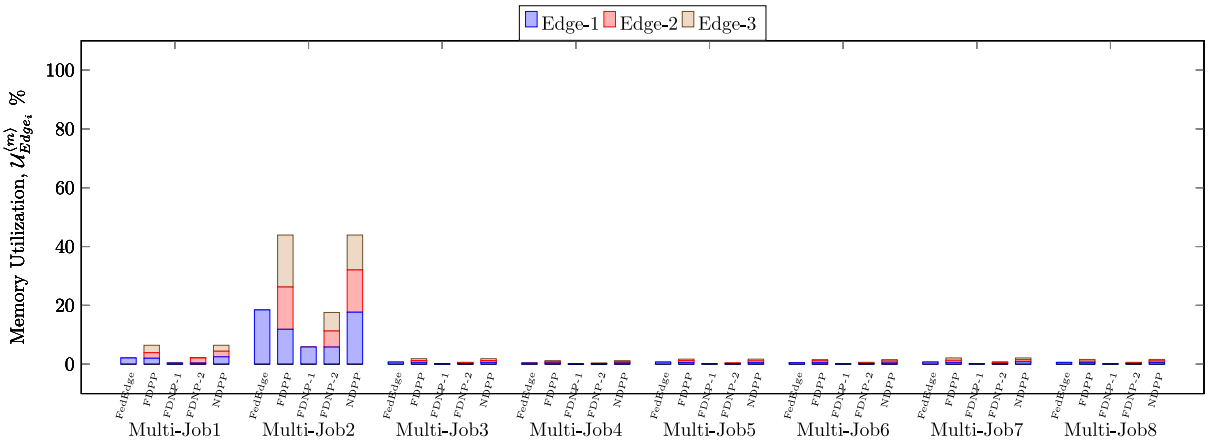

Fig. 8. Memory utilization across the federated edge resources.

packed on nodes using the packing algorithm, which uses 81% of RSU−1 edge resources to execute the tasks, and achieves 93% and 2% utilization of the CPU and memory resources respectively. For the same `Multi-Job1`, some of the baseline schemes such as FDPP, FDNP-2 and NDPP offload the tasks across three edge clusters, using up to 2.7× more resources than FedEdge. FDNP-1 schedules one task on a node at a time using a single-edge cluster. Thus, it uses all available resources (100%) at the cluster and keeps the unscheduled tasks on a task queue until resources become available. Overall, FedEdge achieves better resource usage and utilization compared to the four baseline schemes, as shown in Figs. 6, 7 and 8. **`Multi-Job2:`** This multi-job consists of 19 jobs with a total of 51 tasks, where each job has a task dependency in the range of $(1,\ 5]$. FedEdge optimizes the deployment to ensure that resources are fully utilized. Containers provide isolation to running applications, making it possible to co-locate multiple applications on the same node without any interference. A single container-optimized node can execute more containerized applications, given that there are sufficient available resources. For the scheduling, FedEdge deploys all the tasks at a time on RSU−2 edge cluster, using 80% of the resources, while FDPP, FDNP-2 and NDPP use 85%, 255% and 85% of resources

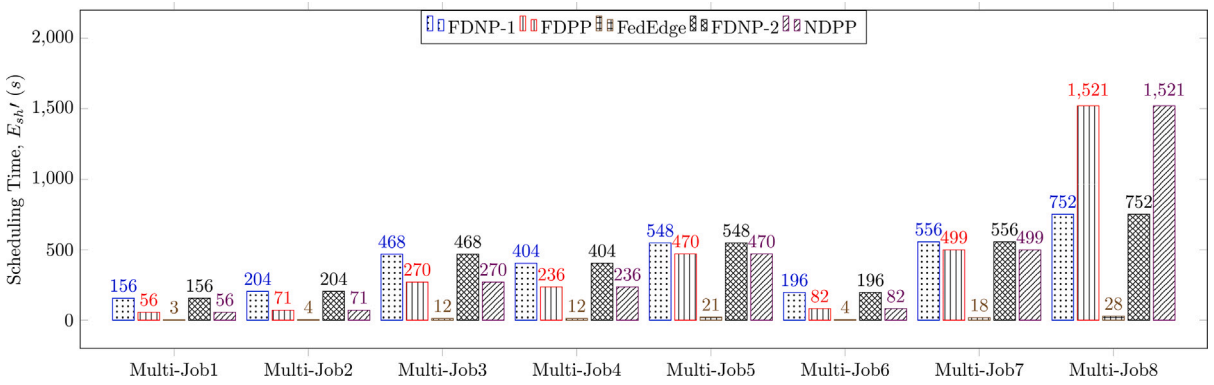

Fig. 9. Task scheduling times across the federated edge resources.

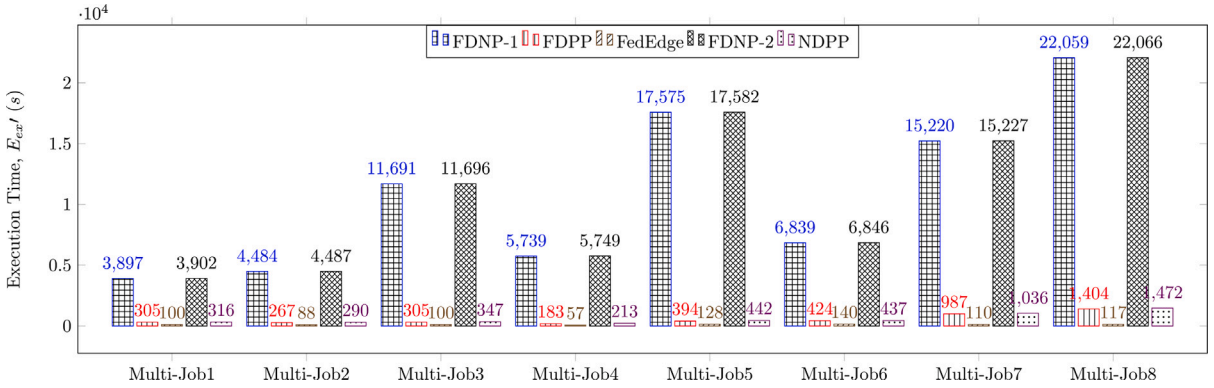

Fig. 10. Task execution times across the federated edge resources.

across three clusters, respectively. FedEdge and FDNP-1 utilize 99% and 31% of CPU resources, respectively. Although FDNP-1 uses all available resources in the cluster but achieves low resource utilization due to its inability to co-locate tasks on nodes, which results in resource under-utilization. Again FedEdge outperformed all four baseline schemes in terms of task deployment ratio, resource usage and utilization.

`Multi-Job3:` FedEdge dispatches `Multi-Job3` to `RSU-3`, which is a memory-intensive and high-capacity edge cluster. This cluster is made up of 2 Dell EMC VxRail edge devices, with CPU and memory capacity of 112 vCPU and 6144 GiB, respectively. The multi-job consists of 40 CPU-intensive jobs, with a total of 117 tasks, where each job has a task dependency range (1, 8]. FedEdge improves resource usage by using a single cluster and up to 3× fewer compared with the four baseline schemes, as can be seen from Fig. 6. It also achieves 98% and 0.65% high CPU and memory resource utilization in a single cluster, respectively. On the other hand, FDPP and NDPP achieve 93% and 0.62% CPU and memory resource utilization at each of the three edge clusters they individually used to offload subtasks of `Multi-Job3`. Note that the `Multi-Job3`~`Multi-Job8` clusters are CPU intensive jobs. Therefore, the jobs can only consume and utilize a few cluster memory resources, as shown in Fig. 8. FDNP-1 and FDNP-2 perform worst by consuming the highest resources and achieving the lowest resource utilization. **`Multi-Job4`, `Multi-Job5`** and **`Multi-Job6:`** These multi-jobs are offloaded by FedEdge to `BS-1`, `BS-2` and `BS-3`, respectively. Among all the schemes, FedEdge uses the least resources for each multi-job execution across the three clusters. Specifically, FedEdge consumes 88%, 69% and 75% of resource at `BS-1`, `BS-2` and `BS-3`, respectively. It also achieves the highest CPU (average of 99%) and memory (average of 0.5%) resource utilization across the BS clusters compared to the four baseline schemes. FDPP and NDPP consume 90% each across three different clusters for `Multi-Job4` execution, with an average of 96% CPU and 0.4% memory utilization. FDNP-1 consumes all available resources at `BS-1`, `BS-2` and `BS-3` for `Multi-Job4`, `Multi-Job5` and `Multi-Job6` executions, respectively, while recording the lowest resource utilization at each cluster. FDNP-2 consumes the second highest resources and achieves the second lowest resource utilization for the same multi-jobs execution. **`Multi-Job7`** and **`Multi-Job8:`** After the multi-jobs are executed at the integrated edge deployments, the cluster resources become available for incoming multi-jobs. Therefore, FedEdge dispatches `Multi-Job7` and `Multi-Job8` to `RSU-3` and `BS-2`, respectively, after previously executed `Multi-Job3` and `Multi-Job5` on the clusters. `Multi-Job7` and `Multi-Job8` are made up of 44 and 56 jobs, respectively. From Figs. 5, 6, 7 and 8, which show the tasks deployment ratio, resource usage, CPU and memory utilization, it shows that FedEdge is the best with the least consumed resources of 82% and 91% at `RSU-3` and `BS-2`, respectively. In addition, FedEdge achieves the highest CPU/memory utilization up to 71% and 0.5% than the benchmark schemes. The other four baseline schemes either consume all available resources in a single cluster or consume more resources across multi-cluster. Moreover, the superior performance of FedEdge over the other benchmark schemes is overwhelmingly clear.

5.3.2. *Multi-task scheduling, execution and response times*

The aggregate job scheduling time $E_{sh}{}'$ defined in Eq. (11), which is the time it takes to place multi-jobs/tasks on the nodes in a cluster, is an important performance metric to assess the integrated edge clusters. Another even more important performance metric is the aggregate job execution time $E_{ex}{}'$ defined in Eq. (12). More importantly, is the response time $E_{rsp}{}'$ which is defined in Eq. (7). Figs. 9–11 compare the scheduling times, execution times and response times, respectively, attained by the five schemes. It can be seen that the scheduling times are typically very small, and the execution times and response times by contrast are significantly larger. Across the integrated edge clusters, FedEdge consistently achieves the fastest scheduling, execution and response times, compared to the four benchmark strategies. Note that

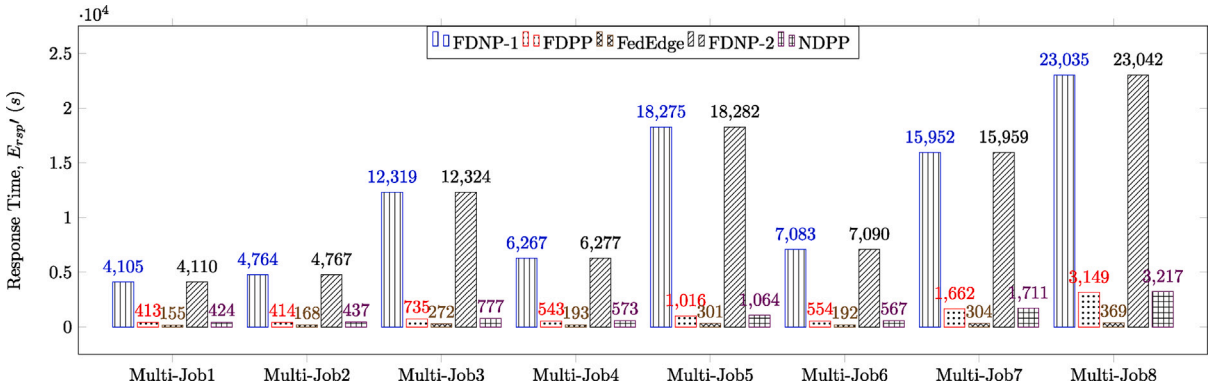

Fig. 11. Task response times across the federated edge resources.

we have focused on the scheduling, execution and result transmission times 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 – Job1`, FedEdge achieves very fast scheduling, which is 17× faster than FDPP and NDPP, and 52× faster than FDNP-1 and FDNP-2. For `Multi – Job2` scheduling, FedEdge achieves significantly faster scheduling times than the four benchmark strategies, i.e., FedEdge is 18× faster than FDPP and NDPP, and 51× faster than FDNP-1 and FDNP-2. For `Multi – Job3`, FDNP-1 and FDNP-2 attain the lowest scheduling times, while FDPP and NDPP attain the second lowest scheduling time. FedEdge achieves the best performance, up to 39× faster than the other four schemes. For `Multi – Job4`~`Multi – Job7`, FedEdge again achieves the fastest scheduling time, followed by FDPP and NDPP, while FDNP-1 and FDNP-2 have the worst scheduling time performance. For `Multi – Job8`, FedEdge is approximately 54× faster than FDPP and NDPP, and 27× faster than FDNP-1 and FDNP-2.

In terms of 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 approach of task offloading across multiple clusters, which leads to input data flow (which is in the range of (0.2, 0.4] s) across the clusters. FedEdge is 26.4×, 2.6×, 26.5× and 2.7× faster than FDNP-1, FDPP, FDNP-2 and NDPP, respectively, in executing `Multi – Job1`, while for `Multi – Job2` execution, it is approximately 51×, 3×, 51× and 3.3× faster, respectively, over the four benchmarks. Similarly, for `Multi – Job3`, `Multi – Job4` and `Multi – Job5` as well as `Multi – Job6`, `Multi – Job7` and `Multi – Job8` executions, FedEdge also achieves approximately up to 189×, 9×, 189× and 13× faster execution times than FDNP-1, FDPP, FDNP-2 and NDPP, respectively. The significant advantage of FedEdge in terms of aggregate job execution time can be explained as follows. It deploys sets of multi-jobs/tasks as a unit through the gang scheduling strategy in a single EC cluster. These applications are deployed and executed concurrently. By contrast, the benchmark approaches schedule and execute the given DAGs individually and in parts across multiple EC clusters, resulting in input data flow transmission delay and a longer time to execute the overall tasks. Recall that the response time of a job as defined in Eq. (7), 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 clusters. Fig. 11 compares the response time of FedEdge and the four benchmark schemes. FedEdge outperforms the four benchmark schemes by achieving the fastest response time for all the multi-jobs, and up to 60.7×, 3.37×, 60.7× and 3.5× faster than FDNP-1, FDPP, FDNP-2 and NDPP, respectively.

5.4. Discussion

FedEdge has demonstrated superior QoS in resource management and IoV multi-task orchestration in integrated edge clusters. Our proposed algorithm achieves both the highest edge cluster resource utilization and the minimum scheduling, execution and response times for IoV multi-tasks/jobs compared to the baseline strategies. We observe that FedEdge consumes up to 34% fewer resources and achieves up to 76% high cluster resource utilization, while leading to up to 54× faster scheduling time, up to 189× faster execution time and up to 62× faster response time. Achieving faster scheduling time and execution time, and in particular faster response time are crucial for smart vehicular applications to perform better. The gains achieved by FedEdge as observed from our experiments include efficient load-balancing and an increase in the number of tasks that can be deployed at a time t as well as faster response time of the overall tasks and improved usage of edge resources. Recall that, unlike FedEdge, the existing methods do not deploy all 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 underutilization as well as causing longer task execution time.

6. Conclusions

This paper has presented a resource-aware and dependency-aware IoV multi-task orchestration in an integrated EC system, called FedEdge, to improve edge resource efficiency and performance. We have utilized a resource-aware dispatching strategy that selects the closest edge cluster suitable for a given job(s), and a container-based bin-packing optimization strategy that packs or co-locates tasks tightly on nodes to fully utilize available resources. Our approach involves obtaining the resource demand of tasks and integrated-edge clusters update the state, through a single control plane (CP), then gang schedule and co-locate multi-task on container-optimized nodes called *container-instances*. To evaluate our approach, we have illustrated use cases of real-world CPU and memory-intensive tasks from Alibaba cluster trace, which records the activities of both long-running containers (for Alibaba's e-commerce business) and batch jobs across 8 days. We have compared our approach with the state-of-the-art dependency-aware IoV task orchestration baseline strategies. Our experimental results have demonstrated that FedEdge achieves both the highest cluster resource utilization and the minimum scheduling, execution and response times for IoV multi-tasks/jobs compared to the baseline strategies.

CRediT authorship contribution statement

Uchechukwu Awada: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Resources, Writing – original draft, Writing – review & editing, Visualization. **Jiankang Zhang:** Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Resources, Writing – original draft, Writing – review

& editing, Visualization, Supervising, Project administration, Funding acquisition. **Sheng Chen:** Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Resources, Writing – original draft, Writing – review & editing, Visualization, Supervising, Project administration. **Shuangzhi Li:** Conceptualization, Methodology, Formal analysis, Supervising, Funding acquisition, Resources, Writing – original draft, Writing – review & editing, Visualization, Project administration. **Shouyi Yang:** Conceptualization, Methodology, Formal analysis, Supervising, Funding acquisition, Writing – original draft, Writing – review & editing, Visualization, Project administration.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Shuangzhi Li reports financial support was provided by National Natural Science Foundation of China.

Data availability

Data will be made available on request.

Acknowledgments

The financial support of the National Natural Science Foundation of China under grants 61571401 and 61901416 (part of the China Postdoctoral Science Foundation under grant 2021TQ0304) and the Innovative Talent of Colleges and the University of Henan Province, China under grant 18HASTIT021 are gratefully acknowledged.

References

[1] U.Z.A. Hamid, Z. Hairi, L.D. Kumar, in: A.-T. Fadi (Ed.), Internet of Vehicle (IoV) Applications in Expediting the Implementation of Smart Highway of Autonomous Vehicle: A Survey, Springer International Publishing, Cham, 2019, pp. 137–157, http://dx.doi.org/10.1007/978-3-319-93557-7_9.

[2] S. Liu, L. Liu, J. Tang, B. Yu, Y. Wang, W. Shi, Edge computing for autonomous driving: Opportunities and challenges, Proc. IEEE 107 (8) (2019) 1697–1716.

[3] S. Sukhmani, M. Sadeghi, M. Erol-Kantarci, A. El Saddik, Edge caching and computing in 5G for mobile AR/VR and tactile internet, IEEE MultiMedia 26 (1) (2018) 21–30.

[4] A. Yousefpour, C. Fung, T. Nguyen, K. Kadiyala, F. Jalali, A. Niakanlahiji, J. Kong, J.P. Jue, All one needs to know about fog computing and related edge computing paradigms: A complete survey, J. Syst. Archit. 98 (2019) 289–330, http://dx.doi.org/10.1016/j.sysarc.2019.02.009.

[5] M. Barzegaran, P. Pop, Extensibility-aware fog computing platform configuration for mixed-criticality applications, J. Syst. Archit. 133 (2022) 102776, http://dx.doi.org/10.1016/j.sysarc.2022.102776.

[6] A. Islam, A. Debnath, M. Ghose, S. Chakraborty, A survey on task offloading in multi-access edge computing, J. Syst. Archit. 118 (2021) 102225, http://dx.doi.org/10.1016/j.sysarc.2021.102225.

[7] L. Qian, Z. Qu, M. Cai, B. Ye, X. Wang, J. Wu, W. Duan, M. Zhao, Q. Lin, FastCache: A write-optimized edge storage system via concurrent merging cache for IoT applications, J. Syst. Archit. 131 (2022) 102718, http://dx.doi.org/10.1016/j.sysarc.2022.102718.

[8] S. Yuan, J. Li, C. Wu, JORA: Blockchain-based efficient joint computing offloading and resource allocation for edge video streaming systems, J. Syst. Archit. 133 (2022) 102740, http://dx.doi.org/10.1016/j.sysarc.2022.102740.

[9] M. Liwang, R. Chen, X. Wang, Resource trading in edge computing-enabled IoV: An efficient futures-based approach, IEEE Trans. Serv. Comput. (2021) 1, http://dx.doi.org/10.1109/TSC.2021.3070746.

[10] C. Chen, Y. Zeng, H. Li, Y. Liu, S. Wan, A multi-hop task offloading decision model in MEC-enabled internet of vehicles, IEEE Internet Things J. (2022) 1, http://dx.doi.org/10.1109/JIOT.2022.3143529.

[11] L. Zhang, W. Zhou, J. Xia, C. Gao, F. Zhu, C. Fan, J. Ou, DQN-based mobile edge computing for smart internet of vehicle, EURASIP J. Adv. Signal Process. 2022 (1) (2022) 45, http://dx.doi.org/10.1186/s13634-022-00876-1.

[12] F. Sun, Z. Zhang, S. Zeadally, G. Han, S. Tong, Edge computing-enabled internet of vehicles: Towards federated learning empowered scheduling, IEEE Trans. Veh. Technol. (2022) 1–17, http://dx.doi.org/10.1109/TVT.2022.3182782.

[13] L. Mu, B. Ge, C. Xia, C. Wu, Multi-task offloading based on optimal stopping theory in edge computing empowered internet of vehicles, Entropy 24 (6) (2022) http://dx.doi.org/10.3390/e24060814.

[14] J. Lu, L. Chen, J. Xia, F. Zhu, M. Tang, C. Fan, J. Ou, Analytical offloading design for mobile edge computing-based smart internet of vehicle, EURASIP J. Adv. Signal Process. 2022 (1) (2022) 1–19.

[15] Z. Ning, J. Huang, X. Wang, J.J.P.C. Rodrigues, L. Guo, Mobile edge computing-enabled internet of vehicles: Toward energy-efficient scheduling, IEEE Netw. 33 (5) (2019) 198–205, http://dx.doi.org/10.1109/MNET.2019.1800309.

[16] D. Zhang, L. Cao, H. Zhu, T. Zhang, J. Du, K. Jiang, Task offloading method of edge computing in internet of vehicles based on deep reinforcement learning, Cluster Comput. 25 (2) (2022) 1175—1187, http://dx.doi.org/10.1007/s10586-021-03532-9.

[17] L. Yao, X. Xu, M. Bilal, H. Wang, Dynamic edge computation offloading for internet of vehicles with deep reinforcement learning, IEEE Trans. Intell. Transp. Syst. (2022) 1–9, http://dx.doi.org/10.1109/TITS.2022.3178759.

[18] D. Pliatsios, P. Sarigiannidis, T. Lagkas, V. Argyriou, A.-A.A. Boulogeorgos, P. Baziana, Joint wireless resource and computation offloading optimization for energy efficient internet of vehicles, IEEE Trans. Green Commun. Netw. (2022) 1, http://dx.doi.org/10.1109/TGCN.2022.3189413.

[19] Z. Liu, X. Xu, Latency-aware service migration with decision theory for internet of vehicles in mobile edge computing, Wirel. Netw. (2022) 1–13.

[20] M.Z. Alam, A. Jamalipour, Multi-agent DRL-based hungarian algorithm (MADRLHA) for task offloading in multi-access edge computing internet of vehicles (IoVs), IEEE Trans. Wireless Commun. (2022) 1, http://dx.doi.org/10.1109/TWC.2022.3160099.

[21] W. Fan, J. Liu, M. Hua, F. Wu, Y. Liu, Joint task offloading and resource allocation for multi-access edge computing assisted by parked and moving vehicles, IEEE Trans. Veh. Technol. 71 (5) (2022) 5314–5330, http://dx.doi.org/10.1109/TVT.2022.3149937.

[22] 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, 2020, pp. 91–98, http://dx.doi.org/10.1109/EDGE50951.2020.00021.

[23] 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, 2021, pp. 671–680, http://dx.doi.org/10.1109/CLOUD53861.2021.00086.

[24] U. Awada, J. Zhang, S. Chen, S. Li, AirEdge: A dependency-aware multi-task orchestration in federated aerial computing, IEEE Trans. Veh. Technol. (2021) 1, http://dx.doi.org/10.1109/TVT.2021.3127011.

[25] 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, http://dx.doi.org/10.1109/TNET.2020.2979361.

[26] J. Ahmed, M.A. Razzaque, M.M. Rahman, S.A. Alqahtani, M.M. Hassan, A stackelberg game-based dynamic resource allocation in edge federated 5G network, IEEE Access 10 (2022) 10460–10471, http://dx.doi.org/10.1109/ACCESS.2022.3144960.

[27] H. Baghban, A. Rezapour, C.-H. Hsu, S. Nuannimnoi, C.-Y. Huang, Edge-AI: IoT request service provisioning in federated edge computing using actor-critic reinforcement learning, IEEE Trans. Eng. Manage. (2022) 1–10, http://dx.doi.org/10.1109/TEM.2022.3166769.

[28] W. Li, Q. Li, L. Chen, F. Wu, J. Ren, A storage resource collaboration model among edge nodes in edge federation service, IEEE Trans. Veh. Technol. (2022) 1, http://dx.doi.org/10.1109/TVT.2022.3179363.

[29] 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) http://dx.doi.org/10.1145/3382756.

[30] B. Shen, X. Xu, L. Qi, X. Zhang, G. Srivastava, Dynamic server placement in edge computing toward internet of vehicles, Comput. Commun. 178 (2021) 114–123, http://dx.doi.org/10.1016/j.comcom.2021.07.021.

[31] 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, 2020, pp. 1053–1062, http://dx.doi.org/10.1109/INFOCOM41043.2020.9155445.

[32] C. Anderson, Docker [software engineering], IEEE Softw. 32 (3) (2015) 102–c3, http://dx.doi.org/10.1109/MS.2015.62.

[33] 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), 2016, pp. 110–117, http://dx.doi.org/10.1109/CloudCom.2016.0032.

[34] M. Wang, T. Ma, T. Wu, C. Chang, F. Yang, H. Wang, Dependency-aware dynamic task scheduling in mobile-edge computing, in: 2020 16th International Conference on Mobility, Sensing and Networking, MSN, 2020, pp. 785–790, http://dx.doi.org/10.1109/MSN50589.2020.00134.

[35] 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, 2019, pp. 2037–2046, http://dx.doi.org/10.1109/ICDCS.2019.00201.

[36] 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, OSDI '16, USENIX Association, USA, 2016, pp. 81–97.

[37] Y. Liu, S. Wang, Q. Zhao, S. Du, A. Zhou, X. Ma, F. Yang, Dependency-aware task scheduling in vehicular edge computing, IEEE Internet Things J. 7 (6) (2020) 4961–4971, http://dx.doi.org/10.1109/JIOT.2020.2972041.

[38] 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, ICC '17, Association for Computing Machinery, New York, NY, USA, 2017, http://dx.doi.org/10.1145/3018896.3056798.

[39] 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, CCGrid '17, IEEE Press, 2017, pp. 929–934, http://dx.doi.org/10.1109/CCGRID.2017.113.

[40] G. Liu, F. Dai, B. Huang, Z. Qiang, S. Wang, L. Li, Dependency-aware task offloading for vehicular edge computing with end-edge-cloud collaborative computing, 2022, http://dx.doi.org/10.21203/rs.3.rs-1905904/v1, PREPRINT (Version 1).

[41] Q. Shen, B.-J. Hu, E. Xia, Dependency-aware task offloading and service caching in vehicular edge computing, IEEE Trans. Veh. Technol. (2022) 1–16, http://dx.doi.org/10.1109/TVT.2022.3196544.

[42] F. Wu, X. Li, X. Luo, K. Gu, A novel authentication scheme for edge computing-enabled internet of vehicles providing anonymity and identity tracing with drone-assistance, J. Syst. Archit. 132 (2022) 102737, http://dx.doi.org/10.1016/j.sysarc.2022.102737.

[43] A. Fontes, I.d.L. Ribeiro, K. Muhammad, A.H. Gandomi, G. Gay, V.H.C. de Albuquerque, AI-empowered data offloading in MEC-enabled IoV networks, 2022, arXiv preprint arXiv:2204.10282.

[44] 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, Boston, MA, 2018.

[45] 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, Middleware '17, Association for Computing Machinery, New York, NY, USA, 2017, pp. 95–108, http://dx.doi.org/10.1145/3135974.3135984.

[46] Y. Li, D. Sun, B.C. Lee, Dynamic colocation policies with reinforcement learning, ACM Trans. Archit. Code Optim. 17 (1) (2020) http://dx.doi.org/10.1145/3375714.

[47] 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 Internet Things J. 7 (3) (2020) 1678–1689, http://dx.doi.org/10.1109/JIOT.2019.2943373.

[48] 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, http://dx.doi.org/10.1109/TII.2020.2990674.

[49] J. Chen, Y. He, Y. Zhang, P. Han, C. Du, Energy-aware scheduling for dependent tasks in heterogeneous multiprocessor systems, J. Syst. Archit. 129 (2022) 102598, http://dx.doi.org/10.1016/j.sysarc.2022.102598.

[50] J. Cheng, J. Cheng, M. Zhou, F. Liu, S. Gao, C. Liu, Routing in internet of vehicles: A review, IEEE Trans. Intell. Transp. Syst. 16 (5) (2015) 2339–2352, http://dx.doi.org/10.1109/TITS.2015.2423667.

[51] Z. Hong, W. Chen, H. Huang, S. Guo, Z. Zheng, Multi-hop cooperative computation offloading for industrial IoT-edge-cloud computing environments, IEEE Trans. Parallel Distrib. Syst. 30 (12) (2019) 2759–2774, http://dx.doi.org/10.1109/TPDS.2019.2926979.

[52] 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), 2019, pp. 1–10, http://dx.doi.org/10.1145/3326285.3329074.

[53] 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, 2019, pp. 696–707, http://dx.doi.org/10.1109/IPDPS.2019.00078.

[54] 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, in: ICBDC 2019, Association for Computing Machinery, New York, NY, USA, 2019, pp. 48–53, http://dx.doi.org/10.1145/3335484.3335513.

[55] C. Liu, K. Liu, S. Guo, R. Xie, V.C.S. Lee, S.H. Son, Adaptive offloading for time-critical tasks in heterogeneous internet of vehicles, IEEE Internet Things J. 7 (9) (2020) 7999–8011, http://dx.doi.org/10.1109/JIOT.2020.2997720.

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

Jiankang Zhang is a Senior Lecturer at Bournemouth University. Before 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. His research interests are in the areas of aeronautical communications, aeronautical networks, evolutionary algorithms and edge computing. He serves as an Associate Editor for IEEE ACCESS.

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

Shuangzhi Li received the B.S. and Ph.D. degrees from the School of Information Engineering, Zhengzhou University, Zhengzhou, China, in 2012 and 2018, respectively. From 2015 to 2017, he was a Visiting Student with the Department of Electrical and Computer Engineering, at McMaster University, Canada. He is currently a Lecturer at the School of Information Engineering, Zhengzhou University, China. His research interests include noncoherent space–time coding and ultra-reliable low-latency communications.

Shouyi Yang received a Ph.D. degree from the Beijing Institute of Technology, Beijing, China, in 2002. He is currently a Full Professor at the School of Information Engineering, Zhengzhou University, Zhengzhou, China. He has authored or co-authored various articles in the field of signal processing and wireless communication. His current research interests include signal processing in communications systems, wireless communications, and cognitive radio.