EdgePV: Collaborative Edge Computing Framework for Task Offloading

Khoa Nguyen
Carleton University
Ottawa, ON, Canada
khoatnguyen@sce.carleton.ca

Steve Drew
BitQubic Corp.
Kanata, ON, Canada
steve@bitqubic.com

Changcheng Huang
Carleton University
Ottawa, ON, Canada
huang@sce.carleton.ca

Jiayu Zhou
Michigan State University
East Lansing, MI, USA
jiayuz@msu.edu

***Abstract*—Recent research has pointed out that almost all vehicles spend over 95% of their times in parking lots where their powerful computing resources are wasted. In this paper, we propose a novel collaborative computing paradigm that efficiently offloads online heterogeneous computation tasks to parked vehicles during peak hours. A container orchestration based on Kubernetes is integrated into the infrastructure due to its cutting-edge features such as auto-healing, load-balancing, and security. We formulate the offloading problem analytically and present an intelligent metaheuristic algorithm to address dynamic online demands. Extensive evaluation demonstrates that our proposed paradigm improves task acceptance ratio and average offloading cost for more than 40% with high task arrival rate compared with a set of existing algorithms.**

***Index Terms*—Parked Vehicles, Edge Computing, Container Orchestration, Kubernetes.**

I. Introduction

The number of vehicles has been dramatically increasing in the last decade, that are predicted to attain two billions by 2035 [1]. Many of which are generally equipped with powerful on-board computing hardware (e.g. CPU, GPU) to provide modern advanced features such as auto-pilot, intelligent radar, sensing safety systems. These on-board facility supporting level-four autonomous driving might cost thousands of dollars but the resource utilization of these vehicles is extremely low during parking time. Recent studies have indicated that there are 70% of individual vehicles spending almost 95% of their time for parking in parking lots, street parking and home garages [1] [2].

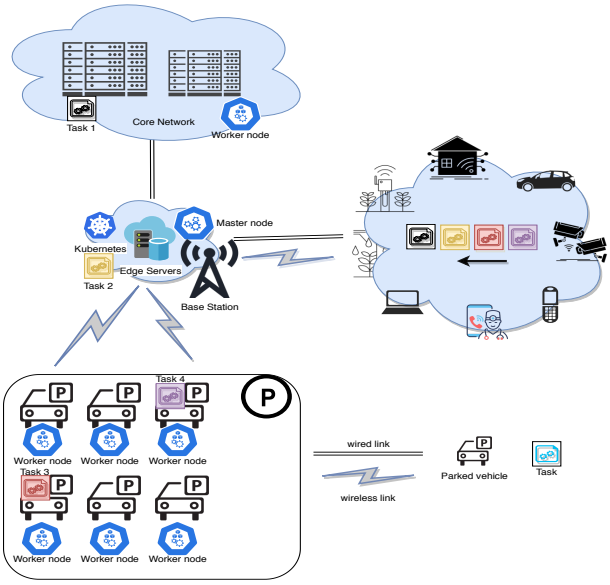


Fig. 1: Edge computing architecture integrated with parked vehicles (PVs) enabled by Kubernetes.

In America, for instance, the average daily driving time was merely 50.6 minutes according to the AAA Foundation survey for Traffic Safety in 2016 [3]. These statistics reveal that the powerful on-board hardware of vehicles is idle for most of the time, giving a great chance to exploit these omitted computation resources for additional services. The neglected computational resources of PVs can be an excellent candidate for Mobile Edge Computing (MEC) where the conventional computation and storage services usually offered by remote cloud are now migrated to the network edge. With the advent of PVs, MEC capacity can be enlarged. However, the collaborative framework between cloud-edge and PVs complicates the task offloading problems by determining appropriate network resources to handle given tasks. Moreover, online heterogeneous tasks can be classified into delay-sensitive (e.g. mobile gaming, autopilot, video surveillance) and delay-insensitive with stringent computing requirements (e.g. health monitoring, vehicular sensing, location-based augmented reality games) [4]. The explosive growth of data traffic with arbitrary requirements, either computation intensive or sensitive-delay tasks, will impose a heavy burden on the existing infrastructure during peak hours. Incoming tasks that cannot

be processed at the cloud due to delay-sensitive requirements can be offloaded to the edge network. Due to the limited capacity at both remote cloud and edge during peak hours, networks can rapidly become congested when a number of tasks increase. On the other hand, parking time of PVs is uncertainty, that makes PV nodes unreliable to run applications/services on them. Thus, a new network design is desired to solve these aforementioned problems.

Applying a generic container orchestration to edge computing enhanced by PVs is still infancy. Container orchestration (e.g. Kubernetes) enables PVs to efficiently run several replicas of a task simultaneously since it allows fast bootup, auto-scaling, self-healing and rapid termination time. These agile features are critical to solve the uncertainty problem of limited parking duration of PVs. Moreover, containerization requires lower hardware requirements, and offers less operation costs and resource isolation when each container is independently running replicas of a given task. However, where the replicas of a task are offloaded to a collaborative infrastructure meeting rigid resources and reliability constraints while minimizing network costs is not an easy task. For example, if all task replicas are allocated to a single node (e.g. PVs), but this node gets an unexpected failure (e.g. outage, sudden vehicular leaving). Services provided through running such containerized task will be interrupted. To guarantee the reliability, a least proportion of replicas should be running on different worker nodes and in this paper we set this proportion less than 50%. This proportion can be determined and adjusted easily by SPs depending on their network service strategies.

In this paper, we propose EdgePV, a novel collaborative architecture in which PVs expands the existing resource capacity of Cloud-Edge infrastructure to handle online containerized tasks during peak hours at edge. An incoming task can be abstracted in a form of multiple replicas running on independent containers in a containerization framework. The online task offloading problem of the proposed collaborative framework is formulated as Binary Integer Programming (BIP) with respect to minimizing offloading costs while maximizing accumulative rewards. Efficiently scheduling the tasks, specifically the number of task replicas throughout the collaborative paradigm with respect to stringent constraints, has been still unsolved. Hence, we propose Genetic Algorithm (GA), a mature metaheuristic algorithm, to solve the task offloading problem meeting rigid task requirements (e.g. delay-sensitive) with low costs and high reliability. Furthermore, owners of PVs who are selling their idle resources can obtain accumulating reward points (user utility) that are able to be converted to parking tickets, gift-card, shopping vouchers, gas, and so on.

As illustrated in Fig. 1, we suggest edge server implements Kubernetes as a master node whereas remote cloud, available computing resources of edge server and PVs can run as worker nodes. PVs are installed a light-weight Kubernetes version (e.g. K3S), permitted as preamble nodes in the network due to the uncertainty of their parking time. The master node manage the state of the cluster, schedule the containers, accept or reject the task requests. A scheduler in the master node conducts pod placement (running container) across the set of available nodes. This proposed architecture improves the elasticity and agility of the existing infrastructure to cope with any possible service disruption caused by the mobility of PVs, which has not been solved before. It is also envisaged that PVs could be completely electric-based in near future, and they would be automatically charging while parking. Moreover, D2D technology can be possibly integrated into this collaborative infrastructure, but it is beyond the scope of this research since its main goal is to tackle the online task offloading problem. We divide the contents of this paper into the following sections. The related work will be presented in Section II. Section III will formulate the problem. Section IV describes the proposed GA algorithm based on the problem formulation. Then the simulation results will be shown in Section VI. Section VII will conclude the paper.

II. Related Work

PVs as infrastructure have recently received significant attentions since they expand the existing computing infrastructure for computation, communication and storage (CCS). Enabling PVs for vehicular cloud computing in Internet of Vehicle has studied in [5]–[11].

Arif et al. in [5] studied the simple model of a vehicular cloud (VC) formed by PVs in an international airport. [6] presented a multi-layered vehicular cloud architecture based on cloud computing and Internet of Thing (IoT) technology. Similarly, the recognition of a VC erected on PVs in a parking lot as a spatial-temporal network architecture for CCS was investigated in [7]–[9], [12]. Additionally, [10] concerned the feasibility of PVs as a computation paradigm, and introduced an incentive algorithm offering accumulating rewards for PVs by selling their resources. Moreover, Hou et al. [13] approached the vehicular fog computing (VFC) that exploited connected PVs as the infrastructure to handle realtime services at the edge. Similarly, [14] presented a fog computing architecture deployed in Internet of Vehicle (IoV) systems to provide computational resources to end users with latency guarantee. Recently, Parked Vehicle Edge Computing (PVEC) where PVs as accessible edge computing nodes to deal with task allocation has been proposed in [1], [3], [15]. The authors in [1] explored possible opportunistic resources to handle computational tasks in a combined infrastructure between vehicle edge computing (VEC) servers and PVs. [3] introduced a dynamic-pricing strategy in aim at minimizing average costs and meeting the QoS constraints. Additionally, a containerized task scheduling scheme enabled by PVEC was proposed in [15] concerning the social welfare optimization for both users and PVs. [11] proposed a scalable vehicle-assisted MEC infrastructure that integrated the remote cloud, MEC and mobile volunteer vehicles (buses) to process task requests from IoT devices. It may look similar to the idea of this paper but our paper is aimed at solving the online task offloading problems in container-based computing framework (EdgePV) concerning the allocation of several task replicas. Our proposed algorithm not only take the network costs, but also the accumulating rewards achieved by selling computational resources of PVs into account.

EdgePv involves PVs in parking lot which are more popular and reliable than buses due to their less mobility.

III. Problem Formulation

In this section, we will formulate our model that considers resource constraints of the network edge as the orchestration scheduler placed in edge server aside in 5G base station (BS).

A. System Model

In this paper, we concern are CPU, memory and bandwidth resources. Network edge consists of various types of worker nodes (cloud, edge and PVs) that are connected to the master node located in edge server via different links. For example, cloud node connects to the master node via optical link while PVs integrate into the edge via wireless links in which available bandwidth of a vehicle is dependent on the distance to BS. Thus, it can consider the network edge with a star topology in which root and leaves are master node and worker nodes respectively. As illustrated in Fig. 1, a typical outdoor parking lot is investigated where PVs are initially required to register their information (e.g. owner's ID, license plate, preferable parking availability) and vehicle resources (e.g. computational capacity, storage) with a SP. When PVs arrive a parking lot or complete tasks, they can send/update their state information to the SP or master node. The edge network is modeled as a directed graph $G = (N, L)$ where N is the set of worker nodes whereas L is the set of corresponding links. The edge server connects to the remote cloud via an optical link and PVs connects to the network edge via wireless links, denoted as l_c and l_v respectively. Each worker node can initialize several pods to run containers processing task replicas simultaneously with QoS guarantee. A given task k has CPU $c(k)$, Memory $m(k)$, Bandwidth $b(k)$, tolerable maximum latency $t_m(k)$ and the set of replicas $\eth(k)$ requirements. k_j denotes a j^{th} replica of the task $k \in K$, then $\sum k_j = \eth(k)$. A worker node $n_i \in N$ has its own resource capacity to operate a limited number of containers. Denote $C(n_i)$ and $M(n_i)$ as CPU and memory capacity that i^{th} worker node can provide respectively. Let denote K_c, K_e, and K_p as the set of tasks offloaded to the cloud, edge and PVs respectively. The residual CPU and Memory capacity of a worker node can be computed as below:

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

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

where u denotes worker node type (Cloud: c, Edge: e, PVs: p)

B. Channel Model

1) Core network offloading latency: Total network latency comprises data transmission time and task execution time. The formal is highly correlated to the remaining bandwidth of the link l_c, while the later depends on how busy the cloud is to handle the offloaded tasks or to operate other services. Large number of tasks offloaded to the cloud via l_c and less residual resources increase network latency, especially in peak hours. Thanks to accelerating technologies in data center, the delay caused by writing or accessing the data volumes from memory can be neglected. The cloud offloading latency $t_c(l_c)$ including the transmission delay and the processing delay can be computed as below:

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

where ξ_c and f_k denote the transmission rate of server and the number of CPU cycles utilized to calculate per bit respectively. Thus, the total number of CPUs required to calculate the task k can be expressed as $\chi_k f_k$. d, v and T_h are the distance between the core cloud and edge cloud, the speed of light and the constant time of handling an incoming task respectively. Edge devices transmit tasks to the edge servers via wired or wireless links (base station) for processing. For simplification, the delay caused by handling a task can be described as $T_h = \frac{\chi_k}{B_e \nu}$. where ν is a discount factor that reflects fluctuations of bandwidth at the edge ($0 < \nu < 1$). Different from the remote cloud, when the task is managed at the edge, the delay can be only caused by the remaining computational capacity to process the task. The edge offloading latency $t_e(k)$ can be computed as below:

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

2) PVs latency: Unlike the cloud/edge nodes that can be considered as stable, PVs shall be considered as preemptible nodes due to their uncertain mobility. Increasing the number of replicas can be a possible approach to avoid service disruption. By that solution, the master node might have more time to migrate the current task to other nodes for QoS guarantee.

Similar to [11], we leverage LTE-A for wireless communication between the base station and PVs, and consider the system applying orthogonal frequency-division multiple access (OFDMA) scheme. Denote the parameter $d_{bs,p}$ is the distance between the base station and p^{th} PV. The path loss of the base station and parked vehicle can be characterized as $d_{bs,p}^{-\sigma}$ and the white Gaussian noise power N_0, where σ factor is the path loss exponent. The corresponding wireless channel is modeled as frequency-flat block-fading Rayleigh fading channel that is denoted as h. Accordingly, the data rate capacity of the p^{th} PV can be expressed as:

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

where the parameter B_p denotes the channel bandwidth. P_{TX} and I represents the transmission power of base station and inter-cell interference respectively. The PVs offloading latency $t_p(k)$ can be computed as below:

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

In this paper, we define two types of online tasks including delay-sensitive and delay-insensitive akin to [4]. The former is merely handled at the edge node or PVs due to their closest proximity, while the later can be placed on any network nodes (Cloud, Edge and PVs). Next, we compute the costs of mapping the task replicas into the worker nodes through the collaborative computation platforms. The mapping costs involve sum of total number of CPUs required to compute

task replicas, memory, bandwidth and energy consumption (e.g. battery) for operating replicas at PVs since cloud and edge computing platforms possess very high energy efficiency. Offloading cost at the Cloud can be expressed as below:

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

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

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

where δ is small positive number to prevent dividing by zero. Total cost of offloading a task to the cloud:

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

When the given task is processed at the edge, it can be considered as local processing, so the offloading cost at the Edge is computed as below:

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

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

Total cost of offloading a task to the edge:

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

Similarly, offloading cost of a task to PVs can be computed with additional energy consumption attribute as below:

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

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

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

$$E_p(k) = \sum_{j \in \eth(k)} \chi_{k_j} f_{k_j} e_p \tag{17}$$

where e_p is a coefficient, that can be achieved by:

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

where ϵ denotes the energy coefficient.
Total cost of offloading a task k to a PV:

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

where ς is energy cost coefficient.

3) PVs' Utility: To encourage PVs to sell their idle resources while parking in the parking lots, owners of PVs should receive the rewards when they accept to process tasks on their vehicles. Let φ^p represent the revenues by accepting the tasks and the utility of a PV can be calculated as below:

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

where ρ denotes a coefficient of energy price, and φ^p can be expressed:

$$\varphi^p = \sigma \sum_{j \in \eth k} r_p^c \chi_{k_j} f_{k_j} + r_p^m m(k_j) \tag{21}$$

where r_p^c and r_p^m are the unit prices for offering CPU and memory resources respectively. It is recognized that minimizing the cost embedding tasks would increase the economical benefits gained by accepting to process the requested tasks in PVs. Variables:

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

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

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

Objective:

$$Minimize \sum_{k \in K} \sum_{j \in \eth k} \Xi_k^c \mathcal{A}_{k_j}^c + \Xi_k^e \mathcal{A}_{k_j}^e + (\eta \Xi_k^p + (1-\eta) \frac{1}{\varphi_k^p}) \mathcal{A}_{k_j}^p \tag{22}$$

$$w.r.t \qquad \mathcal{A}_{k_j}^c, \mathcal{A}_{k_j}^e, \mathcal{A}_{k_j}^p$$

Constraints:

$$\mathcal{A}_{k_j}^c + \mathcal{A}_{k_j}^e + \sum_{p \in N} \mathcal{A}_{k_j}^p = 1, j \in \eth(k), \forall k \in K \tag{23}$$

$$1 \leq \sum_{j \in \eth(k)} \mathcal{A}_{k_j}^p \leq \frac{\eth(k)}{2}, \forall k \in K \tag{24}$$

$$\sum k_j = \eth(k), \forall k \in K \tag{25}$$

$$R_C^c(n_i), R_C^e(n_i), R_C^p(n_i) \geq c(k), n_i \in N \tag{26}$$

$$R_M^c(n_i), R_C^e(n_i), R_M^p(n_i) \geq m(k), n_i \in N \tag{27}$$

$$R_B^c(n_i), R_B^e(n_i), \xi_{n_i} \geq b(k), n_i \in N \tag{28}$$

$$t_c, t_e, t_p \leq t_m(k) \tag{29}$$

Remarks:

- Function (22) includes dual objectives: minimizing the cost of offloading computation tasks and maximizing the PV rewards when the tasks are offloaded to PVs where η is a damping factor within (0,1).
- Constraint (23) ensures that each task replica is only deployed at one worker node.
- Constraint (24) guarantees that no more than 50% the number of task replicas can be placed at the same PV node.
- Constraint (31) makes sure that total number of replicas scheduled are at least equal to the required replicas of the corresponding task.
- Constraints (26),(27), and (28) assure that the residual resources of the worker nodes (e.g. Cloud, Edge, PVs) satisfies the capacity requirements of the task.

- Constraints (29) ensures the selected nodes satisfy the latency constraints.

IV. Our Proposed Genetic Algorithm

A. Descriptions of Genetic Algorithm

GA algorithm is a mature metaheuristic that is motivated by the Darwin evolution principle through natural selection, including four major operations: initialization, selection, crossover and mutation [1]. To solve BIP problem, we present a distributed parallel GA-based algorithm that operates on a predefined number of independently machines, denoted as p, to explore feasible solutions widely known as chromosomes. The design of our proposed parallel algorithm is depicted in Fig 2 in which p is set to 16. As illustrated, the offloading procedures are successively working under a master node (e.g. synchronization). Several working nodes run a GA algorithm to discover as many feasible solutions as possible for replica scheduling. The best solutions from the worker nodes are synchronized to select the final solution for task offloading. Our proposed algorithm in this study assumes to schedule multiple task replicas at once instead of sequentially mapping.

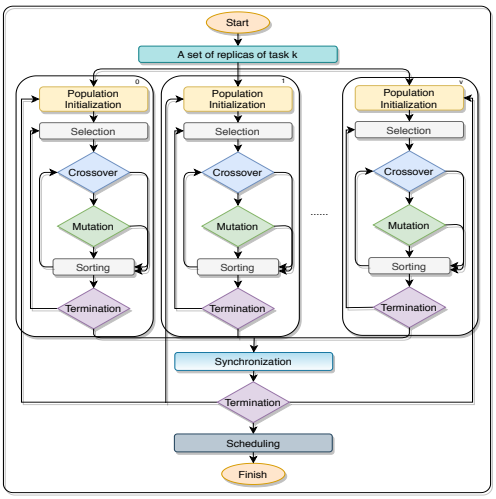


Fig. 2: *Parallel operation scheme*

Chromosome: A chromosome $\mathcal{C}_f$ represents a scheduling solution of a set of replicas of a given task request that are selected from the available nodes in random. Each gene in a chromosome involves a mapping solution for a single task replica. If there are G genes and M chromosomes, the initial population $\mathcal{P}$ (MxG size) at the k^{th} working machine can be delineated as below:

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

[1]Due to page limitation, more details about GA algorithm can be found in [16]

When a chromosome is established by N potential genes that have qualified the feasibility process, such is defined as a feasible solution for a task request.

Fitness Function: Our objective is to minimize the scheduling costs and maximize the network utility when replicas of a given task are offloaded to PVs. Fitness values are utilized to evaluate the quality of a scheduling solution, so higher fitness value represents a good solution.

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

New generations: In this paper, we randomly select chromosomes to become parents to generate new population. New chromosomes are intentionally generated to produce new generation as a result of crossover and mutation operations. Aimed at improving the diversity, the population updates these new generated generations so it is consequently evolved, increasing the possibility to achieve near-optimal scheduling solution.

B. Terminations and Synchronization

A parallel operation is typically consisted of concurrent processes, and each accomplishes its job at different time. Waiting until all tasks finish their assignments is frustrated as one or more tasks might take longer time to be done (e.g. deadlock). Thus, if there is no better solution to be found in t times; where t is denoted as a termination parameter, the master procedure terminates worker nodes to reduce total execution time. Moreover, the feasible solutions received from the worker nodes are synchronized to determine the best scheduling solution for the corresponding task request, based on fitness function values. If accepted, the given task replicas are consequently placed onto the corresponding nodes in the network following the scheduling solution found. Substrate network then updates its residual computational resources to end the scheduling procedures.

V. Compared Algorithms

We propose a metaheuristic algorithm that minimizes the embedding cost while improves the PVs utility efficiently by selecting proper worker nodes to run the task replicas. All proposed algorithms allows to schedule multiple requested task replicas into the same worker node, but the proportion of the number of replicas placed in this node cannot exceed 50% (except Cloud/Edge nodes) in order to ensure the service reliability. This setting can be flexibly changed by SP. We investigate the scheduling efficiency of our proposed GA-based algorithm by comparing with some heuristic algorithms including: Baseline_1, Baseline_2, and Baseline_3. Baseline_1 prefers Kubernetes default scheduler with filtering and scoring procedures while Baseline_2 schedules online tasks by selecting worker nodes for task offloading in random. Furthermore, Baseline_3 deploys branch and bound selection policy to handle incoming tasks [17]. There are basically three performance metrics for performance evaluation including task acceptance ratios (A/R), costs and accumulated utility.

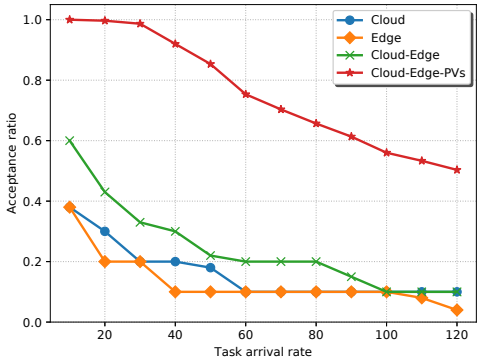


Fig. 3: Acceptance ratio

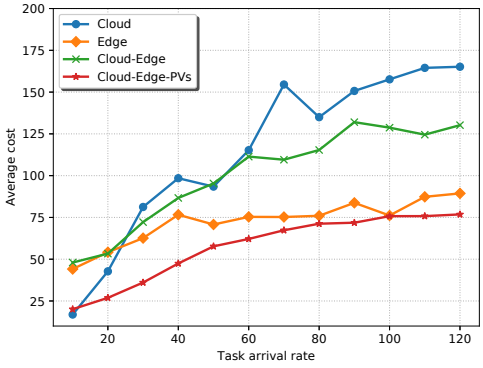


Fig. 4: Average costs between architectures

VI. Numerical Results

A. Simulation setup

We have developed a discrete event simulator to evaluate the proposed algorithms. PVs dynamically arrive and depart from a parking lot with 50 free parking spots. In fact, the parking lot can fully be occupied, but we assume that the parking capacity remains at least 50% up to 85% in peak hours since not all of PVs are willing to sell their resources or are qualified to join into the network to provide the services. Furthermore, it is observed that parking duration of PVs is ranging from 08 to 240 minutes [1] or 30 to 120 minutes [12]. More than 85% of PVs spend maximum three hours in a parking lot and serviceability probability of PVs achieves around 90% at 60 minutes [1]. In this paper, the accumulative parking duration of PVs is following Poisson distribution with $\lambda = 3600$. As discussed in previous sections, the online requested tasks can be classified into delay-sensitive and delay-insensitive tasks. If the delay tolerance of a task exceeds 20 ms, we considered it as a delay-sensitive demand. Our simulation run approximately for 8 hours (peak business working time) and the simulator will update the PVs every 20 minutes. Additionally, 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} [12], respectively. Details of simulation parameters can be found in Table I.

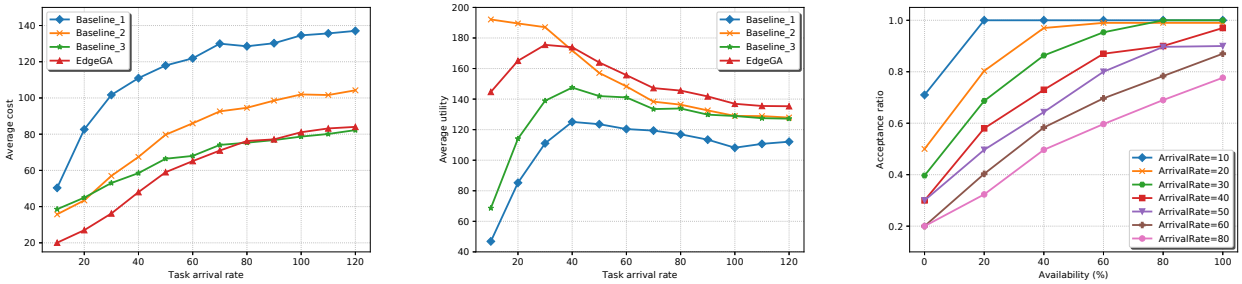
B. Performance Results

As illustrated in Fig. 3, Cloud-Edge-PVs paradigm extends the resource availability of the network that increases the possibility of accepting the incoming requests more than

TABLE I: Simulation Parameter Settings

Parameter	Values
Maximum parking capacity	50
Total simulation time	30,000 seconds
Vehicle lifetime	[480-14400] seconds
C_c / M_c / B_c	50GHz/1000MB/1Gbps
C_e / M_e	20GHz / 500MB
$W_{\{C_c,M_c,B_c\}}$	750
$W_{\{C_e,M_e\}}$	250
$W_{\{C_p,M_p,B_p\}}$	10
Channel Bandwidth B_p	10 MHz
P_{TX} / N_0 / σ	1.3W / 3×10^{-13} / 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 A/R	[10-120]
Request replications $\eth(k)$	[2 - 10]
r_p^c / r_p^m	10 / 100

40% compared to Cloud-Edge and Cloud architectures at the task arrival rate of 120, respectively. Cloud or edge itself gets lowest acceptance ratio due to their limited resource capacity during peak hours. By preferring PVs for task offloading, Cloud-Edge-PVs achieved the lowest average cost values compared to all compared platforms as illustrated in Fig. 4. In performance evaluation between proposed algorithms as illustrated in Fig. 5, Baseline_1 performed worst in terms of average costs due to its allocating strategies through filtering and scoring procedures while Baseline_2 based on a random strategy to select the worker nodes performed better than Baseline_1. Amongst baseline algorithms, Baseline_3 was originally designed to target reducing the offloading cost so its performance was comparative to our proposed algorithm, but EdgeGA was still better than Baseline_3 until arrival rate of 80 and seemed to be lightly the same afterward as shown in Fig. 5a. Online heterogeneous tasks were mostlikely to be assigned to PVs which expectedly produce lower offloading costs. In utility metric, EdgeGA outperformed all compared algorithms following Baseline_2,Baseline_3 and Baseline_1 respectively as depicted in Fig. 5b. The reason is that EdgeGA simultaneously took both cost and utility into account driven by an efficient fitness function (31). Besides, we evaluated the availability of PVs regarding the acceptance ratios on several arrival rates as depicted in Figure 5c. It has been demonstrated that depending on the selected arrival rates, each had different preferable PV availability. For example, arrival rates (10, 20, 30, 40, 50) required 60% availability of PVs to exceed 80% acceptance ratios while arrival rates of 60 and 80 needed to reach 80% and 100% to obtain the same result, respectively. These information is vital for the network planners to achieve desired Key Performance Indicators (KPIs) by adopting appropriate strategies. For instance, SP may increase user incentives to appeal more PVs join into the network, offload to another cluster or expand edge server capacity. Furthermore, our proposed GA-based algorithm successfully processed a given task in average $1.217ms$ compared to $14.725ms$



(a) Average offloading cost (b) Average utility (c) A/R towards PV availability

Fig. 5: Performance evaluation between compared algorithms

in sequential GA operation. This superior execution-time performance attained by deploying the parallel scheme for GA algorithm as proposed in Fig. 2. This execution time is very competitive, and it somehow lifts the curse of possibly high computation time when running GA algorithms.

ACKNOWLEDGEMENT

This research was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) Engage grant (EGP 543449-19).

VII. CONCLUSION

In this paper, we have studied the collaborative framework where PVs are potentially considered as an extension for the existing cloud-edge computing infrastructure to handle online container-based task offloading in peak hours. We have devised Kubernetes, a container orchestrator, deployed at edge servers as master nodes, while remote cloud, edge itself and PVs perform as worker nodes. Extensive experiments demonstrated that our proposed collaborative paradigm not only extends the computation resources of existing network infrastructure by taking advantage of high on-board computers of PVs efficiently, but also brings a flexible, agile and reliable framework for task offloading problems. In future work, we consider sophisticated algorithms (e.g. machine learning techniques) for task offloading problem.

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. 66 649–66 663, 2018.

[2] F. H. Rahman, A. Yura Muhammad Iqbal, S. H. S. Newaz, A. Thien Wan, and M. S. Ahsan, "Street parked vehicles based vehicular fog computing: Tcp throughput evaluation and future research direction," in *2019 21st International Conference on Advanced Communication Technology (ICACT)*, 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 Communications Letters*, vol. 8, no. 2, pp. 420–423, 2019.

[4] O. Fadahunsi and M. Maheswaran, "Locality sensitive request distribution for fog and cloud servers," *Service Oriented Computing and Applications*, vol. 13, no. 2, pp. 127–140, Jun 2019. [Online]. Available: https://doi.org/10.1007/s11761-019-00260-2

[5] 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 Transactions on Parallel and Distributed Systems*, vol. 23, no. 11, pp. 2067–2080, 2012.

[6] W. He, G. Yan, and L. D. Xu, "Developing vehicular data cloud services in the iot environment," *IEEE Transactions on Industrial Informatics*, vol. 10, no. 2, pp. 1587–1595, 2014.

[7] F. Dressler, P. Handle, and C. Sommer, "Towards a vehicular cloud - using parked vehicles as a temporary network and storage infrastructure," in *Proceedings of the 2014 ACM International Workshop on Wireless and Mobile Technologies for Smart Cities*, ser. WiMobCity '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 11–18. [Online]. Available: https://doi.org/10.1145/2633661.2633671

[8] E. Al-Rashed, M. Al-Rousan, and N. Al-Ibrahim, "Performance evaluation of wide-spread assignment schemes in a vehicular cloud," *Vehicular Communications*, vol. 9, pp. 144 – 153, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2214209616301863

[9] T. Kim, H. Min, and J. Jung, "Vehicular datacenter modeling for cloud computing: Considering capacity and leave rate of vehicles," *Future Generation Computer Systems*, vol. 88, pp. 363 – 372, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X18300487

[10] 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 of Things Journal*, vol. 6, no. 4, pp. 6079–6088, 2019.

[11] 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," *Journal of Cloud Computing*, vol. 9, no. 1, p. 28, Jun 2020. [Online]. Available: https://doi.org/10.1186/s13677-020-00175-w

[12] 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 *2020 IEEE Wireless Communications and Networking Conference (WCNC)*, May 2020, pp. 1–6.

[13] X. Hou, Y. Li, M. Chen, D. Wu, D. Jin, and S. Chen, "Vehicular fog computing: A viewpoint of vehicles as the infrastructures," *IEEE Transactions on Vehicular Technology*, vol. 65, no. 6, pp. 3860–3873, 2016.

[14] X. Wang, Z. Ning, and L. Wang, "Offloading in internet of vehicles: A fog-enabled real-time traffic management system," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 10, pp. 4568–4578, 2018.

[15] X. Huang, P. Li, and R. Yu, "Social welfare maximization in container-based task scheduling for parked vehicle edge computing," *IEEE Communications Letters*, vol. 23, no. 8, pp. 1347–1351, 2019.

[16] M. Mitchell, *An Introduction to Genetic Algorithms*. Cambridge, MA, USA: MIT Press, 1998.

[17] H. Zhu and C. Huang, "VNF-B&B: Enabling edge-based NFV with CPE resource sharing," in *2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, 2017, pp. 1–5.