IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received March 21, 2022, accepted April 12, 2022, date of publication April 18, 2022, date of current version April 25, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3167641

Parked Vehicles Task Offloading in Edge Computing

KHOA NGUYEN[1], STEVE DREW[2], (Member, IEEE), CHANGCHENG HUANG[1], (Senior Member, IEEE), AND JIAYU ZHOU[3], (Member, IEEE)

[1]Department of Systems and Computer Engineering, Carleton University, Ottawa, ON K1S 5B6, Canada
[2]Department of Electrical and Software Engineering, University of Calgary, Calgary, AB T2N 1N4, Canada
[3]Department of Computer Science and Engineering, Michigan State University, East Lansing, MI 48824, USA

Corresponding author: Khoa Nguyen (khoatnguyen@sce.carleton.ca)

This research was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC) Engage under Grant EGP543449-19, the National Science Foundation IIS-1749940, the Office of Naval Research N00014-20-1-2382, and University of Calgary Start-up Funding 10032260.

ABSTRACT The analytical research has recently indicated that the computational resources of Connected Autonomous Vehicles (CAVs) have been wasted since almost all vehicles spend over 95% of their time in parking lots. This paper presents a collaborative computing framework to efficiently offload online computational tasks to parked vehicles (PVs) during peak business hours. To maintain the service continuity, we advocate for integrating Kubernetes-based container orchestration to leverage its advanced features (e.g., auto-healing, load balancing, and security). We analytically formulate the task-offloading problem and then propose an intelligent meta-heuristic algorithm to dynamically deal with online heterogeneous demands. Additionally, we take a cumulative incentives model into account, where the PV owners are able to earn profit by sharing their computation resources. We also compare our algorithm with several existent heuristics on different sizes of the parking lot. Extensive simulation results show that our proposed computing framework significantly increases the possibility of accepting the online tasks and improves average task offloading cost by at least 40%. Besides, we quantify the PV availability by task acceptance ratios, which can be a critical criterion for network planners to achieve desired network service goals.

INDEX TERMS Parked vehicles, cloud computing, edge computing, collaborative cloud-edge computing, online task offloading, container orchestration, Kubernetes.

I. INTRODUCTION

In the last decade, we have experienced a rapid proliferation of vehicles worldwide, which is estimated to reach two billion by 2035 [1]. The majority of them would come equipped with powerful on-board hardware (e.g., sensors, general-purposed CPU, GPU) to offer advanced key features such as autopilot, driver-assistance, smart radars, enhanced sensing-safety systems. Especially, the on-board equipment enabling future full self-driving capabilities could cost vehicle owners thousands of dollars. However, the resource utilization of these modern vehicles is extremely low: 70% of all vehicles spend almost 95% of the time in parking lots, home garages, and street parking as disclosed in [1], [2]. For example, America's average daily driving time was only 50.6 minutes reported for Traffic Safety by AAA Foundation in 2016 [3]. These facts obviously imply that the powerful on-board vehicular facility is unused for most of the time, providing an excellent opportunity for exploiting these neglected computing resources for ordinary network services, and potentially gaining profit by trading the idle computational power [4].

The explosive growth of mobile data traffic, either latency-insensitive (e.g., health monitoring, location-based augmented reality games, vehicular sensing) or latency-sensitive (e.g., video surveillance, mobile gaming, autopilot) tasks with heterogeneous demands [5], will pose a formidable obstacle to the existing architecture during peak hours indeed. When most services are deployed at the cloud-side, service vendors are barely in with an opportunity to negotiate the costs for offloading services which are most likely provisioned by Service Providers (SPs) with a fixed amount of service charges. Moreover, the major impediment of the core cloud is large propagation latency, so the advent of

The associate editor coordinating the review of this manuscript and approving it for publication was Tariq Umer.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

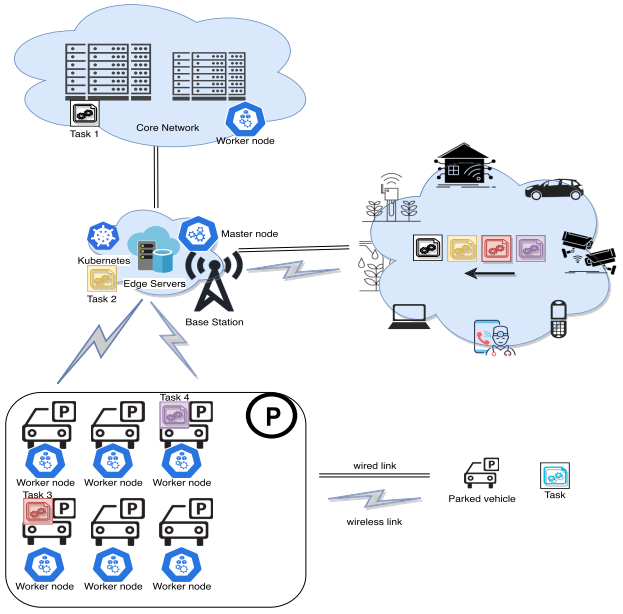

FIGURE 1. **Kubernetes-enabled parked vehicle edge computing architecture.**

edge computing with proximity to end-users is indispensably a sound solution for this problem. Indeed, the coexistence of the cloud and edge computing paradigm is among the most dominant task-offloading schemes in practice [6], and incoming tasks, in reality, are not always latency-sensitive. As such, tasks with sensitive-latency tolerance are most likely to be processed at the edge, while those with insensitive latency can be managed at both the cloud and edge network. However, networks can quickly become congested when tasks dramatically increase in peak hours. We need an effective solution to solve this problem.

The idle computation resources of parked vehicles (PVs) could be an ideal candidate for multi-access computing, where the typical computational and storage services commonly handled by the core cloud can be moved to the network edge. Due to the emergence of PVs, the capacity at the network edge can now be extended. Despite this obvious advantage, a collaboration among cloud, edge, and PVs would escalate the task offloading problems by efficiently allocating proper network resources to arrived online tasks. In addition to this, the parking duration of PVs is inconstant, making the potential PVs unreliable to host applications or services. Therefore, a novel network architecture is designed to resolve those aforesaid problems.

In fact, deploying a generic container orchestration to edge computing assisted by PVs has been hitherto in infancy. The de facto industrial standard framework for container orchestration such as Kubernetes allows PVs to simultaneously and efficiently handle several task replicas, enabling quick boot-up, autoscaling, self-healing features, and rapid termination. The appealing features can be capable of addressing the uncertain parking duration of PVs. It also offers resource isolation, allowing a PV to run multiple tasks independently. However, it is a non-trivial task to indicate in which several replicas of a given task are offloaded to be processed in the collaborative computing architecture, satisfying rigorous resource constraints while achieving minimized offloading costs. For instance, all replicas of a task can be embedded into different nodes (e.g., PVs, cloud, edge server) or a single node (e.g., a PV). In the latter, if this node suffers an abrupt failure (e.g., battery outage, accidental mobility of vehicles), the containerized network services operating on this node will experience a service disruption. It is different from our previous work in [7] that did not fully take where the task replicas could be placed into account, meaning that the whole task replicas could be allocated in the same worker node. This paper solves such limitation of [7]. To maintain service continuity and reliability, we consider an upper boundary on the proportional number of task replicas running on a network node. We set out this proportion no greater than 50%, which can be easily adjusted by SPs based on their service strategies. It means that a single worker node can handle a maximum of 50% proportion of task replicas. This setting is aimed at failover negotiation and load balancing. In the first sense, it might look simple, but the online task offloading problem at the edge itself is challenging with several constraints, and now its complexity is increased considerably with this constraint.

In our article, we propose EdgePV, a novel collaborative framework where PVs increase the computing capacity of the existing Cloud and Edge infrastructure to manage the online containerized tasks during peak business hours at the network edge. A containerized task is abstracted as a set of replicas operating on several containers in a containerization environment. Scheduling several replicas of an online task in the collaborative framework efficiently while meeting rigid resource constraints (e.g., latency-sensitive) remains a critical challenge. We formulate the task offloading problem on the collaborative paradigm as Binary Integer Programming (BIP), focusing on minimizing offloading cost while maximizing the cumulative rewards. We then propose a meta-heuristic Genetic Algorithm (GA) to deal with time complexity and scalability problems of BIP regarding low offloading costs and guaranteed reliability. In addition, PV owners who are willing to share their idle resources can accumulate incentives, which can be converted into membership, gift cards, promotion vouchers, parking tickets, gas credits, and so on.

Our proposed solution studies the feasibility of integrating the core cloud computing, the edge computing, and PVs in a unified computing infrastructure. As illustrated in Fig. 1, we suggest the Kubernetes master node be placed at the edge server. In contrast, core cloud, available computing resources of edge server, and all PVs can be considered as worker nodes. In this paradigm, PVs are installed a lightweight version of Kubernetes (e.g., K3S [8]), implemented as preamble network nodes due to their uncertain parking duration. All network components can automate the container deployment

to handle online task requests. Thanks to the high availability of cloud and edge nodes, when a task arrives at the edge server, all the replicas of a task can be scheduled at the same cloud or edge node or both. They can also be allocated on distributed PVs to exploit the idle resources of PVs to save the network cost, reducing heavy workloads in the core networks during peak hours. All master nodes and their worker nodes form a container orchestration cluster where the control plane in the master node is responsible for managing worker nodes and pods in the cluster, monitoring the state of the cluster, and making global decisions towards the cluster such as scheduling, scaling. The master node's scheduler carries out pod placements on a set of available worker nodes. When an online task with rigid constraints (e.g., CPU, BW, latency tolerance, replicas) arrives in the master node, a kube-scheduler in the control plane of the master node will create pods and then assign the worker nodes for them to run on. Interested readers can find more details about Kubernetes in [9]. Our proposed collaborative paradigm strives for improve the elasticity and agility of the existing computing infrastructure to minimize the service disruptions caused by the unforeseen mobility of PVs. In fact, all PVs would be completely electric-based, which could be enabled the automatically-charging feature during their parking in near future. Additionally, this paper considers a generic scenario in which one base station covers all PVs within its coverage of a parking lot. Thus, the control signalling (e.g., MCS, resource management, QoS, etc.) between BS and PVs over radio interface is neglected for simplification.

Our contributions are summarized below:

- We propose a collaborative computation paradigm integrating the core cloud, edge, and PVs into a consolidated architecture to address the online task offloading problem in peak hours.
- A container orchestration framework relied upon the Kubernetes platform is leveraged to deal with the uncertain parking duration of PVs. Kubernetes platform provides non-disruptive services and minimizes the possible service interruptions due to an advanced self-healing feature. When a worker node is out-of-service, Kubernetes reallocates the corresponding task replicas into another active node automatically.
- We also formulate the online task offloading model as a BIP problem to minimize the offloading cost whilst optimizing cumulative rewards of PVs by selling their idle resources considering the replicas as a constraint.
- A meta-heuristic algorithm that relied on a Genetic Algorithm, namely EdgeGA, is proposed to deal with the time complexity as well as scalability problems of BIP. We simulate the task-offloading model in respect to the random mobility behaviors of PVs under dynamic and arbitrary task arrivals. EdgeGA is compared with existing heuristics to prove its efficiency on different generic parking lots in performance simulation. We also propose a distributed parallel scheme for running the EdgeGA algorithm to reduce the execution time.

The contents of our paper are divided into the following sections. Section II presents the related work while the formulated offloading problem is introduced in Section III. Section IV proposes the GA algorithm based on the problem formulation. Thereupon, the simulation evaluation is demonstrated in Section VI. Finally, Section VII concludes our work.

II. RELATED WORK

PVs as infrastructure has currently attracted a lot of research attention as they enable the existing computation paradigm for computation, communication, and storage (CCS) to be expanded. Deploying PVs as vehicular cloud computing on the Internet of Vehicles has been well investigated in [10]–[16].

Arif *et al.* in [10] presented a basic model of a vehicular cloud (VC) assisted by PVs in a specific international airport. In contrast, He *et al.* in [11] proposed a multilayered vehicular cloud infrastructure that was relied upon the cloud computing and Internet of Thing (IoT) technologies. The paper was based on the prediction of the parking occupancy in order to schedule the network resources, and eventually allocated the computational tasks. The smart parking and vehicular data-mining services in IoT environment, were investigated. Likewise, establishing a VC built-in PVs as spatial-temporal network infrastructure for CCS in a parking lot was studied in [12]–[14], [17]. Li *et al.* in [15] paid attention to the PVs feasibility as a computing framework, and then presented an incentive mechanism considering accumulated rewards of PVs when they were trading their resources. Furthermore, Hou *et al.* in [18] introduced a vehicular fog computing (VFC) paradigm exploiting the connected PVs as the infrastructure at the edge to process real time network services. In a similar approach, the authors in [19] considered a fog computing infrastructure implemented on the Internet of Vehicle (IoV) systems to offer the computing resources to end-users concerning latency constraint. This proposed architecture allowed the network traffic to be offloaded in real time to the fog-based IoV systems subject to optimizing the average response time.

Moreover, Parked Vehicle Edge Computing (PVEC), in which PVs were recognized as accessible edge computation nodes to address the task allocation problem, has been researched in [1], [3], [20]. Huang *et al.* in [1] exploited the possible opportunistic resources for allocating the computational tasks in a collaborative architecture consisting of vehicle edge computing (VEC) server and PVs. The authors addressed the optimization problem of user payments by relaxing budget or latency constraints, which resulted in suboptimal solutions for the proposed scheme. In similarity, the paper [3] suggested a dynamic pricing approach to reduce the average cost whilst satisfying QoS constraints. This strategy calibrate the price constantly following the current system state. Besides, a containerized task scheduling scheme assisted by PVEC was presented in [20] considering the formulated social welfare optimization for users and PVs at

the same time. Raza *et al.* in [16] studied a vehicle-assisted MEC architecture combining the core cloud, MEC, and mobile volunteering vehicles (e.g., buses) to deal with IoT devices' task requests. The paper [16], at the first look, is analogous to the idea of our paper. Nevertheless, this research mainly targets the online task offloading problems in a container-based computation paradigm regarding the allocation problem for the set of online task replicas of a given VNR. Our solution takes both average network cost and accumulating incentives gained by selling idle computation resources of PVs into account. Moreover, PVs in the EdgePV framework would be more common and reliable than buses, thanks to their popularity and less mobility. Our previous work in [7] addressed the online task offloading problem in a collaborative architecture, but [7] allowed to allocate all task replicas in to the same node, exposing to another fatal problem that the vehicle that is hosting all replicas of a task suddenly leaves from the parking lot, the service provisioning will be interrupted. Additionally, [7] proposed a simple heuristic algorithm, named M&M, which ranked the worker nodes based on the offloading cost and revenue. The transmission data rate was randomly assigned instead of being dependent on the distance between PVs and BS. [7] only considered a medium size of the parking lot.

This paper is an extension of [21] where we expand the related work by analyzing more relevant papers with their specific strengths and limitations. To provide the illustration of our solutions, we depict an example of a simple task offloading scenario and then provide pseudocode for all algorithms. Furthermore, we extend to double the size of a generic parking lot in [21], compare the offloading performance on different sizes and then quantify them accordingly. This makes our proposed solution more comprehensive. We also carry out a further analysis on the achieved simulation results, and make an execution-time comparison of EgdePV algorithm adopted in both sequential and parallel manners to demonstrate the efficiency of the proposed distributed parallel deployment.

III. PROBLEM FORMULATION

In this paper, we formulate the task offloading problem considering multiple resource constraints at the network edge, where the scheduler of the container orchestration is located in the edge server deployed in a 5G base station (BS).

A. OFFLOADING MODEL

We investigate CPU, memory, and bandwidth resources in the online task offloading problems. There are various types of worker nodes, comprising the core cloud, edge server, and PVs, where they are connected to a master node located in the edge server through separate connections. For instance, the link between the master node and the core cloud is optical, whereas PVs connect to the edge server through wireless links in which the available bandwidth are primarily dependent on the distances between PVs and BS. Therefore, the edge network under a containerized cluster in this paper can be quickly realized as a star topology in which the root and its leaves are the master node and several worker nodes, respectively. Fig. 1 illustrates a generic outdoor parking lot. PVs need to register all vehicular information such as owner's ID, available parking time, license plate, available resources (e.g., computing capacity, storage) with their corresponding SPs. Then, they could keep their current vehicular status updated by sending these information to their corresponding

TABLE 1. List of acronyms and notations.

Notation	Description
PVs	Parked Vehicles
SPs	Service Providers
MEC	Multi-access Edge Computing
MCS	Modulation and Coding Scheme
QoS	Quality of Service
$KPIs$	Key Performance Indicators
BIP	Binary Integer Programming
GA	Genetic Algorithm
$D2D$	Device to Device
VC	Vehicle Cloud
CCS	Computation, Communication and Storage
IoT	Internet of Things
VFC	Vehicular Fog Computing
IoV	Internet of Vehicles
$PVEC$	Parked Vehicle Edge Computing
VEC	Vehicle Edge Computing
BS	Base Station
GA	Orthogonal Frequency Division Multiple Access
C_c	Cloud CPU capacity
M_c	Cloud Memory capacity
B_c	Cloud Bandwidth
C_e	Edge CPU capacity
M_e	Edge Memory capacity
$W_{\{C_c,M_c,B_c\}}$	Weights of Cloud capacity
$W_{\{C_e,M_e\}}$	Weights of Edge capacity
$W_{\{C_p,M_p,B_p\}}$	Weights of PVs capacity
$LTE-A$	Long Term Evolution Advanced
B_p	Channel Bandwidth
P_{TX}	BS transmission power
I	Intercell interference
N_0	Gaussian Noise Power
h	Frequency-flat block-fading Rayleigh fading channel
σ	2
C_p	CPU Parked Vehicles
χ_k	Input data size
f_k	CPU cycles per bit
$m(k)$	Memory requests
t_m	Tolerable latency of tasks
$\eth(k)$	requested replicas
r_p^c	10
r_p^m	100

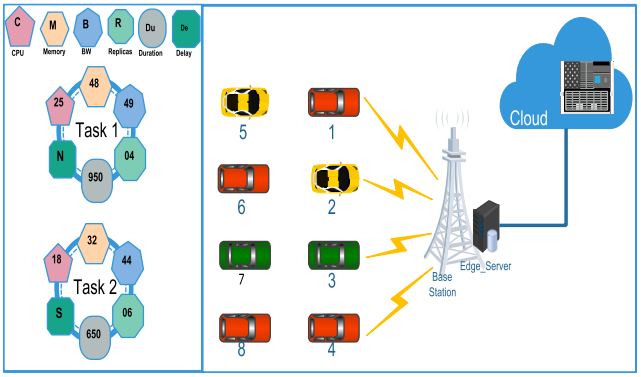

FIGURE 2. An example of task scheduling.

master node as well as the registered SPs whenever PVs either arrive at a parking lot or accomplish processing the allocated tasks.

As a result, the edge network can be modeled as a directed graph $G = (N, L)$ in which N is the set of worker nodes while L is the set of links. For instance, Fig. 2 depicts different tasks with a set of required resources such as CPU, memory, bandwidth, the number of replicas, task duration, and eventually latency requirements. Accordingly, task 1 associated with replicas could be allocated to any types of network nodes (e.g., core cloud, edge server, or PVs) due to the insensitive latency demand. However, task 2 with sensitive latency demand cannot be offloaded to the core cloud, even it acquires less computing resources. In case PVs are selected for offloading, a PV is merely permitted to process maximum three replicas to guarantee the service reliability. In addition, the edge server and the core cloud are connected through an optical connection, whereas PVs link to their network edge through wireless connections, denoted as l_c and l_v respectively. In fact, a single worker node is able to initialize several pods to handle multiple containers of the corresponding task replicas simultaneously. An online task k in our model demands CPU $c(k)$, Memory $m(k)$, Bandwidth $b(k)$, tolerable latency $t_m(k)$ and a number of replicas $\eth(k)$ requirements. k_j represents the j^{th} replica of the task $k \in K$, and $\sum k_j = \eth(k)$. Additionally, a worker node $n_i \in N$ possesses a resource capacity to serve a limited number of containers. An i^{th} worker node has CPU and memory capacity, denoted as $C(n_i)$ and $M(n_i)$, respectively. Besides, K_c, K_e, K_p and K are sets of tasks that are successfully offloaded to the cloud, edge, PVs and the whole network respectively, so $K = K_c + K_e + K_p$. Consequently, the residual CPU and memory capacity of each worker node can be calculated as follows:

$$R_C^u(n_i) = C(n_i) - \sum_{k \in K} \sum_{j \in \eth(k)} c(k_j), \quad \forall n_i \in N \tag{1}$$

$$R_M^u(n_i) = M(n_i) - \sum_{k \in K} \sum_{j \in \eth(k)} m(k_j), \quad \forall n_i \in N \tag{2}$$

where u represents the worker nodes (Cloud: c, Edge: e, PVs: $p \in P$), so $|N| = 2 + |P|$.

B. SYSTEM MODEL

In this paper, it is assumed that the latency is inversely proportional to the remaining capacity as the consequence of the M/M/1 queuing model.

1) CORE NETWORK OFFLOADING LATENCY

Network latency can be associated with data transmission and task execution time. The formal is highly correlated with the residual bandwidth of the link l_c, whereas the latter is based upon the availability of the core cloud to process the offloaded tasks and/or other network services. Hence, more tasks allocated to the remote cloud through l_c with less residual resources can dramatically raise the network latency, intensely in peak hours. In practise, the latency involved in accessing or writing on data volumes from/to memory can be overlooked thanks to the advanced technologies in datacenters. The offloading latency $t_c(l_c)$ to the cloud is consisted of the transmission, and the processing latency that can be calculated as follows:

$$t_c(k) = \max_{j \le |\eth(k)|} \{\frac{\chi_{k_j}}{\xi_c} + \frac{\chi_{k_j} f_{k_j}}{R_C^c(n_i)} + \frac{d}{v} + T_h\} \le t_m(k) \tag{3}$$

where ξ_c and f_k respectively denote the transmission rate of the server and CPU cycles utilizing for computing data per bit.

Consequently, the total amount of CPUs required for serving the task k are described as $\chi_k.f_k$. d, v, and T_h denote the cloud-to-edge distance, the light speed, and the constant time of processing a given task, respectively. Online tasks arrive at the edge through wired or wireless links (BS) in which the master node and edge server are resided. The management and control of the offloaded tasks at the edge can be indeed concerned as a local process; thus, the latency is primarily associated with the remaining computation capacity to host the task. Latency produced by processing a task is described as $T_h = \frac{\chi_k}{B_e \nu}$, in which the discount factor is ν that reflects the bandwidth fluctuations at the edge ($0 < \nu < 1$). Then, the offloading latency $t_e(k)$ at the edge is calculated as follows:

$$t_e(k) = \max_{j \in \eth(k)} \{\frac{\chi_{k_j} f_{k_j}}{R_C^e(n_i)} + T_h\} \le t_m(k) \tag{4}$$

2) PVs LATENCY

As opposed to the stable cloud or edge node, PVs can be defined as preemptible worker nodes for the sake of the erratic mobility. Thus, containerized task replicas would be a potential solution to diminish any service disruptions. It is assumed that the master node has enough time to reallocate the currently affected task replicas to other available nodes to maintain the QoS. In general, task replicas are allowed to reliably operate under a load-balancing mode in a normal state to enhance availability, performance efficiency, and reliability.

In this research, we basically leverage the LTE-A wireless connections between BS and PVs in respect to orthogonal

frequency-division multiple access (OFDMA) scheme as similar to [16]. $d_{bs,p}$ denotes the geographical distance between the BS and the p^{th} PV. The path loss between them is defined by $d_{bs,p}^{-\sigma}$ and white Gaussian noise power N_0, in which σ factor expresses the path loss exponent. Hence, the wireless channel can be modeled as a frequency-flat block-fading Rayleigh fading channel, denoted as h. As a result, the data rate of p^{th} PV is calculated as:

$$\xi_p = B_p log_2(1 + \frac{P_{TX}.d_{bs,p}^{-\sigma}.|h^2|}{N_0 + I}) \tag{5}$$

where B_p, P_{TX} and I are the channel bandwidth, transmission power of BS, and inter-cell interference, respectively. The offloading latency of PVs $t_p(k)$ is then formulated as follows:

$$t_p(k) = \max_{j \in \eth(k)} \{\frac{\chi_{k_j}}{E[\xi_p]} + \frac{\chi_{k_j} f_{k_j}}{R_C^p(n_i)} + T_h\} \leq t_m(k) \tag{6}$$

We also investigate the offloading efficiency of two types of online containerized tasks, comprising the latency-sensitive and latency-insensitive akin to [5]. While the latency-sensitive tasks are merely allocated to the edge nodes (e.g., edge server, PVs) due to their closest proximity, but the latency-insensitive tasks are processed at any worker nodes such as the remote cloud, edge server, or PVs. Then, we compute the cost for offloading an online task replica in our proposed collaborative computing architecture, which is associated with the sum of total CPUs, memory, bandwidth, and energy consumption for processing task replicas at PVs. In fact, the remote cloud and edge server achieve a high energy efficiency, so we do not consider this attribute in their costs. The offloading cost at the core cloud is defined as follows:

$$\Xi_{C_c}(k_j) = \frac{W_{C_c} \chi_{k_j} f_{k_j}}{C_c - \sum_{k' \in K_c} \sum_{j \leq |\eth(k')|} c(k'_j) + \delta} \tag{7}$$

$$\Xi_{M_c}(k_j) = \frac{W_{M_c} m(k_j)}{M_c - \sum_{k' \in K_c} \sum_{j \leq |\eth(k')|} m(k'_j) + \delta} \tag{8}$$

$$\Xi_{B_c}(k_j) = \frac{W_{B_c} \frac{\chi(k_j)}{t_m(k_j)}}{B_c - \sum_{k' \in K_c} \sum_{j \leq |\eth(k')|} \frac{\chi(k'_j)}{t_m(k'_j)} + \delta} \tag{9}$$

where δ is a small positive number to prevent dividing by zero. The offloading cost for processing a task replica at the core cloud is:

$$\Xi_{k_j}^c = \Xi_{C_c}(k_j) + \Xi_{M_c}(k_j) + \Xi_{B_c}(k_j) \tag{10}$$

As we discussed earlier, when a task is handled at the edge server, this is widely recognized as a local processing. Therefore, the offloading cost at the edge server is computed as follows:

$$\Xi_{C_e}(k_j) = \frac{W_{C_e} \chi_{k_j} f_{k_j}}{C_e - \sum_{k' \in K_e} \sum_{j \leq |\eth(k')|} c(k'_j) + \delta} \tag{11}$$

$$\Xi_{M_e}(k_j) = \frac{W_{M_e} m(k_j)}{M_e - \sum_{k' \in K_e} \sum_{j \leq |\eth(k')|} m(k'_j) + \delta} \tag{12}$$

Total offloading cost of a task replica at the edge is:

$$\Xi_{k_j}^e = \Xi_{C_e}(k_j) + \Xi_{M_e}(k_j) \tag{13}$$

Similarly, the cost of offloading a task replica and the energy consumption at a parked vehicle is formulated as follows:

$$\Xi_{C_p}(k_j) = \frac{W_{C_p} \chi_{k_j} f_{k_j}}{C_p - \sum_{k' \in K_p} \sum_{j \leq |\eth(k')|} c(k'_j) + \delta} \tag{14}$$

$$\Xi_{M_p}(k_j) = \frac{W_{M_p} m(k_j)}{M_p - \sum_{k' \in K_p} \sum_{j \leq |\eth(k')|} m(k'_j) + \delta} \tag{15}$$

$$\Xi_{B_p}(k_j) = W_{B_p} \frac{\chi_{k_j}}{t_m(k_j) \xi_p;}, \forall k \in K_p \tag{16}$$

$$E_p(k_j) = \chi_{k_j} f_{k_j} e_p \tag{17}$$

where e_p is a coefficient, which is attained by:

$$e_p = \epsilon (R_C^p(n_i))^2 \tag{18}$$

where ϵ denotes an energy coefficient.

Hereafter, total offloading cost of each task replica k_j at a parked vehicle is:

$$\Xi_{k_j}^p = \Xi_{C_p}(k_j) + \Xi_{M_p}(k_j) + \Xi_{B_p}(k_j) + \varsigma E_p(k_j); \tag{19}$$

where ς is an energy cost coefficient.

3) PVs' UTILITY

Owners of PVs are encouraged to share the idle computational resources while parking in parking lots, so they can gain accumulative rewards by hosting the task replicas in their vehicles. φ^p is the rewards by processing a task replica at a parked vehicle p. Thus, the corresponding utility is defined as follows:

$$\varpi^p = \varphi^p - \rho E_p(k_j) \tag{20}$$

where ρ denotes a coefficient of energy price, and φ^p is presented as:

$$\varphi^p = \mu r_p^c \chi_{k_j} f_{k_j} + r_p^m m(k_j) \tag{21}$$

where r_p^c and r_p^m are the unit prices of CPU and memory, respectively. We can see that minimizing the offloading cost can directly maximize the profits gained by hosting the according task replicas at PVs.

Variables:

$$\mathcal{A}_{k_j}^c = \begin{cases} 1, & k_j \text{ deployed on cloud, } \forall j \leq |\eth(k)|. \\ 0, & \text{otherwise.} \end{cases}$$

$$\mathcal{A}_{k_j}^e = \begin{cases} 1, & k_j \text{ deployed on edge, } \forall j \leq |\eth(k)|. \\ 0, & \text{otherwise.} \end{cases}$$

$$\mathcal{A}_{k_j}^p = \begin{cases} 1, & k_j \text{ deployed on a PV, } \forall j \leq |\eth(k)|. \\ 0, & \text{otherwise.} \end{cases}$$

Objective:

$$Minimize \sum_{j\leq|\eth k|} \Xi^c_{k_j}\mathcal{A}^c_{k_j} + \Xi^e_{k_j}\mathcal{A}^e_{k_j} + (\eta\Xi^p_{k_j} + (1-\eta)\frac{1}{\varpi^p})\mathcal{A}^p_{k_j}$$
$$w.r.t\ \mathcal{A}^c_{k_j}, \mathcal{A}^e_{k_j}, \mathcal{A}^p_{k_j} \quad (22)$$

Constraints:

$$\mathcal{A}^c_{k_j} + \mathcal{A}^e_{k_j} + \sum_{p\in N}\mathcal{A}^p_{k_j} = 1, \quad j \leq |\eth(k)| \quad (23)$$

$$1 \leq \sum_{j\leq|\eth(k)|}\mathcal{A}^p_{k_j} \leq \alpha * |\eth(k)| \quad (24)$$

$$\sum_{j\leq|\eth(k)|}\mathcal{A}^{c|e|p}_{k_j} * c(k_j) \leq C_{c|e|p} \quad (25)$$

$$\sum_{j\leq|\eth(k)|}\mathcal{A}^{c|e|p}_{k_j} * m(k_j) \leq M_{c|e|p} \quad (26)$$

$$\sum_{j\leq|\eth(k)|}\mathcal{A}^{c}_{k_j} * b(k_j) \leq B_c \quad (27)$$

$$b(k_j) \leq \xi_{n_i}, \forall n_i \in N \quad (28)$$

$$\sum_{j\leq|\eth(k)|}\mathcal{A}^{c|e|p}_{k_j} * t_{c|e|p} \leq t_m(k) \quad (29)$$

Remarks:

- Function (22) focuses on dual optimization objectives: optimizing both the offloading cost as well as PVs' rewards on which the task replicas are offloaded to PVs where η denotes a damping factor within (0,1).
- Constraint (23) makes sure that each task replica is merely processed at a single worker node.
- Constraint (24) determines that the proportion of task replicas offloaded to a PV cannot exceed 50% due to their uncertain mobility.
- Constraints (25),(26), (27), and (28) guarantee that the remaining capacity of the worker nodes (e.g., Cloud, Edge, PVs) must satisfy the rigid task demands.
- Constraint (29) eventually guarantees the chosen worker nodes must meet the latency constraint.

IV. OUR PROPOSED GENETIC ALGORITHM

A. BACKGROUND OF METAHEURISTIC ALGORITHMS

An optimization process is technically a kind of process that approaches better and better solutions by searching and comparing feasible ones until it cannot achieve a better result [22]. Nevertheless, the major optimization aim is to come up with an optimal solution meeting a set of predefined objectives, and is expected to conciliate multiple stringent constraints. Meta-heuristics include a set of optimization techniques that efficiently discover feasible solutions, aiming at achieving the global optimum. Intrinsically, meta-heuristic algorithms carry out diversified variation operations to explore new potential solutions effectively, and their multi-objective fitness function will drive such potentials to the optimum. Even designing an efficient algorithm satisfying several desired constraints is not an easy task, meta-heuristics have been successfully employed in various applications from different fields, including operation research, industrial engineering to management science. Evolutionary Computation (EC) that imitates the natural selection evolutionary concepts includes meta-heuristic algorithms to search for globally optimal solutions. EC techniques provide flexibility, adaptability, and importantly an exceptional performance. They are effective when the search space is huge with a large number of involved parameters.

In fact, a mature meta-heuristic GA algorithm is inspired by the Darwin's theory of evolution principle through the natural selection, has been the most common population-based meta-heuristic algorithms. GA is able to solve either linear or non-linear programming optimization problems with multiple objectives. Thanks to the simpleness and straightforward deployments, GA is fast. It has been proved to be more efficient than many conventional heuristic algorithms by efficiently maintaining a balance between exploration and exploitation through a proper set of parameters. GA is recently recognized as a scalable alternative to reinforcement learning (RL) algorithm [23], [24] over competitive performance outcomes. Although GA cannot always perform better than RL as shown in [25] and [26], GA algorithm is indeed faster than the counterpart since GA exposes its greater scalability as well as parallelism. Furthermore, GA is practically acknowledged as a parallel search [27] with mutual independency amongst multiple exclusively feasible solutions.

GA can produce a set of ''good'' solutions in lieu of a single solution. Those are able to be evolved over several iterations, driven by an efficient fitness function. A typical Genetic Algorithm comprises four major operations: population initialization, selection, crossover, and mutation [28]. At each iteration, GA chooses two individuals randomly in the generated population as parents to create their children (also known as offspring) for the next generation. As the nature of the selection process, if good parents are selected for generating new generations, their offspring is most likely to be good through good characteristics inherited from their parents. This somehow guarantees good solutions produced. Over generated generations, the population can eventually get evolved, so that the chance of approaching an optimal solution is remarkably increased. In detail, GA is first generating an initial population randomly. Each feasible solution in the population, widely acknowledged as a chromosome, will quantify its quality by the fitness function. Then, two chromosomes are deterministically or randomly chosen as parental individuals in the selection operator. Accordingly, these chromosomes enable the production of their offspring by interchanging the partial genes at a random point, widely known as crossover operation. The next operation is called mutation that applies a small random tweak to a chromosome, deployed on a randomly selected chromosome with a random position to produce a new solution. Moreover, the mutation is expected to consider an exploration on the searching space

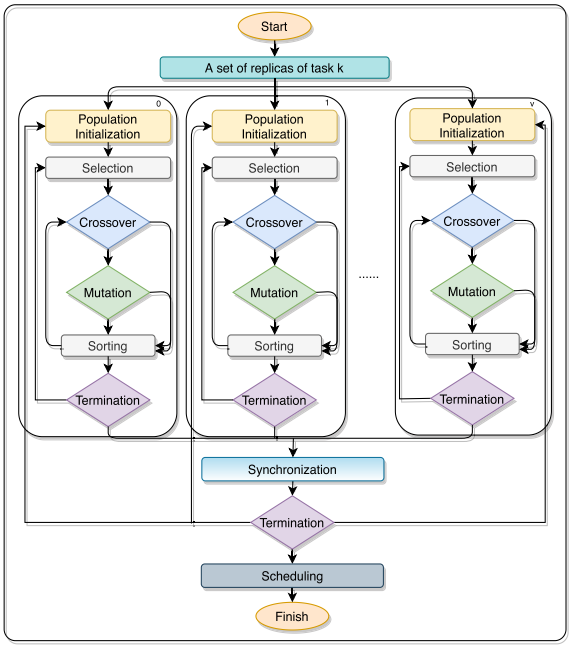

FIGURE 3. Distributed and parallel GA-based implementation.

while retaining the population diversity. GA operations can be rerun until the pre-defined stopping condition is met (e.g., several iterations). Lately, parallel computing is a promising paradigm to efficiently deal with the complicated problems with huge time-saving and lower cost guarantees by enabling concurrency.

B. DISTRIBUTED PARALLEL GENETIC ALGORITHM

For solving the BIP problem, a distributed and parallel GA-based algorithm running on multiple independent machines, denoted as V is proposed to discover the search space in this paper. The operational implementation of our proposed algorithm is demonstrated in Fig 3 where $|V|$ is defined to be 16. The working scheme includes a master node that primarily plays a synchronization role, and several distributed slave nodes exploring as many feasible solutions as they can. At each slave node, GA iteratively deploys its few operators to seek for the feasible offloading solution. The best outcomes based on the fitness values among the distributed parallel nodes are then synchronized by the master node to identify the optimal offloading solution for the given task. Our proposed algorithm in this research is permitted to offload multiple task replicas at once rather than allocating each replica sequentially.

1) GENETIC REPRESENTATION AND SELECTION

a: CHROMOSOME

GA's a chromosome denoted as $\mathcal{C}_f$ in this paper indicates a solution for the whole set of requested replicas of a given task, which is randomly chosen from the available worker nodes meeting task resource demands. Hence, each gene within the chromosome represents an offloading solution for a single task replica, which is described as $g_f^j = \mathcal{A}_{k_j}^{c,f}\mathcal{A}_{k_j}^{e,f}\mathcal{A}_{k_j}^{1,f}\cdots\mathcal{A}_{k_j}^{|P|,f}$. If G is the number of genes, so $G = |\eth(k)|$. The evolutionary process is started with M chromosomes, so that the initial population is created as follows:

$$\mathcal{P} = \begin{bmatrix} \mathcal{C}_1 \\ \mathcal{C}_2 \\ \vdots \\ \mathcal{C}_f \\ \vdots \\ \mathcal{C}_M \end{bmatrix} = \begin{bmatrix} g_1^1 & \cdots & g_1^j & \cdots & g_1^G \\ g_2^1 & \cdots & g_2^j & \cdots & g_2^G \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ g_f^1 & \cdots & g_f^j & \cdots & g_f^G \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ g_M^1 & \cdots & g_M^j & \cdots & g_M^G \end{bmatrix} \tag{30}$$

In fact, the chromosome, formed by G genes passing a feasibility check, is established as a feasible offloading solution for a given task with a set of requested replicas.

b: SELECTION

The selection operation essentially determines which chromosomes to become parents for the crossover operator. To enhance the parallelism, parents can be randomly selected from the initial population with a replacement. Due to the nature of randomness, the quality of children, that are generated in the crossover operator, can be better or even worse than their parents. In theory, there are several selection strategies, but the fitness-based proportionate designation relied upon the accumulative sum of fitness-relative weights is usually preferable in this operator.

c: FITNESS FUNCTION

The major goals of our proposed algorithm include optimizing the cost of offloading online tasks and maximizing the user rewards when the task is offloaded to PVs. To achieve these dual objectives, fitness function is utilized to evaluate the quality of an offloading solution, and a better solution could produce higher fitness values in this paper.

$$\mathcal{F}(k) = \sum_{j\in\eth k} \frac{1}{\Xi_k^c}\mathcal{A}_{k_j}^c + \frac{1}{\Xi_k^e}\mathcal{A}_{k_j}^e + ((1-\eta)\frac{1}{\Xi_k^p} + \eta\varphi_k^p)\mathcal{A}_{k_j}^p \tag{31}$$

2) THE PROCESSES OF EVOLUTION

After initial population is generated in the initialization operator, two chromosomes are selected in random to be parents. Then, new generations are formed by an evolutionary process including the crossover and mutation operators. To maintain the population diversity, the newly generated generations are updated into the existing population. This strategy is able to improve the opportunity to obtain near-optimal task offloading solutions.

a: CROSSOVER

This is considered as the most vital operator to create new offspring by stitching up the parental chromosomes in GA. Suppose $\mathcal{C}_s$ and $\mathcal{C}_r$ are two parental chromosomes that have particular indexes s and r in the initial population.

Denote j^c as a random crossover point within N length, their corresponding descendants are $\mathcal{C}_{(M+1)}$ and $\mathcal{C}_{(M+2)}$. By exchanging genes beginning from the crossover point $j^c + 1$ to the last gene between the parents, new generations are generated as below:

$$\mathcal{P} = \begin{bmatrix} \mathcal{C}_1 \\ \vdots \\ \mathcal{C}_s \\ \vdots \\ \mathcal{C}_r \\ \vdots \\ \mathcal{C}_M \\ \mathcal{C}_{M+1} \\ \mathcal{C}_{M+2} \end{bmatrix} = \begin{bmatrix} g_1^1 & \cdots & g_1^{j^c} & g_1^{j^c+1} & \cdots & g_1^G \\ \vdots & \ddots & \vdots & \ddots & \vdots & \\ g_s^1 & \cdots & g_s^{j^c} & g_s^{j^c+1} & \cdots & g_s^G \\ \vdots & \ddots & \vdots & \ddots & \vdots & \\ g_r^1 & \cdots & g_r^{j^c} & g_r^{j^c+1} & \cdots & g_r^G \\ \vdots & \ddots & \vdots & \ddots & \vdots & \\ g_M^1 & \cdots & g_M^{j^c} & g_M^{j^c+1} & \cdots & g_M^G \\ g_s^1 & \cdots & g_s^{j^c} & g_r^{j^c+1} & \cdots & g_r^G \\ g_r^1 & \cdots & g_r^{j^c} & g_s^{j^c+1} & \cdots & g_s^G \end{bmatrix} \quad (32)$$

b: MUTATION

This operator applies a small modification on the current parent to form new offspring/chromosome. The mutation stage allows to sample the large search space, improving the searching efficiency. This operation is widely known a primary component in the evolutionary process, which prevents potential solutions from falling into the local optima. Technically, a new gene selected randomly replaces an existing one within one of children produced in Crossover operator to create a new offspring. The gene must inevitably satisfy the resource demands to survive through the feasibility check. If both children are infeasible in Crossover, one of parents chosen in random is then used for mutation. Suppose j^m is a random mutation point and $g_{r'}^{j^m}$ is a new gene that substitutes the existing gene within $\mathcal{C}_{(M+1)}$. Consequently, new offspring generated from the mutation stage is $\mathcal{C}'_{(M+1)} = [g_s^1 \cdots g_{r'}^{j^m} \cdots g_s^G]$.

To maintain a balance between exploitation and exploration in GA algorithm, crossover rate p_c is typically set higher than mutation rate p_m. Determining p_m is never an easy task since small mutation rate leads to premature convergence while high mutation rate could improve the exploration process in the search space, but this selection might prevent GA algorithm from converging to optimal solution. By preferring high efficiency of GA while keeping a trade-off between exploitation and exploration, we set $p_c = 0.9$ and $p_m = 0.2$ in this paper.

3) TERMINATIONS AND SYNCHRONIZATION

Parallel processing is associated with multiple concurrent processes in which each process might accomplish its assignment at a different time. Unfortunately, waiting for all tasks to completely finish their assigned jobs is painful due to the fact that one or more tasks might take too much time for processing (e.g., deadlock). Thus, to reduce the overall execution time, the master node will terminate GA algorithms running at worker nodes if there is no better

Algorithm 1 EdgeGA - An Intelligent GA-Based Algorithm

1: **Input**:
2: An online task k with five tuples $\{c(k), m(k), b(k), t_m(k), \eth(k)\}$
3: **Output**:
4: A list of worker nodes hosting task replicas.
5: **procedure** task offloading
▷ **Step 1:** *Generate a list ζ_k of node candidates including cloud, edge, PVs*
6: **function** `GET_CANDIDATES` (k)
7: **empty** ζ_k
8: **for** all $n_i \in N$ **do**
if $R_C^u(n_i) \geq c(k)$, $R_M^u(n_i) \geq m(k)$, $R_B^u(n_i)$ or $\xi_p \geq b(k)$ **then**
9: **add** n_i to ζ_k
end for
11: **return** ζ_k
if (none of worker nodes are available) **then**
12: **reject** the task k
13: **end function**
▷**Step 2:** *Deploy Genetic Algorithm in a distributed parallel operation scheme*
14: **call Algorithm 2**
▷**Step 3:** *Synchronize all incumbents obtained in independent working machines*
15: Choose the best solution relied upon the sum of fitness values (E.q. 31)
16: **return** the list of worker nodes $n_i \in N$
▷ **Step 4:** *Update SN resources*
17: **end procedure**

solutions obtained within t times. Eventually, the feasible solutions found from several slave machines is finalized through a synchronization in order to choose the optimal offloading solution relied upon fitness values. If accepted, task replicas of the given task are then allocated to the worker nodes following the information of the achieved offloading solution. SN eventually updates the rnetwork resources to finish the offloading processes.

The technical details of the proposed GA-based algorithm are provided in Algorithm 1 and 2. When an online task including a number of required replicas arrives, the algorithm creates a list of potential node candidates which must meet resource requirements of the given task demands (e.g., CPU, memory, bandwidth, delay) as shown in lines [6-13]. GA is then implemented in a single working machine in a distributed parallel operation scheme in order to seek the best offloading solution for a given task by calling Algorithm 2 in line 14. Lines [15-16] are the synchronization process that selects the optimal offloading solution among the outcomes of the parallel machines, and eventually updating the network information status in Step 4. In terms of Algorithm 2, lines [4-13] are associated with population initialization where each chromosome is randomly generated from the list of node candidates. By selecting parents from the population

Algorithm 2 GA Runs at Each Paralleled Machine

1: **Input**: ζ_k
2: **Output**: The best offloading solution for the task k
3: **procedure** Genetic Algorithm operations
▷*Initial Population Generation*
4: $r = 0$
5: **for** $m = 1$ to M **do**
▷ *Generate a chromosome with* $|\eth(k)|$ *genes. Each gene is a task offloading solution for a replica* $g_f^j = \mathcal{A}_{k_j}^{c,f} \mathcal{A}_{k_j}^{e,f} \mathcal{A}_{k_j}^{1,f} \cdots \mathcal{A}_{k_j}^{|P|,f}$
6: **for** $n = 1$ to $|\eth(k)|$ **do**
▷ *Try to map a task replica to a randomly selected worker node in* ζ_k *with up to Q trials*
7: **for** $q = 1$ to Q **do**
8: Map a replica to a randomly selected worker node in ζ_k
9: **if** feasible **goto** 11
10: **end for**
11: **end for**
12: $r = r + 1$ and add the chromosome to population
13: **end for**
▷*Evolution process*
14: **if** $r > 1$ **then**
15: **for** $p = 1$ to *maxIterations* **do**
16: **if** $ranNum \in (0, 1) < p_c$ **then**
17: **Select** two parents in random
18: **Conduct** crossover operation
19: **if** both parents are feasible **then**
20: One of children is randomly chosen for mutation
21: $r = r + 2$ and **add** them to population
22: **else**
23: **if** only a child is feasible
24: $r = r + 1$ and **add** the one to population
25: **if** $ranNum \in (0, 1) < p_m$ **then**
26: **if** both children in crossover are infeasible **then**
27: One parent is randomly selected for mutation
28: **end if**
29: **Conduct** Mutation operation
30: **if** new mutated child is feasible **then**
31: $r = r + 1$ and **add** the one to population
32: **end for**
33: **if** $r > M$ **eliminate** chromosomes produced lower fitness values
34: **else if** $r = 1$ **then** the current offloading solution will be final.
35: **else**
36: **reject** the task k
37: **end if**
38: **end procedure**

in random as shown in line 17, we try to balance the exploration and exploitation. Lines [14-37] involve the GA's evolution operations by exploring the searching space. The Crossover operator is conducted in lines [16-24], whereas lines [25-31] are the Mutation operator. Line 33 is to maintain the elite population by eliminating the chromosomes producing the lowest fitness values to remain the population at most M chromosomes. In case that we only achieve one feasible chromosome (e.g., due to network congestion), this will become the final offloading solution in line 34; otherwise, the task will be rejected in line 36.

C. EXECUTION TIME ANALYSIS

Due to the lower cost of computing hardware recently, parallel algorithms can be beneficially exploited to tackle intricate computational tasks. As a result, we advise a distributed parallel GA framework to deal with the online task offloading problem. In this paper, the execution time of the proposed task offloading solution is measured in two manners: sequential and parallel modes. In sequential mode, the time complexity follows a linear increase as we can see that the execution time is the sum of the operation time of all working machines. However, the total execution time of the parallel mode is estimated at the latest machine that finishes its offloading assignment. The time complexity of GA algorithm at each machine is roughly $O(G \times M \times maxIterations)$. In fact, the representation of GA algorithm at each working machine is not static, which is depended on the number of replicas of a given task. In addition, we cannot always guarantee to achieve M chromosomes when the SN becomes increasingly congested. In GA algorithm, the iteration process is terminated earlier if the best fitness value does not change for t consecutive iterations. It would be better to measure the time complexity by measuring the average runtime and to indicate how the parallel manner is enhanced in a comparison with sequential one.

Similar to [29], we apply Cramer-Chernoff technique and Jensen's inequality to provide a reasonable approximation to the total execution time of the parallel mode. Hence, our distributed parallel offloading framework is able to indeed enhance the time complexity from linear to logarithmic scale subject to $|V|$. Interested readers may refer to [29] further theoretical analysis.

V. COMPARED ALGORITHMS

We evaluate the efficiency of not only our proposed collaborative framework compared with conventional computing paradigms including cloud and edge computing, but also our GA-based algorithm in a comparison with some heuristic algorithms, comprising Baseline_1, Baseline_2, and Baseline_3 towards the acceptance ratio, offloading cost, and utility. Baseline_1 is considered as a Kubernetes default scheduler applying the filtering and then scoring algorithms, whereas Baseline_2 processes the task replicas by randomly selecting the worker nodes. In contrast, Baseline_3 deploys a branch and bound strategy to tackle the given task with a set of replicas sequentially [30]. Different from these heuristics, our proposed GA-based solution enables a set of all task replicas to be processed at once. To remain the service stability and reliability, a proportional number of replicas can be solely offloaded to a single worker node, which cannot exceed 50% (except cloud and edge nodes). Indeed, SPs is able to easily adjust this parameter to meet their specific goals (e.g., in network congestion). Several performance

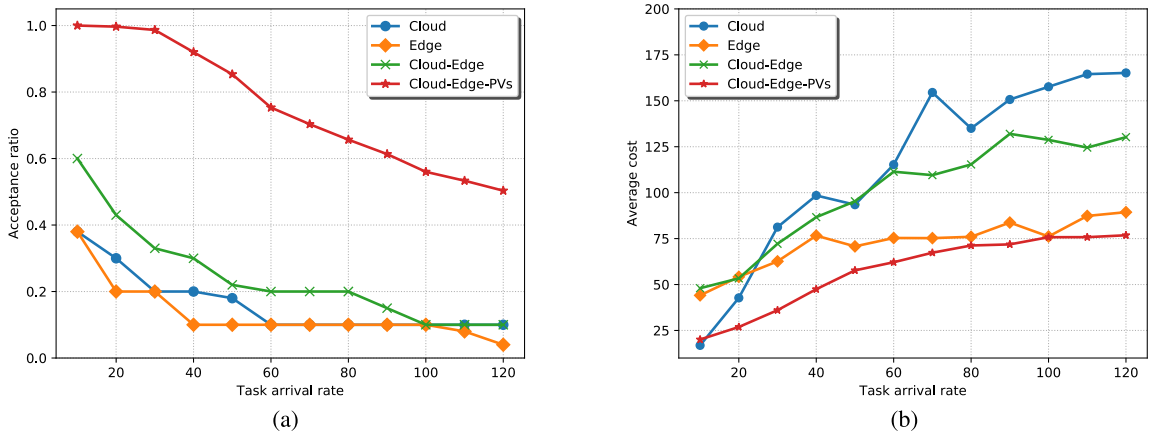

FIGURE 4. (a) Acceptance ratio (b) Average costs between architectures.

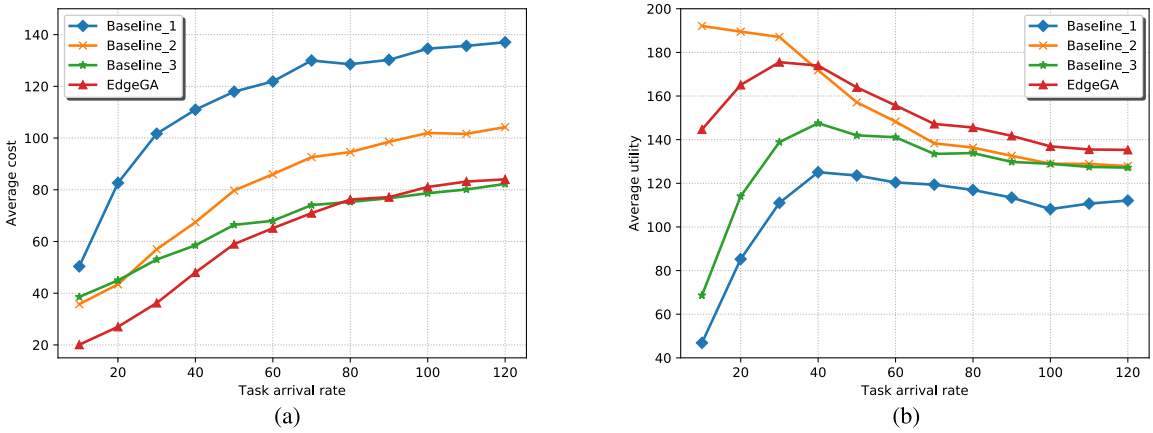

FIGURE 5. (a) Average offloading cost (b) Average utility.

evaluation metrics including average task acceptance ratios, average offloading costs, and average accumulated utility are conducted to assess the efficiency amongst the compared algorithms. Besides, we extend our assessment by comparing our proposed GA-based algorithm on different parking-lot sizes: 50 and 100 parking spots according to small and medium ones. The offloading results of them are crucial to determine the possible network strategies towards SPs in order to guarantee QoS or Key Performance Indicators (KPIs).

VI. NUMERICAL RESULTS

A. SIMULATION SETUP

In this paper, we have evaluated the algorithms by developing a discrete event simulator. Vehicles dynamically arrive at and leave parking lots which have 50 or 100 free parking spots. In practice the whole parking lot might be fully utilized, but we set out the capacity of the parking lot merely ranging from 50% up to 85% in peak hours. It can be argued that not all PVs are ready to share the computation resources while parking or meet the essential qualifications to provision the network services (e.g., outdated vehicles, lacking computing capability, running errands). In addition, [1] indicated that the parking duration of PVs is analytically varying

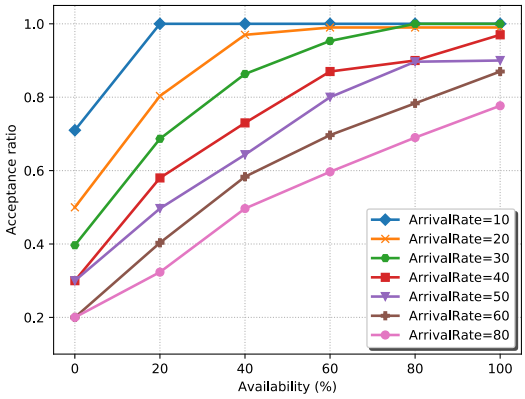

FIGURE 6. Acceptance ratio towards PV availability.

[08-240] minutes. The service behaviours of PVs in [1] was estimated from the real dataset provided by ACT Government Open Data Portal dataACT. The SmartParking application was installed to collect more than 180, 000 parking records in the Manuka shopping precinct in Canberra, Australia. Following these statistics, it is pointed out that more than 85% of PVs approximately spent maximum average 3 hours in the parking lot, and the probability of serviceability of PVs gains around 90% at 60 minutes [1]. In this research, the

TABLE 2. Simulation parameter settings.

Parameter	Values
Maximum parking capacity	50 and 100
Total simulation time	30,000 seconds
Vehicle lifetime	[480-14400] seconds
Cloud CPU capacity C_c	50 GHz
Cloud Memory capacity M_c	1000 MB
Cloud Bandwidth B_c	1.0 Gbps
Edge CPU capacity C_e	20 GHz
Edge Memory capacity M_e	500 MB
$W_{\{C_c,M_c,B_c\}}$	750
$W_{\{C_e,M_e\}}$	250
$W_{\{C_p,M_p,B_p\}}$	10
Channel Bandwidth B_p	10 MHz
P_{TX}	1.3W
N_0	3×10^{-13}
σ	2
CPU Parked Vehicles C_p	[1.5-2.0] GHz
Input data size χ_k	[100 - 300] kb
CPU cycles per bit f_k	1000 cycles
Memory requests $m(k)$	[20-50] MB
Tolerable latency of tasks t_m	(0-100] ms
Arrival request rates	[10-120]
Request replications $\eth(k)$	[2 - 10]
r_p^c	10
r_p^m	100

parking duration of PVs follows the Poisson distribution with $\lambda = 3600$. The simulation runs for almost 8 hours following the common pattern of business working time in peak hours, and the simulator indeed updates the PV availability for every 20 minutes. As mentioned, the online tasks can be commonly divided into latency-sensitive and latency-insensitive tasks; thus, when the latency tolerance of a task exceeds 20 ms, it is marked as a latency-sensitive request. In this paper, the offloading task requests arrive in the network following the Poisson process with an average rate varying from 10 to 120 requests per 100 time units. Each online task request has an exponentially distributed lifetime with an average of $\mu = 1200$ time units. These workloads are extremely extensive for evaluating the proposed framework as well as the compared algorithms. Besides, energy coefficient ϵ, coefficient for energy price ρ and unit price for each CPU cycle σ are set to 10^{-24}, 0.003 and 2×10^{-9} [17], respectively. Other simulation parameters are detailed in Table 2.

B. PERFORMANCE RESULTS

In terms of the small-size parking lot, evaluation results are shown in Fig. 4, 5 and 6, whereas those of medium-size parking lot are illustrated in Fig. 7 and 8. Finally, Fig. 9 depicts the execution time of the proposed GA-based algorithm measured in different sizes of the parking lots. Fig. 4a indicates that our collaborative paradigm remarkably enhanced the average acceptance ratio for more than 40% in comparison with Cloud-Edge and Cloud infrastructures at the arrival rate of 120, respectively. Additionally, Cloud or edge infrastructure performed worse due to their limited computation capacity in peak hours. Moreover, our proposed collaborative framework significantly saved the offloading cost when being compared to other infrastructures as demonstrated in Fig. 4b. These results in Fig. 4a and 4b come from the facts that the collaborative framework exploited not only typical cloud and edge computing capacities, but also those of available PVs, allowing more online task requests processed. It also deployed the GA-based algorithm for optimizing the offloading cost, so the proposed infrastructure produced less cost compared to others. Similarly, cloud-Edge enabled the computing resources of both cloud and edge computing, which helps cloud-edge paradigm perform better than separate cloud or edge paradigm which could only utilize their own capacity separately. However, due to its merged computing capacities, cloud-edge framework generated more offloading cost than others except cloud infrastructure. The core cloud indeed processed more tasks than the edge computing due to its larger computation capacity, but it also bore more cost than the edge.

In Fig. 5a, Baseline_1 performed worst in terms of the average offloading cost because of its offloading strategies with a simple heuristic filtering and scoring algorithm. Baseline_1 first carried out the filtering procedure to select the feasible nodes that met the task requirements, then cored them based on their current properties (e.g, computing resources). The node with the highest scores that was matched the task demands was selected. It did not take any offloading cost or utility factor into account. In contrast, Baseline_2 was based upon the random mechanism for selecting the worker nodes and, had a better performance than Baseline_1. And, Baseline_2 tended to perform well when the network was less congested; thus, it had more options for preference. Baseline_3 was primarily aimed at optimizing the offloading cost, so it performed best amongst the baseline algorithms; and its performance was indeed very comparative to the proposed GA-based algorithm.

Fig. 5a is revealed that EdgeGA's performance was still better than Baseline_3 prior to the arrival rate 80, and performed slightly similar afterwards. It is because the online tasks were most likely offloaded to PVs, producing lower offloading cost. Towards utility as depicted in Fig. 5b, EdgeGA defeated the heuristics following Baseline_2, Baseline_3, and Baseline_1, respectively. In fact, EdgeGA took the offloading cost as well as the utility into account, driven by the efficient fitness function (31), while the baseline algorithms did not consider utility in their node selection strategies. In Fig. 5b, Baseline_2 performed better than EdgeGA before the arrival rate of 40 because Baseline_2 had more node options for offloading the tasks when the network was less congested; however, starting from the arrival rate of 40 afterwards, EdgeGA outperformed all compared algorithms since EdgeGA smartly searched for the most appropriate worker nodes that were able to produce

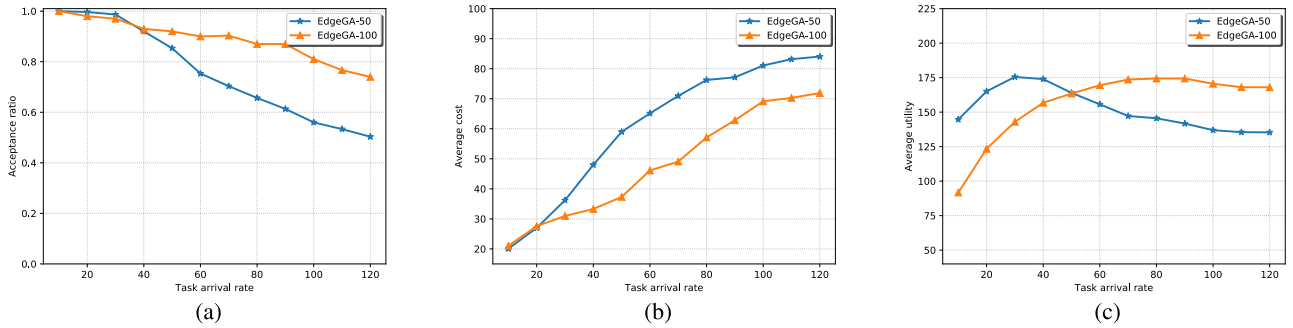

FIGURE 7. EdgePV performance on different sizes: (a) Acceptance ratio (b) Average cost (c) Average utility.

less cost while generating highest revenues, especially in congested environments.

Furthermore, we figured out the PV availability of PVs in relation to acceptance ratios subject to different arrival rates of online tasks, where each arrival rate prefers different PV availability as demonstrated in Figure 6. For precise measurements, we fixed the availability of PVs instead of letting the parking capacity randomly ranging from 50% up to 85% as described in Section VI-A. There are several useful information revealed from these results. For instance, arrival rates (10, 20, 30, 40, 50) demands 60% availability of PVs to reach 80% acceptance ratios, whereas the arrival rates 60 and 80 were required 80% and 100% to gain the same outcome, respectively. These evaluations are critical to the network planners for attaining the expected KPIs by conducting proper strategies. For example, SPs could increase user incentives to appeal PVs to join into the network (e.g., reaching full capacity), or to extend edge server capacity, or to offload to another cluster. Depending on particular situations, SPs can determine which strategy is the most appropriate.

In addition, we assessed our proposed GA-based algorithm on different sizes of the parking lot: a small size with 50 free parking spots (denoted as EdgeGA-50) which has been done in previous section, a medium one with 100 free parking spots (denoted as EdgeGA-100). Both were run on the same network loads ranging from 10 to 120 requests per 100 time units as similar to previous section. The main reason for this study are threefold: first we want to examine the scalability of our proposed algorithm, and how well EdgeGA adapts to the increase of the search space. Second, conducting this evaluation can quantify how much gains exactly we can achieve if the capacity of parking lot is doubled. Further, this study can provide a flexible offloading strategy for SPs who can statistically determine which proper parking lots to host their services on them. For example, SPs can select a medium size of parking lot at first since they anticipate that the network can be quickly congested with large traffic loads, and then switch to smaller size in off-peak times or vice versa in order to balance between QoS and generated revenues. Additionally, they might decide to choose one medium parking lot instead of two small ones depending on the workloads.

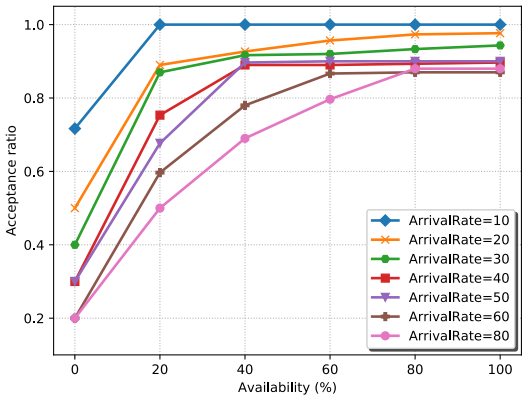

FIGURE 8. Acceptance ratio regarding PV availability with 100 PVs.

Fig. 7a and 7c depicted that both acceptance ratio and average utility were improved up to 24% at the arrival rate 120 when the parking-lot size was increased to double. Additionally, the offloading cost was also enhanced for more than 16% at the same arrival rate as shown in Fig. 7b. It is obvious to recognize that when increasing the parking lot size, there were more options for node selections, leading to the increasing possibility of accepting more arrived tasks while maintaining lower offloading cost. Similarly, the average utility was also enhanced when the workloads were increasing. However, with smaller size of parking lot during low network congestion, EdgeGA rapidly achieved a good result for average utility, which was consistent with the results in Fig. 7a since EdgeGA had to search a smaller search space. When the network became more and more congested with increasing workloads, specifically after the arrival rate of 50, EdgeGA still proved its efficiency in congested environments. Due to less worker nodes for selections, EdgeGA-50 performed worse than EdgeGA-100 that had more available worker nodes for selections. Thanks to those performance outcomes, IPs are able to determine which size of parking lots to maintain a balance between the generated revenues and the offloading cost.

Likewise, we investigated the relationship between the acceptance ratios and the availability of PVs as shown in Fig. 8. To produce 80% of the acceptance ratio for all arrival rates, 60% availability of PVs were demanded against 100% of a small-size parking lot. We also fixed the

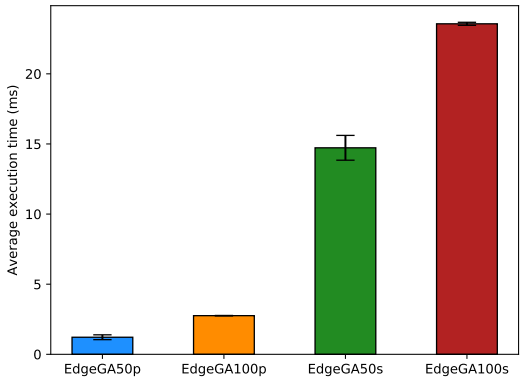

FIGURE 9. Average execution time on sequential and parallel modes.

availability of PVs for precise measurements as explained in previous section. In fact, 10, 20 and 30 arrival rates quickly achieved 80% acceptance ratio when the availability of PVs was smaller than 20%, whereas the arrival rates 40, 50 and 60 demanded 40% to obtain the similar outcomes. Hereafter, the arrival rate 80 required larger than 20% to achieve the same result. Compared to the results in Fig. 6, it is reasonable for these improvements as the size of parking lot was expanded, which means that there were more worker nodes hosting the arrived tasks so the acceptance ratio was improved. As we mentioned in Section IV, we propose a distributed parallel GA-based algorithm for online task offloading problem in this paper. To measure the improvement of the proposed parallel implementation scheme towards reducing execution time, we compare the execution time of our GA-based algorithm running on sequential and parallel manners separately. Due to diverse workloads with several arrival rates, similar to several related papers, the execution time in this manuscript was measured for a single offloading task. As a result, our proposed parallel framework only required 1.217*ms* compared to 14.725*ms* in sequential operation manner to successfully process a given task in average as demonstrated in Fig. 9.

By increasing the size of the parking lot to double, the algorithm finished processing a task in 2.756*ms* and 23.567*ms* regarding parallel (_p) and sequential (_s) modes, respectively. The exceptional performance on the execution-time was due to the distributed parallel implementation of our proposed GA-based algorithm as shown in Fig. 3. The achieved execution time is indeed ambitious, which again proves that our GA-based solution is fast, efficient and practical.

VII. CONCLUSION

This paper has studied the collaborative computation architecture where PVs are promising to become an efficient extension for the existing cloud-edge computation paradigm to deal with the online task offloading in peak business hours. We also advocate the Kubernetes orchestrator that can be implemented at the edge server as the master node. Accordingly, the core cloud, edge computing itself, and PVs are able to manipulate as worker nodes. The extensive evaluations shows that our collaborative infrastructure remarkably increases the computational capability of the existing computing architecture by efficiently making use of the being-wasted powerful hardware of PVs. This novel framework also gives a flexibility, agility, and reliability to address the online task offloading problems. In addition, we propose a GA-based algorithm to deal with the time complexity of BIP problem and then compare our solution with several baseline algorithms as well as on different sizes of the parking lots. The proposed GA-based algorithm outperformed all compared heuristics in terms of several important performance metrics such as task acceptance ratios, offloading cost, and accumulative rewards. Furthermore, PV owners are able to gain extra incentives by sharing their computing resources while parking in parking lots. Furthermore, we quantify the successfully task acceptance ratios towards the availability of PVs on various arrival rates. In fact, the evaluations are critical to SPs for making a proper decision on which offloading strategies are selected to optimize the generated revenues. The proposed GA-based algorithm dramatically improved the average total execution time thanks to the distributed parallel implementation when being compared with the sequential operation.

ACKNOWLEDGMENT

The authors would like to thank their academic editor and anonymous reviewers for their careful reading of their manuscript and many insightful comments and suggestions.

REFERENCES

[1] X. Huang, R. Yu, J. Liu, and L. Shu, "Parked vehicle edge computing: Exploiting opportunistic resources for distributed mobile applications," *IEEE Access*, vol. 6, pp. 66649–66663, 2018.
[2] F. H. Rahman, A. Y. M. Iqbal, S. H. S. Newaz, A. T. Wan, and M. S. Ahsan, "Street parked vehicles based vehicular fog computing: TCP throughput evaluation and future research direction," in *Proc. 21st Int. Conf. Adv. Commun. Technol. (ICACT)*, Feb. 2019, pp. 26–31.
[3] D. Han, W. Chen, and Y. Fang, "A dynamic pricing strategy for vehicle assisted mobile edge computing systems," *IEEE Wireless Commun. Lett.*, vol. 8, no. 2, pp. 420–423, Apr. 2019.
[4] M. Sigalos. (Jan. 2022). *This Tesla Owner Says he Mines up to $800 a Month in Cryptocurrency With his Car*. [Online]. Available: https://www.cnbc.com/2022/01/08/tesla-owner-mines-bitcoin-ethereum-with-his-car.html
[5] O. Fadahunsi and M. Maheswaran, "Locality sensitive request distribution for fog and cloud servers," *Service Oriented Comput. Appl.*, vol. 13, no. 2, pp. 127–140, Jun. 2019, doi: 10.1007/s11761-019-00260-2.
[6] Y. Zhao, W. Wang, Y. Li, C. C. Meixner, M. Tornatore, and J. Zhang, "Edge computing and networking: A survey on infrastructures and applications," *IEEE Access*, vol. 7, pp. 101213–101230, 2019.
[7] K. Nguyen, S. Drew, C. Huang, and J. Zhou, "Collaborative container-based parked vehicle edge computing framework for online task offloading," in *Proc. IEEE 9th Int. Conf. Cloud Netw. (CloudNet)*, Nov. 2020, pp. 1–6.
[8] *Lightweight Kubernetes*. Accessed: May 10, 2021. [Online]. Available: https://k3s.io/
[9] (2020). *What is Kubernetes?* Accessed: May 28, 2020. [Online]. Available: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/
[10] S. Arif, S. Olariu, J. Wang, G. Yan, W. Yang, and I. Khalil, "Datacenter at the airport: Reasoning about time-dependent parking lot occupancy," *IEEE Trans. Parallel Distrib. Syst.*, vol. 23, no. 11, pp. 2067–2080, Nov. 2012.
[11] W. He, G. Yan, and L. D. Xu, "Developing vehicular data cloud services in the IoT environment," *IEEE Trans. Ind. Informat.*, vol. 10, no. 2, pp. 1587–1595, May 2014.

[12] F. Dressler, P. Handle, and C. Sommer, "Towards a vehicular cloud—Using parked vehicles as a temporary network and storage infrastructure," in *Proc. ACM Int. Workshop Wireless Mobile Technol. Smart Cities (WiMobCity)*. New York, NY, USA: Association for Computing Machinery, 2014, pp. 11–18, doi: 10.1145/2633661.2633671.
[13] E. Al-Rashed, M. Al-Rousan, and N. Al-Ibrahim, "Performance evaluation of wide-spread assignment schemes in a vehicular cloud," *Veh. Commun.*, vol. 9, pp. 144–153, Jul. 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2214209616301863
[14] T. Kim, H. Min, and J. Jung, "Vehicular datacenter modeling for cloud computing: Considering capacity and leave rate of vehicles," *Future Gener. Comput. Syst.*, vol. 88, pp. 363–372, Nov. 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X18300487
[15] C. Li, S. Wang, X. Huang, X. Li, R. Yu, and F. Zhao, "Parked vehicular computing for energy-efficient Internet of Vehicles: A contract theoretic approach," *IEEE Internet Things J.*, vol. 6, no. 4, pp. 6079–6088, Aug. 2019.
[16] S. Raza, W. Liu, M. Ahmed, M. R. Anwar, M. A. Mirza, Q. Sun, and S. Wang, "An efficient task offloading scheme in vehicular edge computing," *J. Cloud Comput.*, vol. 9, no. 1, p. 28, Jun. 2020, doi: 10.1186/s13677-020-00175-w.
[17] Y. Cao, Y. Teng, F. R. Yu, V. C. M. Leung, Z. Song, and M. Song, "Delay sensitive large-scale parked vehicular computing via software defined blockchain," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, May 2020, pp. 1–6.
[18] X. Hou, Y. Li, M. Chen, D. Wu, D. Jin, and S. Chen, "Vehicular fog computing: A viewpoint of vehicles as the infrastructures," *IEEE Trans. Veh. Technol.*, vol. 65, no. 6, pp. 3860–3873, Jun. 2016.
[19] X. Wang, Z. Ning, and L. Wang, "Offloading in Internet of Vehicles: A fog-enabled real-time traffic management system," *IEEE Trans. Ind. Informat.*, vol. 14, no. 10, pp. 4568–4578, Oct. 2018.
[20] X. Huang, P. Li, and R. Yu, "Social welfare maximization in container-based task scheduling for parked vehicle edge computing," *IEEE Commun. Lett.*, vol. 23, no. 8, pp. 1347–1351, Aug. 2019.
[21] K. Nguyen, S. Drew, C. Huang, and J. Zhou, "EdgePV: Collaborative edge computing framework for task offloading," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2021, pp. 1–6.
[22] K. Deb, *Multi-Objective Optimisation Using Evolutionary Algorithms: An Introduction*. London, U.K.: Springer, 2011, pp. 3–34, doi: 10.1007/978-0-85729-652-8_1.
[23] B. Gu, X. Zhang, Z. Lin, and M. Alazab, "Deep multiagent reinforcement-learning-based resource allocation for internet of controllable things," *IEEE Internet Things J.*, vol. 8, no. 5, pp. 3066–3074, Mar. 2021.
[24] B. Gu, X. Yang, Z. Lin, W. Hu, M. Alazab, and R. Kharel, "Multiagent actor-critic network-based incentive mechanism for mobile crowdsensing in industrial systems," *IEEE Trans. Ind. Informat.*, vol. 17, no. 9, pp. 6182–6191, Sep. 2021.
[25] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, "Evolution strategies as a scalable alternative to reinforcement learning," 2017, *arXiv:1703.03864*.
[26] F. Petroski Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune, "Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning," 2017, *arXiv:1712.06567*.
[27] H. Mühlenbein, "Parallel genetic algorithms in combinational optimization," in *Computer Science and Operations Research*, O. Balci, R. Sharda, and S. A. Zenios, Eds. Amsterdam, The Netherlands: Pergamon, 1992, pp. 441–453. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780080408064500344
[28] M. Mitchell, *An Introduction to Genetic Algorithms*. Cambridge, MA, USA: MIT Press, 1998.
[29] Q. Lu, K. Nguyen, and C. Huang, "Distributed parallel algorithms for online virtual network embedding applications," *Int. J. Commun. Syst.*, p. e4325, Jan. 2020. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/dac.4325
[30] H. Zhu and C. Huang, "VNF-B&B: Enabling edge-based NFV with CPE resource sharing," in *Proc. IEEE 28th Annu. Int. Symp. Pers., Indoor, Mobile Radio Commun. (PIMRC)*, Oct. 2017, pp. 1–5.

KHOA NGUYEN received the M.Sc. degree in telecommunications engineering from the University of Sunderland, U.K., in 2013, and the Ph.D. degree in electrical and computer engineering from the Department of Systems and Computer Engineering, Carleton University, Canada, in 2021. His main research interests include communication networks, cloud/edge computing, parked vehicle edge computing (PVEC), the Internet of Vehicles (IoV), software-defined networks (SDN), network function virtualization (NFV), containerization technologies, and machine learning.

STEVE DREW (Member, IEEE) received the B.Sc. degree from Beijing Jiaotong University, in 2008, the M.Sc. degree from the Chinese Academy of Sciences, in 2011, and the Ph.D. degree in electrical and computer engineering from Carleton University, in 2018. He was the Chief Architecture and Security Officer at BitOcean Global and the Founder of BitQubic. He was a Senior Cloud Engineer at Cisco NFV Group. He is currently an Assistant Professor at the Department of Electrical and Software Engineering, University of Calgary. His research interests include edge computing, cloud-native initiatives towards network services, and blockchain services.

CHANGCHENG HUANG (Senior Member, IEEE) received the B.Eng. and M.Eng. degrees in electronic engineering from Tsinghua University, Beijing, China, in 1985 and 1988, respectively, and the Ph.D. degree in electrical engineering from Carleton University, Ottawa, ON, Canada, in 1997. From 1996 to 1998, he worked with Nortel Networks, Ottawa, where he was a Systems Engineering Specialist. He was a Systems Engineer and a Network Architect with the Optical Networking Group, Tellabs, Naperville, IL, USA, from 1998 to 2000. Since July 2000, he has been with the Department of Systems and Computer Engineering, Carleton University, where he is currently a Full Professor. He won the CFI New Opportunity Award for building an Optical Network Laboratory in 2001. He is an Associate Editor of *Photonic Network Communications* (Springer).

JIAYU ZHOU (Member, IEEE) received the Ph.D. degree in computer science from Arizona State University, in 2014. He is currently an Associate Professor at the Department of Computer Science and Engineering, Michigan State University. His research has been funded by the National Science Foundation, the National Institutes of Health, and the Office of Naval Research, and published more than 100 peer-reviewed journals and conference papers in data mining and machine learning. His research interests include large-scale machine learning, data mining, and biomedical informatics, with a focus on the transfer and multi-task learning. He was a recipient of the National Science Foundation CAREER Award (2018). His papers received the Best Student Paper Award in 2014 IEEE International Conference on Data Mining (ICDM), the Best Student Paper Award at the 2016 International Symposium on Biomedical Imaging (ISBI), and the Best Paper Award at the 2016 IEEE International Conference on Big Data (BigData).