Collaborative Container-based Parked Vehicle Edge Computing Framework for Online 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*—As most vehicles spend over 95% of their time in the parking lots, the powerful computing resources of parked vehicles (PVs) are underutilized, that can be considered as available computing nodes to run tasks as well as an extension of the existing infrastructure. In this paper, we propose EdgePV, a collaborative computing paradigm to efficiently improve online heterogeneous task scheduling. To guarantee service reliability, a container orchestration (e.g. Kubernetes) is advocated to be integrated into this proposed architecture due to its notable advanced features such as load-balancing, auto-healing, resource isolation, security, etc,. Kubernetes coordinates PVs to run sufficient numbers of task replicas, providing high service availability against possible failure caused by the mobility of PVs. We investigate how efficient PVs can handle the online computational tasks during peak hours. We also present the dual cost and utility-aware heuristic algorithm, compared with a set of heuristics to solve the problem of task scheduling, that can be devised for replacing the default scheduler in Kubernetes platform. Extensive simulation results show that our proposed design improves the task acceptance ratios and average costs at least 23% and 64%, respectively, at lowest task arrival rate compared to the cooperated cloud-edge architecture. Furthermore, owners of PVs can significantly benefit from incentives received by sharing the idle resources of their PVs.**

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

I. Introduction

Last decade has witnessed an increasing number of vehicles that are anticipated to reach 2 billions by 2035 [1]. Many of which are equipped with state-of-the-art on-board computers and sensors to offer advanced features such as autopilot, self-parking, radar, automatic safety systems. Notably, studies have shown that 70% of individual vehicles almost spend 95% of time for parking in parking lots, home garages or street parking spaces [1] [2]. Those on-board computers with level-four autonomous driving support can cost tens of thousands of dollars but the expected utilization of vehicles is extremely low. For example, the average driving time per day in America was only 50.6 minutes according to the data survey of AAA Foundation for Traffic Safety in 2016 [3]. These statistics show that on-board computers are idle in most of the time, suggesting that significant computation resources can be exploited for provisioning additional services. The overlooked resources of PVs can become ideal candidates for transient resources leveraged by Mobile Edge Computing (MEC). By edge computing, the conventional computation and storage services offered by the remote cloud are now extended to the edge of the network. With the introduction of PVs, the strength of edge is now doubled, but they complicates the offloading problem through determining the proper network resources to execute a given task. Moreover, incoming tasks can be classified into delay-sensitive (e.g. augmented reality, video surveillance, mobile gaming, transportation, live streaming, robot collaboration, autopilot) and delay-insensitive with rigid computational requirements [4].

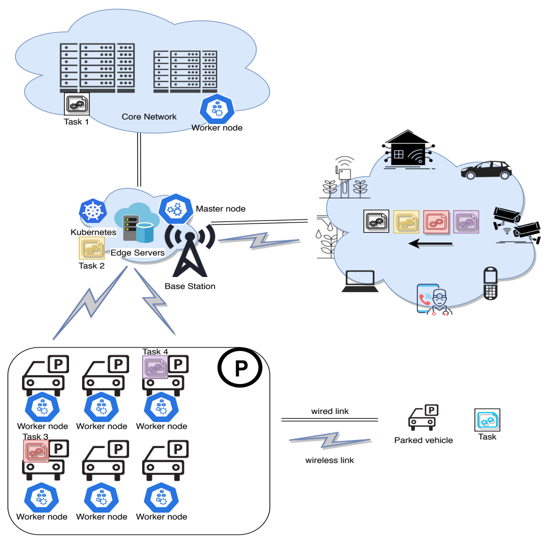


Fig. 1: Edge computing architecture integrated with parked vehicles (PVs) enabled by Kubernetes.

Moreover, an explosive growth of data traffic with heterogeneous computation demands, either sensitive or tolerable to higher latency, will lead to a heavy burden of network workloads during peak hours. Additionally, the main drawback of cloud computing is long propagation delays while edge computing with a close proximity to end users has been widely emerged as a key paradigm for the next generation networks. In fact, the coexisting of remote cloud and edge computing is the most dominant form for task execution, and the offloading tasks are not always delay-sensitive in reality. Thus, incoming tasks that do not match well with the cloud in terms of delay requirements can be offloaded to the network edge. Due to limited computing capacity of edge computing, when more applications demand sensitive-latency tolerance during peak hours, the network becomes congested. Therefore, we need efficient algorithms to embed online heterogeneous tasks onto the collaborative framework between cloud-edge and PVs effectively. Moreover, application vendors have little chance to negotiate the embedding service costs since these services are often provided by Service Providers (SPs) with a fixed amount of service charges. Additionally, owners of those smart vehicles cannot get any benefit from their powerful on-board computing resources while parking.

Little work has been done to apply a generic container orchestration system to edge computing enhanced by PVs. Container orchestration allows PVs to efficiently run multiple task replicas simultaneously with fast boot-up and short termination time. Such agility is important due to PVs' limited and arbitrary parking duration, enabling them to reliably execute tasks. Containers may also require lower hardware requirements, operation costs and support resource isolation since each container processes given tasks independently. The de facto industrial standard container orchestration framework (e.g. Kubernetes [5]) offers advanced features such as auto-scaling, self-healing and security policies.

In this paper, we propose EdgePV, a novel collaborative architecture that enables a flexible task execution during peak hours, where PVs become computing nodes to expand the existing resource capacity at the edge. Owners of PVs may sell their resources cheaper than Infrastructure Providers (IPs) for utility or rewards. These can be converted to accumulating rewarded points for shopping, gas, parking tickets and so on. As illustrated in Fig. 1, our proposed architecture investigates the possibility of an integration between the legacy cloud computing, an emerged edge paradigm and PVs. We do not exclude a possible integration of D2D technology into this infrastructure, but this is beyond the scope of this paper because our goal is mainly to address the offloading problem when online tasks arrive at the network. Moreover, we advocate the possibility of integrating Kubernetes-based platform into this model that is aimed at improving QoS and security concerns. In this model, master node is placed at edge node, managing the worker nodes which include the computation resources from the cloud, edge server itself and mobile PVs to handle online task offloading. Both are adopted Kubernetes system forming a container orchestration cluster to provide several advanced features such as load-balancing, auto-scaling, auto-healing, resource management, security, etc,. Kubernetes will run the task workload by allocating containers into pods to run on worker nodes. When a PV node fails, Kubernetes automatically reschedules the requested tasks onto another healthy node. The control plane in the master node manages the worker nodes and the pods in the corresponding cluster, and determine global decisions about the cluster (e.g. scheduling, scaling). When online tasks dynamically arrive at the master node with stringent constraints (e.g. CPU, BW, delay tolerance), kube-scheduler in the control plane of the master node creates pods and assigns the node for them to run on. More details about Kubernetes can be found in [5].

We formulate online task offloading problem of the proposed collaborative framework as Binary Integer Programming (BIP) aimed at *m*inimizing overall offloading costs while *m*aximizing accumulative utility. A heuristic algorithm, namely *M&M*, that is considered for replacing the default scheduler in Kubernetes is proposed to efficiently handle task offloading problem at the edge. Our proposed paradigm advocates Kubernetes platform to tackle the uncertainty of PVs' mobility and network interruptions with advanced automation techniques like autohealing and autoscaling. We simulate the model with the random mobility behaviors of PVs with random and dynamic task arrivals, and compare a set of scheduling algorithms in performance simulation. To the best of our knowledge, this is the first paper that solves the task offloading problems across collaborative infrastructure domains (Cloud, Edge and PVs), considering several network constraints (e.g. CPU, Memory, utility, energy consumption, task replicas). We divide contents into following sections. The related work is illustrated in Section II. Section III formulates the problem. Section IV proposes the M&M heuristic algorithm based on the problem formulation. Then the simulation results are shown in Section V. Section VI concludes the paper.

II. Related Work

PVs as infrastructure that extend the computing environment for computation, communication and storage have received significant attentions in recent years. Enabling PVs to vehicular cloud computing in Internet of Vehicle system are studied in [6]–[11].

Arif et al. in [6] considered the simplest deployment of a vehicle cloud (VC) assisted by PVs in a long-term parking lot of an international airport. This research work predicted the parking occupancy to schedule network resources and then assigned the computation tasks. In addition, [7] introduced a multilayered vehicular data cloud architecture relied on cloud computing and Internet of Thing (IoT) technologies. Two vehicular data cloud services including an intelligent parking and a vehicular data mining cloud services in IoT environment were also proposed. Similarly, the realizations of a VC built on PVs in a parking lot as a spatial-temporal network infrastructure for communication, computing and storage were explored in [8]–[10]. Additionally, another investigation that studied the feasibility of PVs as computation infrastructure was conducted in [11], proposing an incentive mechanism to offer rewards for PVs on account of the parking duration as well as their energy consumption.

In addition, Hou et al. [12] proposed vehicular fog computing (VFC) utilizing connected PVs as the infrastructure in order to provide real-time services at the edge with low latency. Authors also discussed four scenarios of exploiting PVs and slow moving vehicles as computation and communication paradigm. In similarity, fog computing implemented in Internet of Vehicle (IoV) systems to provision computation resources to end users with latency guarantee was presented in [13]. This proposed paradigm enabled to offload the real-time network traffic in fog-based IoV systems in aim of minimizing the average response time.

Recently, PVs proposed to become accessible edge computing nodes to conduct task execution, called by Parked Vehicle Edge Computing (PVEC) were introduced in [1], [3], [14], [15]. The authors in [1] cooperated vehicle edge computing (VEC) servers and PVs to explore possible opportunistic resources in order to deal with computational tasks. Their study investigated the problem of user payment minimization simplifying budget or latency constraints, which would make the proposed scheme fall into sub-optimal. Similarly, adapting the current system state to calibrate the price constantly, [3] presented a dynamic pricing strategy to minimize the average cost while meeting the user QoS constraints. The authors in [14] proposed a container-based task scheduling scheme for PVEC in term of social welfare aspect concerning both users and PVs. To date, [15] recommended a delay-sensitive parked vehicular computing system that utilized softwarized block-chain technology for security and privacy. They only took PVs into account as an edge computing paradigm to handle task offloading and excluded the core cloud and edge server computing themselves.

III. Problem Formulation

In this section, we formulate our proposed model regarding resource constraints of the network edge as the orchestration scheduler is placed in edge server aside in 5G base station.

A. System Model

We consider a network edge which includes different types of worker nodes (cloud, edge and PVs) that are connected to the master node located in edge server via different types of links. For example, cloud nodes connect to the master node via fiber optic while PVs join into the network via wireless links. Hence, the network edge can be considered as a star topology, where root is master node and leaves are worker nodes. In this model, we merely focus on downloading input data because as many online tasks (e.g. face recognition), the data size of output is much smaller than the input data. We do not consider the receiving time of the processed task results from all worker nodes either. The network edge can be modeled as a directed graph $G = (N, L)$ where N is the set of worker nodes in the network and L is the set of corresponding links. These links are also a guarantee of QoS in term of bandwidth provision. The network edge is connected to the core network via a fiber optical link, denoted by l_c. Each node can build several containers to process tasks simultaneously with efficiency guarantee. In this model, a single node can execute multiple pods that are running containers to process required tasks. Each task k has the requirements of data size χ_k (in bits), memory $m(k)$, tolerable maximum latency $t_m(k)$ and the number of replicas $\eth(k)$. Each worker node $n_i \in N$ has its own resource capacity to execute a limited number of containers. Denote $C(n_i)$ and $M(n_i)$ are CPU and memory capacity that i^{th} worker node can provide respectively. Let denote K_c, K_e, and K_p as a set of tasks offloading to the cloud, edge and PVs respectively. The residual CPU, memory and bandwidth capacity of a worker node can be calculated as below:

$$R^u_C(n_i) = C(n_i) - \sum_{k \in K} c(k), \forall k \in K, \forall n_i \in N \tag{1}$$

$$R^u_M(n_i) = M(n_i) - \sum_{k \in K} m(k), \forall k \in K, \forall n_i \in N \tag{2}$$

where u denotes the type of the worker nodes (Cloud: c, Edge Cloud: e, PV: p).

B. Delay Model

Due to accelerating technologies in data centers (e.g. caches, solid-state drives), the delay caused by writing the tasks into or accessing the data volumes from memory can be neglected. Hence, in this model, the task delay is mainly derived from task offloading time, the computation time and propagation delay.

1) Core network offloading latency: When the task is offloaded to the core cloud, the offloading time is highly correlated to the remaining bandwidth of the link l_c while the computation time depends on how busy the cloud currently is to execute the offloaded tasks or to run other services. The latency could be worse in peak hours when more tasks can be offloaded to the cloud while it would be busy with current running tasks.

The cloud offloading latency $t_c(l_c)$ can be computed as below:

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

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 cloud, if the task is decided to process at the edge cloud, the delay may be only caused by the residual computation capacity that can process the task. The edge offloading latency $t_e(k)$ can be computed as below:

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

2) PVs latency: Unlike the worker nodes in the core network that are usually stable, PVs should be considered as preemptible nodes due to vehicle 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 worker nodes for QoS guarantee. As for the task offloaded to PVs via wireless links, it can be considered as multi-user MIMO transmissions. In order to reduce the inter-user interference, a simple phased-zero forcing precoder is applied to manage such interference in orthogonal domain [16]. The transmission data rate for each PV p can be given as:

$$\xi_p = B log_2(1 + \gamma_p) \tag{5}$$

where γ_p denotes the interference and noise ratio (SINR) of each PV p. The channel state may fluctuate while offloading, PVs offloading latency $t(l_c)$ can be computed as below:

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

Similar to [4], we define two types of online tasks including delay-sensitive and delay-insensitive requirements where the former can be merely processed at the edge node or PVs that are closest proximity and the later can be mapped onto any network nodes (Cloud, Edge and PVs). Next, we calculate the cost of mapping the tasks into the worker nodes through different computational platforms. It notably includes total number of CPUs required to compute a task, memory, bandwidth consumption of such offloaded task and finally energy consumption for executing its task at PVs since cloud and edge computing platforms possess 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:

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

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:

$$\Xi^e_k = \Xi_{C_e}(k) + \Xi_{M_e}(k) \tag{13}$$

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:

$$e_p = \epsilon (R^p_C(n_i))^2 \tag{18}$$

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

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

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:

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

where r^c_p and r^m_p 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:

$$\mathcal{A}^c_k = \begin{cases} 1, & \text{k deployed on cloud.} \\ 0, & \text{otherwise.} \end{cases} \tag{22}$$

$$\mathcal{A}^e_k = \begin{cases} 1, & \text{k deployed on edge.} \\ 0, & \text{otherwise.} \end{cases} \tag{23}$$

$$\mathcal{A}^p_k = \begin{cases} 1, & \text{k deployed on parked vehicle.} \\ 0, & \text{otherwise.} \end{cases} \tag{24}$$

Objective:

$$Minimize \sum_{k\in K}\sum_{j\in\eth k} \Xi^c_k \mathcal{A}^c_{k_j} + \Xi^e_k \mathcal{A}^e_{k_j} + (\eta \Xi^p_k + (1-\eta)\frac{1}{\varphi^p_k})\mathcal{A}^p_{k_j} \tag{25}$$

$$w.r.t \qquad \mathcal{A}^c_{k_j}, \mathcal{A}^e_{k_j}, \mathcal{A}^p_{k_j}$$

Constraints:

$$\mathcal{A}^c_{k_j} + \mathcal{A}^e_{k_j} + \sum_{p\in N} \mathcal{A}^p_{k_j} = 1, \forall j \in \eth(k), \forall k \in K \tag{26}$$

$$\sum_{i\in\eth(k)} \mathcal{A}^p_{k_j} = \eth(k), \forall k \in K \tag{27}$$

$$R^c_C(n_i), R^e_C(n_i), R^p_C(n_i) \geq \chi_{k_j} f_{k_j}, j \in \eth(k), n_i \in N \tag{28}$$

$$R^c_M(n_i), R^e_M(n_i), R^p_M(n_i) \geq m(k_j), j \in \eth(k), n_i \in N \tag{29}$$

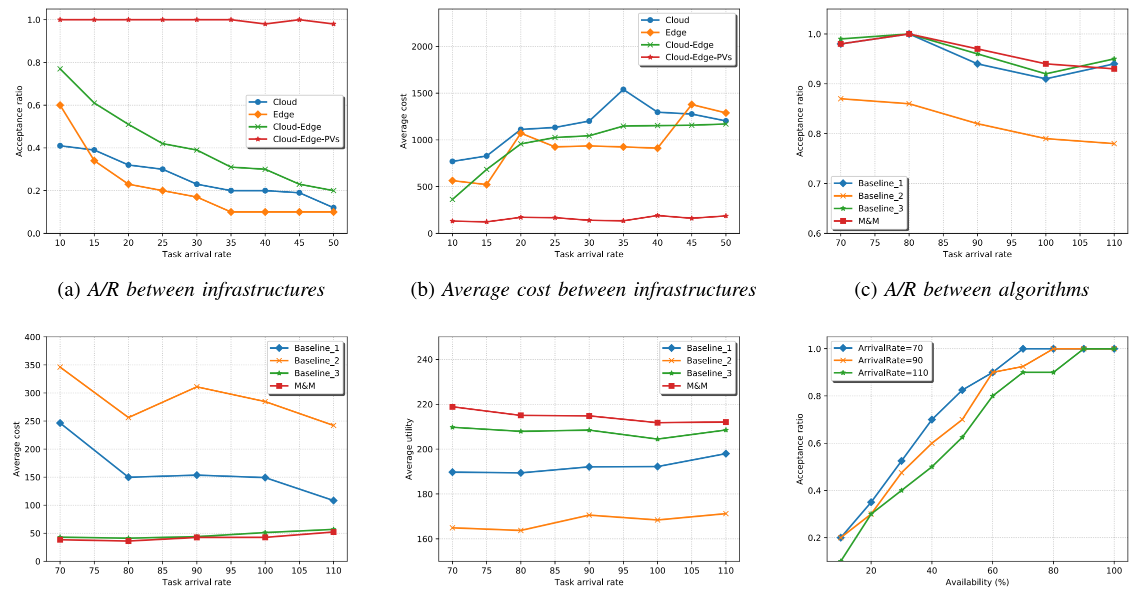
$$t_c, t_e, t_p \leq t_m(k) \tag{30}$$

Remarks:

- Function (25) comprises dual objectives that minimizes the cost of offloading computation tasks as well as maximize the PV utility when the tasks are offloaded to PVs where η is a damping factor within (0,1).
- Constraint (26) ensure that a task replication is only deployed at a single worker node.
- Constraint (27) makes sure that total number of scheduled replicas are equal to the required replicas of the corresponding task.
- Constraints (28),(29) are the Cloud, Edge and PV capacity bounds of CPU and Memory respectively.
- Constraints (30) ensures the selected nodes satisfy the latency constraints.

IV. M&M Heuristic Algorithm

BIP problems, that are computationally intractable to solve exactly, are widely known to be NP-hard. Hence, we propose a heuristic algorithm to obtain sub-optimal results by applying min-max algorithm (denoted as M&M) in aim of minimizing offloading costs while maximizing utility/rewards when given tasks are assigned to PVs. We compare M&M with three baseline algorithms, namely *Baseline_1, Baseline_2, Baseline_3*. Baseline_1 refers Kubernetes default scheduler with filtering and scoring procedures whereas Baseline_2 schedules online tasks with random policies which means that worker nodes are randomly selected for task offloading. Besides, Baseline_3 applies branch and bound selection policy as described in [17]. There are three performance metrics for evaluation including task average acceptance ratios (A/R), average costs and average accumulated utility/rewards. Details in our proposed algorithms are shown in Algorithm 1.



(a) *A/R between infrastructures* (b) *Average cost between infrastructures* (c) *A/R between algorithms*

(d) *Average cost of algorithms* (e) *Accumulated utility of algorithms* (f) *A/R towards PV availability*

Fig. 2: *Performance evaluation*

Algorithm 1 The M&M algorithm

```
 1: function GET_CANDIDATES($k$)
 2:     create an empty candidate array
 3:     for each replica of task $k$ do
 4:         for all $n \in N$ do ▷ including cloud, edge and PV nodes
 5:             if ($CPU\_remaining \geq c(k)$ and $Memory\_remaining \geq m(k)$) then
 6:                 add n to candidates
 7:             end if
 8:         end for
 9:     end for
10:     return candidates
11: end function
12: function NODE_SELECTION($k$)
13:     $utility \leftarrow zero$
14:     $candidates \leftarrow$ GET_CANDIDATES(k)
15:     sort descending all $candidates$ based on their cost values detailed in Eq. (10), (13), (19).
16:     $new\_candidates \leftarrow \lambda($total_candidates$)$ ▷ $\lambda$: candidate selection proportion
17:     for all candidates in new candidates do
18:         if candidate is a PV then
19:             calculate utility value based on Eq. (20)
20:         end if
21:     end for
22:     Select a candidate with highest utility value.
23:     return n
24: end function
```

V. Numerical Results

A. Simulation setup

We have developed a discrete event simulator to evaluate our proposed algorithms. PVs randomly and dynamically arrive and leave a generic parking lot with 100 free parking spots. Although the capacity of a parking lot can be fully occupied, we assume that its 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 services. Eligible PVs are assumed to have adequate minimum resource requirements (e.g. CPU, GPU) that can be defined by SPs, and must be registered under an idle mode to serve incoming tasks during their parking time. Moreover, PV's behaviors are based on an observation that parking duration of a PV is from 08 to 240 minutes [1] or 30 to 120 minutes [15]. More than 85% of PVs spend maximum three hours in a parking lot and the serviceability probability of a PV achieves around 90% at the parking duration of 60 minutes [1]. In this paper, we assume that the accumulative parking duration of a PV 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. When the delay tolerance of a given task exceeds 20 ms, we consider it as a delay-sensitive demand. Our simulation approximately runs for 8 hours (peak business working time) and the simulator automatically updates 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} [15], respectively. More simulation parameters can be found in Table I.

B. Performance Results

Figure 2a illustrates the effectiveness of our proposed architecture in which PVs are integrated into the ubiquitous network architecture to handle online task requests in peak hours. Cloud-Edge-PVs deployed M&M heuristic algorithm, that is extended the resource availability of the network, to increase the possibility of accepting the incoming requests. Specifically, the desired architecture improved the acceptance ratios from 23% up to 78% compared to common Cloud-Edge architecture at task arrival rates of 10 and 50, respectively.

TABLE I: Parameter Settings in Simulation

Parameter	**Values**
Maximum parking capacity	100
Total simulation time	30,000 seconds
Vehicle lifetime	[480-14400] seconds
C_c / M_c / B_c	50GHz/1000MB/1Gbps
C_e / M_e	25GHz / 500MB
$W_{\{C_c,M_c,B_c\}}$	750
$W_{\{C_e,M_e\}}$	250
$W_{\{C_p,M_p,B_p\}}$	10
Frequency	28 GHz
Channel Bandwidth B	50 MHz
CPU Parked Vehicles C_p	[1.5-2.0] GHz
Data rate ξ_p	[332-1065] Mbps
Input data size χ_k	[1.5 - 5.0] Mb
CPU cycles per bit f_k	1500 cycles
Memory requests $m(k)$	[20-50] MB
Tolerable latency of tasks t_m	(0-100] ms
Arrival request rates	[10-110]
Request replications $\eth(k)$	[2 - 10]
r_p^c / r_p^m	10 / 100
Candidate selection proportion λ	0.25

Cloud itself got lowest acceptance ratio when it was only able to execute delay-insensitive requests while Edge had a limited processing capacity.

By preferring to allocate task replicas to PVs, Cloud-Edge-PVs outperformed all compared platforms towards average offloading costs as illustrated in Fig. 2b. The proposed paradigm achieved at least 64% up to 85% better absolute average costs in a comparison with all compared infrastructures. In contrast, as our proposed paradigm achieved significantly better performance than other architectures, we would like to stress its scheduling capability with other baselines at higher task arrival rates. In performance evaluation between compared algorithms, Baseline_2 attained worst performance in terms of acceptance ratios as well as averages costs due to its allocating strategies by selecting the worker nodes randomly without considering network compatibility. Baseline_1 based on filtering and scoring procedures accepted more tasks but was still lightly worse than Baseline_3 and M&M algorithms as shown in 2c. Regarding cost evaluation metric, M&M and Baseline_3 significantly performed better than others (Fig. 2d). It was not surprised since Baseline_3 was intentionally designed for minimizing the network cost. M&M seemed just lightly better than Baseline_3, but it could not compete our proposed algorithm in utility performance as illustrated in Fig. 2e. The reason is that we simultaneously considered both cost and utility at once, and the given tasks were most likely to be assigned to PVs which expectantly produced lower unit costs. Furthermore, we evaluated the availability of PVs on different arrival rates in respect of various acceptance ratios as depicted in Fig. 2f. It has been shown that when PV availability reached 85%, PVs are able to handle online heterogeneous tasks at all selected arrival rates successfully. In lower bound, we needed at least 60% availability of PVs remaining in parking lot to obtain 80% acceptance ratio.

VI. Conclusion

In this paper, we have investigated the potential of PVs as an extension of the existing computing infrastructure for container-based task execution in peak hours. We have leveraged Kubernetes architecture as the container orchestrator deploying edge servers as master nodes, while PVs have acted as worker nodes. Extensive simulation has shown that our proposed paradigm not only extends the computation resources by taking advantage of powerful on-board computing resources of PVs, but also brings a flexible and agile architecture to task offloading problems.

Acknowledgement

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

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] *What is Kubernetes?*, 2020 (accessed May 28, 2020). [Online]. Available: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

[6] 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.

[7] 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.

[8] 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

[9] 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

[10] 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

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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)*, 2020, pp. 1–6.

[16] L. Liang, W. Xu, and X. Dong, "Low-complexity hybrid precoding in massive multiuser mimo systems," *IEEE Wireless Communications Letters*, vol. 3, no. 6, pp. 653–656, 2014.

[17] H. Zhu and C. Iluang, "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.