An Infrastructure-Assisted Workload Scheduling for Computational Resources Exploitation in Fog-Enabled Vehicular Network

Ibrahim Sorkhoh, Dariush Ebrahimi, Chadi Assi, Sanaa Sharafeddine and Maurice Khabbaz

Abstract*—The vehicle-as-a-resource is an emerging concept that allows the exploitation of the vehicles' computational resources for the purpose of executing tasks offloaded by passengers, vehicles or even an internet-of-things devices. This paper revolves around a scenario where a roadside unit located at the edge of a hierarchical multi-tier edge computing sub-network resorts to the utilization of idle vehicles computational resources through fog-enabled substructure yielding a cost-effective computational tasks offloading solution. In this context, scheduling the offload of these tasks to the appropriate vehicles is a challenging problem that is subject to the interaction of major role playing parameters. Among these parameters are: the variability of vehicles availability and their computational power, the individual tasks' weighted priorities and their deadlines, the tasks required computational power as well as the required data to upload/download. This paper proposes an infrastructure-assisted task scheduling scheme where the road side unit receives computational tasks from different sources and schedule these tasks over a computationally-capable vehicles residing within the roadside unit's range. The aim is to maximize the weighted number of admitted tasks while considering the constraints mentioned above. Compared to other works, our work broaches more realistic scenario by considering a more accurate computational task and system model. Our system considers both the latency and throughput of tasks accomplishments by maximizing the weighted number of admitted tasks while at the same time respecting the tasks accompanied deadlines. Both radio and computational resources are part of optimization problem. After proving the NP-hardness of the scheduling problem, we formulated the problem as a mixed integer linear program. A Dantzig-Wolfe decomposition algorithm is proposed which yields to a master program solvable by the Barrier algorithm and subproblems solved optimally with a polynomial time dynamic programming approach. Thorough numerical analysis and simulations are conducted in order to verify and assert the validity, correctness, and effectiveness of our approach compared to branch-and-bound and greedy algorithms.

I. Introduction

Intelligent Transportation Systems (ITSs) currently exhibit a tangible proof of a remarkable technological evolution. This is especially true since the massive amount of intelligence they have gained over time and the various digital services they can support (*e.g.* passenger safety, on-the-fly Internet access, autonomous driving) renders them as major role player in the realization of smarts cities. ITSs today constitute one of the integral industry verticals of 5G. This is a consequence of the notable evolution of vehicular networks lying at the core of ITSs and constituting their baseline infrastructure that supports the sophisticated services they provision. Yet, these systems never cease to experience a continuous upsurge of end-users demands for in-transit services and applications; these pushing towards an unparalleled cooperation between the automotive, electronics and telecommunication industries with the objective of equipping vehicles with modern ubiquitous technologies (*e.g.* server-level computerized modules, wireless communication devices, etc) that mediate the physical and digital layers of vehicular networks. Indeed, these advanced onboard digital systems offer ample computational power and resources that allow for the deployment of revolutionary Artificial Intelligence (AI) algorithms that transform vehicular networks into continuously evolving groups of connected smart entities exhibiting inordinate amounts of mobile intelligence. In particular, this evolving trend gives birth to a novel concept, namely, the Vehicle-as-a-Resource (VaaR) [1], which aims at the exploitation of the vehicular computational resources to instantiate a powerful edge computing platform.

In recent years, the conventional solution of the inflating computational processing demands in vehicular network was to offload the intensive tasks over the distant cloud servers through the Road Side Unit (RSU). Such an RSU-to-Cloud communication exhibits considerable delays that marginalize the cloud servers efficiency for handling delay-sensitive services; let alone the incurred considerable bandwidth consumption when forwarding tasks in periods of elevated offered loads. Under such circumstances, Mobile Edge Computing (MEC) presents itself as an alternative other setting that yields lower end-to-end delays through the deployment of computationally-capable servers (*a.k.a.* cloudlets) at the networks edge in close proximity to the RSU (*e.g.* [2]). However, these MEC servers inevitably find themselves unable to cope with high offered loads due to the timely unavailability of adequate computational resources; this being a limitation that obliterates MECs' short latency virtue (*e.g.* [3]). An early solution suggested in [3], [4], [5] was to organize cloudlets in a hierarchy of multiple edge-tiers. With this arrangement, lower-tier cloudlets are allowed to issue task migration requests to upper cloudlet tiers. Analytical studies were conducted in [3] and [5] demonstrating the superiority of Hierarchical MECs (H-MECs) over typical flat MECs in terms of delay and task admissibility. However, installation of multiple proprietary cloudlets and their organization in H-MECs incur considerable capital and operational expenditures, (*e.g.* [6]). This motivates the necessity of an alternative cost-minimal solution that accounts for both latency requirements as well as the tasks admissibility. This level is where Vehicular Fog Computing (VeFC) and the concept of VaaR come into play.

Fog computing is a framework that facilitates the exploitation of computational resources available in the very edge of the network. By utilizing idle resources in that level, fog computing can be expanded beyond using the infrastructure resources to exploiting the computational potential of any network edge node. Clearly, such a framework will reduce the end-to-end communication delay and increases the reliability of the service while reducing the amount of bandwidth consumed in the backhaul network. VeFC broaches the concept of VaaR by harvesting the vehicles On-Board Unit (OBU) computational capabilities.

Over a long time, if the vehicles OBUs are only dedicated for their own vehicles internal processing, their capabilities most probably will be underutilized. As these vehicles OBUs possess AI tools, other vehicles that lack such resources can improve their driving automation potentials through wireless computational tasks offloading. One of the typical example in driving automation is Augmented Reality (AR) where the system creates 3D objects in order to support the driver safety and avoid traffic jams [7]. A vehicle can request from other vehicles to sense their surrounding environments, analyze, and send back the results to the requester to support the creation of accurate AR objects that support both short-term and long-term navigation. A vehicle that tries to choose between lanes can request from other vehicles to evaluate their lanes condition based on a received or collected information in order to support the decision of the requesting vehicle [8]. Besides, Internet-of-Thing (IoT) devices and pedestrians may also benefit from the computational capabilities available on OBUs. For example, IoT devices, that require a road network status, may provide vehicles with data, or ask vehicles to execute some artificial intelligent tasks over some data and send results back to them. A pedestrian who tries to apply a heavy task, such as object recognition over a picture, can get help from nearby vehicles to perform such an energy intensive and complex task instead of doing it locally.

Enabling VeFC requires tackling several intricate challenges manifesting the need for an efficient and scalable task offloading scheme. First, the dynamism of the resource availability and the tasks' deadline makes the scheduling process a cumbersome one. The residence of vehicles within the coverage of the RSU is limited and any scheduler must consider this system variability. Second, although the vehicles OBUs do have sufficient resources, but a task offloaded to the system usually issues elevated computational demands to ensure in-time processing. Note that each vehicle prioritizes its own tasks over offloaded tasks from others. Third, since the deadline of a task is usually in order of milliseconds, providing an adequate portion of available radio spectrum should be sufficiently enough to upload and download data in a timely manner. Given these challenges, it is critical to equip the RSU with an efficient scheduler to schedule tasks on OBUs. The contributions of this paper can be categorized as:

- We propose a system, deployed on RSUs, to collect low-latency computational tasks offloading requests from different sources, and efficiently schedule them on available OBUs before exiting the RSUs' coverage range.
- We mathematically formulate the scheduling problem as a mixed integer linear program, while taking into account the dynamic availability of vehicles and their limited computational capacity.
- To combat the complexity of the problem, after proving its NP hardness, we propose a scalable solution based on Dantzig-Wolfe decomposition technique. The method, using column generation algorithm, decomposes the problem into a master and multiple pricing sub-problems. The master sub-problem is solved using barrier algorithm, and all other pricing sub-problems are solved using polynomial-time dynamic programming approach.

The remainder of the paper is as follows. Section 2 summarizes related work. Section 3 presents the system model. The problem is defined in Section 4, followed by the mathematical formulation in Section 5. The Dantzig-Wolfe decomposition and the column generation algorithm are discussed in Sections 5 and 6 respectively. We test our approach and discuss the results in Section 8, and finally conclude in Section 9.

II. Related Work

Vehicular fog computing is taking a considerable attention from academia leading to extensive studies addressing major concerns. In [9], authors used machine learning techniques to choose the best fog server deployed over a base station to be connected to a vehicle once it leaves a certain server range. The work considers the load and the location of the vehicles to accomplish an accurate prediction for the server. In [10], the authors suggested a machine learning algorithm that tries to utilize the vehicles mobility in order to minimize the delay of the tasks computation. The work proposed an architecture with three offloading modes, namely, vehicle-vehicle offloading, vehicle-RSU-vehicle offloading and pedestrian-RSU-Vehicle offloading. The work discusses why mobility can be useful to minimize the download time from a node to another. Two case studies were considered; in the first, an existing ML algorithm was combined with coded computing to make it adaptable against the changes in the network topologies and workload. In the second, they investigated the idea of replicating the tasks in order to minimize the delay. The authors in [11] suggested a three layers scheme to support traffic management. The three layers are the cloud, the cloudlet and the fog layer. The fog consists of parked and moving vehicles having a certain computation capability. With this architecture, the work proposed an algorithm that balance the load over the layers resources by distributing messages passed by vehicles over the computation resources in order to process them. A design principle for fog-enabled vehicular software-defined networking was suggested in [12]. The design was evaluated with the use case of traffic management system for fast traffic rescue using real traffic accident data. In [13], authors tried to utilize the idle computation resources of the parked electrical vehicles. The problem was formulated as a Markov decision process and solved through dynamic programming. The work in [14] suggested an offloading scheme of tasks generated by the users equipment to the vehicles based on contract theory and matching theory. The aim was to minimize the computation delay. A task is represented by its required number of

computational cycles, the upload data size and the deadline deadline. The download data size is assumed to be negligible, meaning, the result of the computation can be downloaded instantaneously. They can be scheduled in a non-preemptive manner only. A vehicle is represented by its offered amount of computational resources it is willing to share and assumed to have a fixed location. A vehicle can only be assigned one task. The work assumes a dedicated bandwidth assigned to each user equipment. First, a contract is designed relating the required performance levels to the payments issued to vehicles offering this performance. Then a two-sided matching game approach is performed to assign the tasks to the vehicles. In [15], authors proposed a method to take advantage of the possibility of dividing the tasks offloaded from vehicles into several subtasks (there is no constraint on how these tasks can be divided). The work first studied the status of the channels through hidden markov model and proposed a scheme to leverage the full parallelism of computational tasks. The authors in [16] provided a mathematical model that studies quantitatively the vehicular fog computing capabilities. The work used a realistic data acquired from tens of thousands of taxis. For data distribution application, [17] suggested a joint optimization of access mode selection and spectrum allocation while considering the randomness of the vehicular network, the edge cache and the content download delay. VeFC was applied for various applications like mobile crowd sensing [18], caching [19], traffic management [20].

For general system architecture and transmission strategies, [21] suggested a general scheme to build a framework for a vehicular edge computing system consisting of multiple RSUs and several computation resources. This work concluded with an efficient transmission strategy that reduces the V2V and V2I transmission costs while maintaining a good transmission speed. The work in [22] proposed a communication protocol for vehicular multi-access edge computing by integrating licensed Sub-6 GHz band, IEEE 802.11p and millimeter wave communications to distribute data in vehicular networks. An interesting work about the role of fog computing in the context of Information-Centric Networking (ICN) was proposed in [23]. The work suggested an integrated fog-computing and ICN architecture with on-demand caching function virtualization scheme and a communication scheme between the fog nodes and future internet nodes. A smart control was designed to manage the operations between these nodes and a cognitive resource allocation as well.

We consider a realistic and general scenario where the tasks are offloaded by pedestrians, vehicles or IoT devices. They are characterized by their upload data size, download data size, computational requirement along with their deadline and importance. A task is indivisible, it can not be divided over multiple servers. The scenario of tasks splitting can be emulated in our system model by considering a set of tasks issued by one source to be part of one larger task. A server can accept scheduling more than one task as long the computational requirements do not exceed the server capacity. As opposed to minimizing the latency, our aim is to maximize the weighted number of admitted tasks since these kind of time-sensitive tasks are accompanied with a certain deadline.

III. System Model

We consider a network scenario as depicted in Fig. 1; a RSU is located in a dense urban area and provided with wireless communication capabilities allowing it to communicate with vehicles present in its communication range.

The RSU is assumed to be equipped with edge computing capabilities and renders services to incoming requests. Requests arrive at the RSU and ask for computational processing and are assumed to be emanating from either incoming vehicles, requesting particular computing and processing beyond the resources available on the vehicle, or pedestrians or IoT devices whose computing capabilities are limited. The RSU schedules tasks and assigns them resources over the cloudlet co-located with the RSU or over an in-range vehicles (servers) with available resources demanded by the tasks. Here, it is assumed that some of the in-range vehicles may have computing capabilities that can be leveraged to offload the tasks awaiting processing. Our objective is therefore to schedule the processing of incoming tasks request either at the RSU or at in-range vehicles, and assign them enough computing resources to complete processing within their deadlines.

A. Communication Model

The network operates on a radio spectrum allocated for the communication between the clients and servers; the total spectrum width is assumed to be B and both uplink and downlink transmissions are assigned portions of this bandwidth. Transmissions are assumed to be, for simplicity, orthogonal to avoid interference. The downlink and the uplink bandwidth portions are α and β respectively ($\alpha \leq B$ and $\beta \leq B$). In our model, we decide the portion of bandwidth allocated to each of the links. Let $r(t)$ be the rate achieved on the link between a server and a client, at time t. Then, $r(t)$ is a function of the radio spectrum allocated to the link as well as the distance $d(t)$ separating the client from the server, at time t.

$$r(t) = \alpha_t \times log_2(1 + \frac{P \times d(t)^{-\sigma}}{I_r + N_0 * \alpha_t}) \tag{1}$$

where P is the transmission power, N_0 is 2 times the power spectral density, $d(t)$ is the distance between the source and destination at time t, σ is the path loss exponent and I_r is the interference. Now suppose the client uploads (download) a task of size u to/from the server, the time it takes to offload this task is $T_u = \frac{u}{r}$ where r is the transmission rate achieved during the offload, which is a function of the instantaneous rate on the link. Hence, the equation that relates the upload time T_u, the task size u and the rate $r(t)$ is:

$$u = \int_{t_0}^{T_u + t_0} r(t)dt \tag{2}$$

t_0 is the time a client starts offloading the task. $r(t)$ depends on the distance $d(t)$ between the server and the client, where $d(t) = \left[(t \times v_c^x - t \times v_s^x)^2 + (t \times v_c^y - t \times v_s^y)^2\right]^{0.5}$. Here, $v_c^x = v_c \cos(\theta_c)$ ($v_c^y = v_c \sin(\theta_c)$) and $v_s^x = v_s \cos(\theta_s)$ ($v_s^y = v_s \sin(\theta_s)$) are the x-component (y-component) of the client and the server velocities, v_c and v_s, respectively and θ_c (θ_s) is the angle between $v_c(v_s)$ and the x-axis. t is the time that

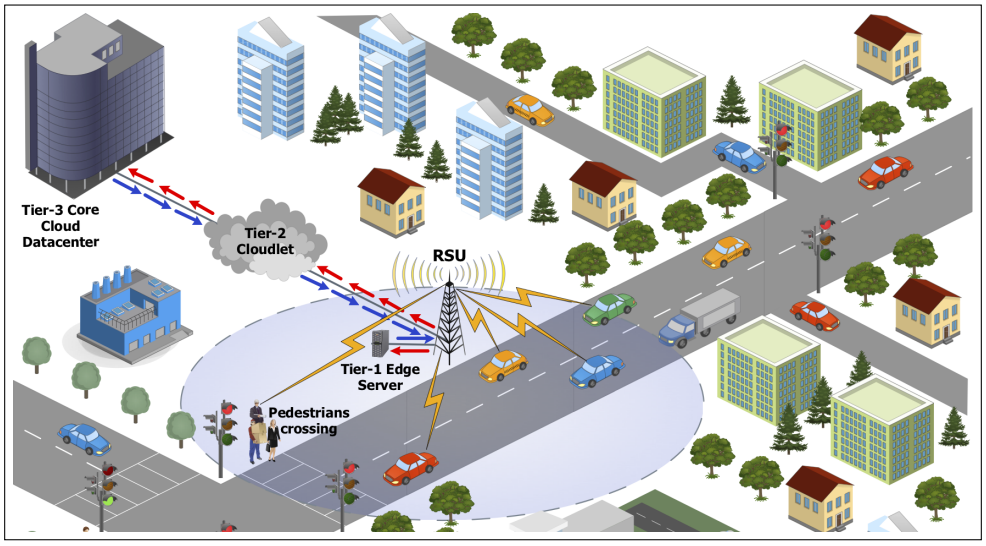

Fig. 1. Hierarchical MEC-based sub-networking scenario.

has elapsed from the start of the upload time and (0,0) being the entry point to the coverage of the RSU. This is a complex integration that is difficult to solve and particularly for this dynamic environment and for online operation. In order to overcome this, at each time slot, we calculate the distance between the clients and the servers in the middle of the slot and compute the rate accordingly.

B. Computation Model

Our system assumes there is a server available at the RSU and at each vehicle $j \in J$. The computational capacity of each vehicle is depicted by its CPU frequency f_j. For simplicity, we assume that each vehicle's OBU server contains only one CPU. Each task $i \in I$ is characterized by five deterministic values: upload and download data sizes γ_i and σ_i respectively with their computation requirement c_i, weight ω_i which represents the importance of the task to be scheduled before its deadline, and deadline δ_i. The tasks are indivisible tasks, they can not be partitioned and should be computed by only one server. A server can not start computing a task unless it receives all the required data, and can not send back the calculated result unless it completes the task computation.

C. Problem Definition

The problem input is a set of tasks $I = \{1, ..., i, ..., n_i\}$ required to be offloaded from several sources to a set of servers J available at the RSU and the vehicles' OBUs $S = \{0, 1, ..., j, ..., n_j\}$. Task i requires a certain application available only on a subset of these servers, say J_i and it is characterized by its computational cycles required c_i, its upload data size γ_i, its download data size σ_i, its deadline δ_i and its weight ω_i. The aim is to find the amount of computing resources assigned from server j to task i $f_{t,ij}$ in each time unit t. Also the radio resource/bandwidth $\alpha_{t,ij}$ (respectively $\beta_{t,ij}$) dedicated for the link used for transmission of task i client and server j. The objective is to maximize the weighted number of processed offloaded tasks. Note that the processing of a task can not start until all the required data is uploaded to the assigned vehicle. Similarly, a task cannot be downloaded until the processing of the task is finished. The distance between vehicle j and the source of task i at time unit t is indicated by d_{ij}^t.

Proposition 1. *The scheduling problem defined above is a NP-hard problem.*

Proof. Consider the well-known NP-hard scheduling problem $P|pmtn|\sum w_i U_i$ [24], where there are N tasks with deadlines to be scheduled over M parallel identical machines in order to maximize the number of admitted tasks. It is easy to show that instances of this problem can be reduced to our problem. For each task in this problem, we may create a task for our problem having the same specifications without being implicitly downloaded or uploaded to a certain vehicle. Then we create M machines similar to the above known NP-hard problem to process the maximum number of admitted tasks during the entire time line. By doing so, we reduced the scheduling problem $P|pmtn|\sum w_i U_i$ to our problem in polynomial time, hence our problem is an NP-Hard problem. □

We therefore mathematically formulate the problem as a mixed integer linear program. Then we propose a decomposition scheme based on Dantzig-Wolfe decomposition method in Section 6. This method divides the problem into a linear master problem and multiple pricing sub-problems, which can be solved in polynomial time through dynamic programming.

IV. Mathematical Model

The RSU receives offloading requests with deadlines. Requests are buffered at the RSU until the offloading decision is made; the RSU having complete knowledge of the computing resources on its co-located cloudlets, as well as the computing resources of in-range vehicles, attempts to maximize the number of admitted tasks within their deadlines. Here, the time it takes to offload the task to the cloudlet, and the computing

TABLE I
SYMBOLS USED IN FORMULATING THE PROBLEM

Symbol	Explanation
a) Parameters	
I	Set of offloaded tasks
J	Set of servers
J_i	Set of servers that have the application of task i
T	Maximum time segment
B	Total spectrum
c_i	Required number of clock cycles for task i
δ_i	Deadline of task i
γ_i	Upload data size of task i
σ_i	Download data size of task i
d_{ij}^t	Distance between vehicle j and the RSU
Δ	Time unit length
ω_i	Weight of task i
f_j	Maximum computation capacity of server j
N_0	Power spectral density
α	Path loss exponent
b) Variables	
$f_{ij} \in \mathbb{R}$	Frequency assigned for task i on server j.
$us_i^t \in \{0,1\}$	Task i is in upload stage at time t or not.
$ps_i^t \in \{0,1\}$	Task i is in processing stage at time t or not.
$ds_i^t \in \{0,1\}$	Task i is in download stage at time t or not.
$\alpha_{t,ij} \in \mathbb{R}$	Bandwidth assigned for i to be uploaded to server j.
$\beta_{t,ij} \in \mathbb{R}$	Bandwidth assigned for i to be downloaded from server j.
$x_{ij} \in \{0,1\}$	Indicates whether task i is assigned to server j or not.
$rd_{t,ij} \in \mathbb{R}$	Download data rate of task i from server j at time t.
$ru_{t,ij} \in \mathbb{R}$	Upload data rate of task i to server j at time t

time on the cloudlet plus the time it takes to send back the output, must not exceed the deadline of the task.
[1]

The objective of the mathematical model is:

$$maximize \sum_{i \in I} \sum_{j \in J_i} \omega_i x_{ij} \tag{Obj1}$$

In words, maximize the weighted number of tasks scheduled over the servers, subject to the following constraints:

1) Sum of computation resources assigned to the offloaded tasks to a server can not exceed maximum capacity.

$$\sum_{i \in I} f_{t,ij} \leq f_j \qquad \forall j \in J \qquad \forall t \leq T \tag{C1}$$

2) The computation resources assigned to a task from one server should be sufficient to finish the task.

$$\sum_{t \leq T} f_{t,ij} \times \Delta = x_{ij} c_i \qquad \forall i \in I \qquad \forall j \in J_i \tag{C2}$$

3) A task, if scheduled, is either in upload, compute, or download stage.

$$us_i^t + ps_i^t + ds_i^t = \sum_{j \in J} x_{ij} \qquad \forall i \in I \qquad \forall t < \delta_i \tag{C3}$$

4) The first time unit in the lifetime of each task must be in the upload stage unless it is decided to be rejected.

$$us_i^1 = \sum_{j \in J} x_{ij} \qquad \forall i \in I \tag{C4}$$

[1]The list of symbols is shown in Table I

5) The last time unit before the deadline of a task must be in the download stage unless it is decided to be rejected.

$$ds_i^{\delta_i} = \sum_{j \in J} x_{ij} \qquad \forall i \in I \tag{C5}$$

6) Throughout the time, a task can not be in an upload stage unless it was in the upload stage in the previous time unit.

$$us_i^t \leq us_i^{t-1} \qquad \forall i \in I \qquad : 2 \leq t \leq \delta_i \tag{C6}$$

7) In any time unit, a task can not be in a download stage unless it will be in the download stage in the next time unit.

$$ds_i^t \leq ds_i^{t+1} \qquad \forall i \in I \qquad \forall t < \delta_i \tag{C7}$$

8) After the deadline, no activities should be in progress.

$$us_i^t + ps_i^t + ds_i^t = 0 \qquad \forall i \in I \qquad \forall t \geq \delta_i \tag{C8}$$

The bandwidth used for all the transmissions happening in one time unit can not exceed the total spectrum

$$\sum_{\substack{i \in I \\ j \in J}} \beta_{t,ij} + \sum_{\substack{i \in I \\ j \in J}} \alpha_{t,ij} \leq B \qquad \forall t \leq T \tag{C9}$$

9) A task can not be assigned a computation resource from a vehicle unless it is in the computation stage.

$$f_{t,ij} \leq f_j \times ps_i^t \qquad \forall i \in I, \forall j \in J_i, \forall t \leq T \tag{C10}$$

10) In each time unit, a task can not be assigned a bandwidth to download data unless it is in the download stage.

$$\beta_{t,ij} \leq B \times ds_i^t \qquad \forall i \in I, \forall j \in J_i, \forall t \leq T \tag{C11}$$

11) In each time unit, a task can not be assigned a bandwidth to upload data unless it is in the upload stage.

$$\alpha_{t,ij} \leq B \times us_i^t \qquad \forall i \in I, \forall j \in J_i, \forall t \leq T \tag{C12}$$

12) The download transmission rate for task i is :

$$r_{ij}^{dt} = \beta_{t,ij} \times log\left[1 + \frac{p_i \times (d_{ij}^t)^{-\sigma}}{\beta_{t,ij} \times N_0}\right] \tag{3}$$

The upload transmission rate r_{ij}^{ut} can be calculated similarly.

13) For each task, the bandwidth should be assigned for the entire data to be uploaded or downloaded.

$$\sum_{t \leq T} r_{ij}^{ut} \times \Delta = \gamma_i \times x_{ij} \qquad \forall i \in I \qquad \forall j \in J_i \tag{C12a}$$

$$\sum_{t \leq T} r_{ij}^{dt} \times \Delta = \sigma_i \times x_{ij} \qquad \forall i \in I \qquad \forall j \in J_i \tag{C12b}$$

Note that constraints C12a and C12b are non-linear. To linearize them, we apply a well-known technique that is based on the calculation of the gradient of the bit rate function.

Finally, a task should only be assigned to one server.

$$\sum_{j \in J} x_{ij} \leq 1 \qquad \forall i \in I \tag{C13}$$

V. Dantzig-Wolfe Decomposition

Dantzig-Wolfe decomposition technique is a utilization of an ILP property that several constraints of a problem contain only a subset of the variables (Blocks structure). These constraints, if separated, define an easy-to-solve subproblem. The approach uses the presentation theorem of a linear programming to encode all the feasible points of the original problem as an affine combination of the subproblem feasible points.

Now, looking into our problem model, we can infer that only two constraints have variables of different tasks ($C1$ and $C9$). All the other constraints contain variables of one task. Hence, using Dantzig-Wolfe decomposition, we can decompose the problem into one master problem and N pricing subproblems.

Let X^i be the set of points that satisfies all the constraints of task i except constraints in the sets $C1$ and $C9$. Then any feasible solution for the problem can be written as affine combinations of the points in x_{ij} under the condition that it satisfies $C1$ and $C9$. Let K_i be the set of integer points in the task i feasible space. Let $\mathbf{x}_i^k$ be feasible integer point in task i feasible space. Then the problem can be re-written as follows:

$$maximize \sum_{i\in I}\sum_{j\in J_i}\omega_i \sum_{k\in K_i}\lambda_i^k x_{ij}^k$$

$$s.t.$$

$$\sum_{i\in I}\sum_{k\in K_i}\lambda_i^k f_{t,ij}^k \le f_j \qquad \forall j\in J \quad \forall t\le T \qquad \text{(C1')}$$

$$\sum_{\substack{i\in I\\ j\in J}}\sum_{k\in K_i}\lambda_i^k\left(\beta_{t,ij}^k+\alpha_{t,ij}^k\right)\le B \qquad \forall t\le T \qquad \text{(C9')}$$

$$\sum_{k\in K_i}\lambda_i^k = 1 \qquad \forall i\in I \qquad \text{(Ca)}$$

$$\lambda_i^k\in\{0,1\}//\mathbf{x}_i^k\in X_i$$

As there are exponential number of points in each X_i, an efficient way to solve the problem is by relaxing the variables to linear ones and solving the problem through column generation (CG) [25]. The next subsection will discuss our CG approach.

A. The Column Generation Algorithm

As discussed in the previous section, the possible number of columns of the master problem is exponential. The basic idea of column generation algorithm is to avoid including all the possible columns of a problem in the master model tableau. This is done by making the optimization of the master program to calculate the dual variables and feed it to the pricing subproblems. The role of the subproblems is to generate the columns with the minimum reduced cost using the given dual values. When the master model converges according to a certain criteria, the algorithm starts to solve the integer version of the master problem.

1) Initial solution: We chose to provide an initial solution with a constructive greedy heuristic. The algorithm pseudocode is shown in Algorithm 1. The algorithm starts by sorting the tasks according to their processing times in ascending order. Then for each task, it sorts the vehicles' OBUs according to their available computation resources. Then for each server, it tries to assign it that task. If it succeeds, it considers the task being scheduled. Otherwise, it resets the resources as the task was not scheduled. The input of the greedy algorithms are the set of tasks $\boldsymbol{I}$, the set of servers $\boldsymbol{J}$ and the transmission rate between each task-server pair in each time unit $\boldsymbol{R}$.

Proposition 2. *The time complexity of the greedy approach in Algorithm 1 is* $O(N\big(Mlog(M)+MT\big))$.

Proof. We can sort the tasks with merge sort with complexity $O\big(Nlog(N)\big)$. Since for each task, we are sorting the servers, this will take $O(NMlog(M))$. Checking for available server whether the task can fit or not will take $O(NMT)$. So the total complexity will be $O(O\big(Nlog(N)\big)+N\big(Mlog(M)+MT\big))$ and since the second term is more complex, we can neglect the first term. □

Proposition 3. *The greedy approach in Algorithm 1 always returns a feasible solution.*

Proof. We prove this with loop invariant. The initial step is when the algorithm tries to schedule the first task. Since the resources are totally vacant, the algorithm will not find a problem in detecting the available resources. Hence, all the "if" conditions on lines 10, 16 , and 23 will be satisfied. If the task requirement is big enough that all the available resources can not satisfy it, then the function $DeleteTask$ will free all the resources acquired by this task. Otherwise, the task will be admitted. Up to this point, the solution is feasible. Now, for all the other tasks, the "if" conditions on line 10, 16, and 23 will check whether the resources in a certain time unit is available or not. If available, the new task will be assigned this available resources, otherwise it will check other time units. This will avoid assigning two or more tasks the same resource at the same time. Again, if the task requirement is completely satisfied, the task is admitted. Otherwise, the function $DeleteTask$ will free all the resources the current task has acquired. After going through all tasks, the algorithm will terminate having a certain number of tasks being admitted and another being rejected without any overlap between the tasks assigned resources. Hence the final solution is feasible. □

2) Solving the Subproblems: Let ψ_{jt} be the dual variable corresponding to a constraint in the set $C1$, ϕ_t be the dual variable corresponding to constraint in set $C9$, and ζ_i be the dual variable of constraint Ca for task i. Then the column generation pricing subproblem for each task is modeled as:

$$\begin{aligned} minimize & \sum_{\substack{j\in J\\ t\le T}}\psi_{jt}f_{t,ij} \\ & + \sum_{t\le T}\phi_t\sum_{j\in J}\left(\alpha_{t,ij}+\beta_{t,ij}\right) \\ & - \omega_i x_{ij} \\ & + \zeta_i \end{aligned}$$

$$s.t.$$

$$\mathbf{x}_{ij}\in X_i$$

Algorithm 1 Greedy algorithm

```
1: procedure GREEDY(I, J, R)
2:     x_ij ← 0  ∀i ∈ 𝓘
3:     f̂_j ← T × f_j  ∀j ∈ J
4:     sort(I, c_i ≤ c_{i+1})
5:     bandwidthAcquired[1...T] ← false
6:     compAcquired[1...|J|][1...T] ← false
7:     for i ∈ I do
8:         uploaded ← 0
9:         downloaded ← 0
10:        computed ← 0
11:        sort(J, f̂_j ≥ f̂_{j+1})
12:        for j ∈ J do
13:            for t ← 0 : δ_i do
14:                if uploaded < γ_i then
15:                    if !bandwidthAcquired[t] then
16:                        uploaded += R[i][j][t] × Δ
17:                        bandwidthAcquired[t] ← true
18:                        β_{t,i,j} ← B
19:                else if computed < c_i then
20:                    if !compAcquired[j][t] then
21:                        computed += f_j × Δ
22:                        compAcquired[j][t] ← true
23:                        f_{t,ij} ← f_j
24:                else if downloaded < σ_i then
25:                    if !bandwidthAquired[t] then
26:                        downloaded += R[i][j][t] × Δ
27:                        bandwidthAcquired[t] ← true
28:                        α_{t,i,j} ← B
29:            if downloaded >= σ_i then
30:                x_ij ← 1
31:            else
32:                DeleteTask(i)
       return {x, f, α, β}
```

By looking into the subproblem objective function and constraints, we can state the subproblem as follows:

Definition: Given set of time units each with its own weights for being used only to upload, download or compute the task in every server, specify the process (uploading, downloading or computing) and the amount of resource used in each time unit in one server in order to minimize the total weight.

The solution space of this problem is polynomial in size. Hence, it is possible to solve the problem using a dynamic programming approach that can efficiently find the optimum solution without the need of any branch-and-bound based algorithm.

Our solution to the subproblem of a specific server-client pair is shown in algorithm 2. There are three loops in the algorithm (line 5, 11 and 17). Before the outer loop, the algorithm calls *feasible* (shown in Function 1). This function starts from a time unit given by the parameter il and adds the resources available in this time unit into a certain stage (the variable $stage$). While iterating with a certain direction (the inc operator) of the time line, the function keeps adding resources to the stage until it fulfills the task's stage requirement (the variable A) or it passes the possible time unit that can be acquired (fl). For each time unit added, it calculates its price-amount ratio and adds it to a sorted data structure (i.e., a red-black tree) for a purpose explained later. Once the function is done from adding all the necessary time units, the function calls another function called *removeExtraPrice*. This function gets from the stage the time unit with highest price-amount ratio and keeps only the necessary amount of its resource and remove the rest from the stage. The first loop in Algorithm 2 iterates starting from the last time unit in the upload stage to the end of the time line. For these time units, the algorithm calls the function *(*feasible) given the index of the upload stage as the initial time unit. After finding a feasible solution for the download stage, the algorithm starts another loop iterating over all the time units available for downloading. In this loop, the algorithm finds a feasible solution for the processing stage. The inner loop in the algorithm iterates starting from the time unit in the processing stage feasible solution until the index of the download stage. For each time unit, it calculates its price-amount ratio and checks whether it is smaller than the maximum price-amount ratio in the processing stage data structure. If it is the case, it adds the time unit to the data structure and calls *updateStage* shown in Function 2. This function removes all the extra time units having the highest price-amount ratio until it reaches the exact requirement of the stage. Once done from the third loop, the algorithm updates in the same way the the download stage and decrements its index. Once done from all the time units available for the download stage, the algorithm updates the upload stage by applying the same method applied for the download stage and the processing stage and then increments its index and repeats the entire process.

Function 1 Get feasible solution for one stage

```
1: function FEASIBLE(A, amount, price, il, fl, inc)
2:     stage.index ← il
3:     while stage.amount < A AND
4:     stage.index <= fl do
5:         stage.addCurrentIndex(price[stage.index] / amount[stage.index])
6:         inc(stage.index)
7:     removeExtraPrice(stage)
8:     return stage
```

Function 2 Remove the extra time units.

```
1: function UPDATESTAGE(stage, amount, A)
2:     index ← stage.index()
3:     maxAmount ← amount[index]
4:     while stage.amount − maxAmount > A do
5:         stage.deleteItem(index)
6:         index ← stage.index()
7:         maxAmount ← amount[index]
8:     removeExtraPrice(stage)
9:     return stage
```

Proposition 4. *The time complexity of the subproblem algorithms is* $O(T^4)$

Algorithm 2 Subproblem Solution

```
1: procedure SOLVESUBPROBLEM(i, j, ψ, ϕ, f_j, R)
2:
3:     s.up ←
4:         feasible(γ_i, R[i][j] * Δ, ϕ * S, 0, δ_i, ++)
5:     while s.up.index < δ_i do
6:
7:         s.down ← feasible(σ_i, R[i][j] * Δ,
8:                 ϕ * S, δ_i, s.up.index − −)
9:         while s.down.index
10:        > s.up.index do
11:
12:            s.proc ← feasible(c_i, f_j * Δ,
13:                    ψ_j * f_j, s.up.index, s.down.index + +)
14:            while s.proc.index < s.down.index do
15:
16:                if s.price > bests.price then
17:                    bests ← s
18:
19:                maxRatio ← s.proc.maxRatio
20:                ratio ← ψ[s.procIndex] / Δ
21:                if maxRatio > ratio then
22:                    s.proc.addCurrentIndex(ratio)
23:                    updateStage(s.proc, f_j, c_i)
24:                s.proc.index + +
25:
26:            maxRatio ← s.down.maxRatio()
27:            ratio ← ϕ[s.down.index]*S / rate[i][j][s.down.index]*Δ
28:            if maxRatio > ratio then
29:                s.down.addCurrentIndex(ratio)
30:                updateStage(s.down, rates[i][j], σ_i)
31:            s.down.index − −
32:
33:        maxRatio ← s.up.maxRatio()
34:        ratio ← ϕ[s.up.index]*S / rate[i][j][s.up.index]*Δ
35:        if maxRatio > ratio then
36:            s.up.addCurrentIndex(ratio)
37:            updateStage(s.up, rates[i][j], γ_i)
38:        s.up.index + +
39:
```

Proof. We start the proof by analysing the third loop (lines 17-32). In the worst case scenario, comparing the current solution with the best solution will always lead to copying the current solution (lines 20-23) leading to complexity of $O(T)$. Adding a new element to the data structure can be implemented with complexity $O(T)$ (line 28). The function *updateStage* upper bound is $O(T)$ as well (line 29). So for one iteration, the time complexity is $O(T)$ and since in the worst case of the number of iterations is T. Then the complexity of the loop is $O(T^2)$. Now, the second loop (lines 11-41) contain the third loop ($O(T^2)$), calculates the feasible solution of the processing stage (line 14) which can be done in $O(T)$ and updates the download stage time units (lines 34-39) which takes $O(T^2)$ leading to total complexity of $O(T^3)$. With similar analysis to the first loop (line 5), we can conclude that the total complexity is $O(T^4)$. □

Proposition 5. *The subproblem algorithm always return the optimal solution.*

Proof. Assume that the length of each stage is given. Then, each stage has a set of time units that it should choose from to fulfill its requirement while minimizing the cost given by the dual values. This problem is called the *fractional minimum cost knapsack* problem and it can be solved in polynomial time. To choose the time units, we calculate its cost-effectiveness ratio (price-amount) then sort the time units and choose the ones with the smallest ratios. The last time unit chosen should be partitioned and takes only the required amount of it so the required resources do not get exceeded hence increasing the cost.

The algorithm starts by finding the shortest time that can be assigned to the upload stage to fulfill its requirement (calling the *feasible* function). After that it checks all the possible combinations for the other two stages by going through all the time units available for the download stage, and accordingly checking all the available time units for the processing stage. After adding a new time unit for any stage, the algorithm applies the following recursive equation:

$$\textbf{if } \left(\frac{newUnit.price}{newUnit.amount} < \frac{currentUnit.price}{currentUnit.amount}\right) \textbf{ then}$$

$$price(t) = price(t-1) - price(remove(stage, newUnit.amount)) + newUnit.price$$

else

$$price(t) = price(t-1)$$

where the function $remove(s, a)$ takes a stage s and removes the time units with maximum price with total and amount equals a. To implement that, the algorithm calculates the cost-effectiveness ratio of the time unit. If one of the time units currently in the solution has a higher ratio then it adds the new time unit to the solution and remove any unnecessary time units (extra) from the solution. By doing so, the algorithm will always find the set of time units that has the minimum cost-effectiveness ratio for any length of any stage. Since the algorithm check all possible lengths of the all the stages, then the algorithm will definitely find the optimum solution. □

VI. Performance Evaluation

We study here the performance of the proposed methods, including Mix Integer Linear Programming (MILP) (to get the optimal solution), Dantzig-Wolfe decomposition method, and greedy algorithm (Greedy); we consider different performance metrics such as: execution time, performance gap, and weighted rejection rate. We then study the overall weighted

TABLE II
MEASURABLE FACTORS USED FOR THE SCHEDULING PERFORMANCE

Factors	Distribution	Mean	Variance
Tasks Arrival (tasks/s)	Exponential	4	-
Servers Capacity (GHz)	Gaussian	3	0.2
Vehicle's Velocity	Trunc. Gaussian	25	7
Vehicle's Arrival (m/s)	Geometric	p = 0.1	-
Deadline (s)	Gaussian	0.1	0.03
Upload Data Size (MB)	Gaussian	1	0.25
Download Data Size (MB)	Gaussian	0.1	0.1
Tasks Weight	Gaussian	5	3
Number of Cycles (Millions)	Gaussian	20	5
Total Spectrum (GHz)	-	3	-

rejection rate for the overall system using the Dantzig-Wolfe decomposition method by varying tasks arrival rate, the vehicles emitting probability, the average upload data size, the average number of computational cycles, and the average server size. Vehicular traffic traces are obtained using the well-known traffic simulator SUMO [26]. Table II shows the parameters used throughout this section, and the probability of application availability for a certain task on a particular server is set to 0.75. We assume an RSU with $500m$ coverage range, equipped with one server for processing the offloaded workloads. We use CPLEX to solve our optimization models and C++ to simulate the operation of our algorithms, through a discrete event driven simulation. We generate results on CPU with Intel(R) Core(TM) i7-6700 CPU @ 2.7GHz, 16GB memory ram and 64-bit mac operating system. The results are averaged over ten runs.

A. Scheduling Performance

In this subsection we analyze and evaluate the performance of our proposed Dantzig-Wolfe decomposition method. First, we study the scheduling performance of our method versus the optimal solution obtained from the MILP by considering the execution time and task rejection rate. We limit the number of iterations for the Dantzig-Wolfe decomposition method to 100 iterations. Table III shows the execution time of the three methods by varying the problem size as an input instance. The results are averaged over five samples. From the table, we can observe that, in terms of execution time, even for a small size, the Dantzig-Wolfe decomposition method always surpasses the optimal solution obtained by MILP. For instance, the computational time requires to solve for five vehicles with 100 tasks is over 18 hours which is not computationally acceptable for a real scenario. Whereas, the Dantzig-Wolfe method maintains an acceptable computational time. In addition, the Dantzig-Wolfe method maintains an acceptable deviation from the optimal solution even for large instances. Compared to the greedy method, the Dantzig-Wolfe method is capable to improve the greedy initial solution from 55% to a maximum of 71%. For instance, with 15 vehicles, as shown in the table, the MILP performs relatively much better than the greedy method. This is because as the number of vehicles increases, the number of possible solution combinations that will lead to the optimal value increases. The computational time of the Dantzig-Wolfe method gets slower with 15 vehicles compared to 5 vehicles. This is expected since the number of pricing sub-problems required to solve the column generation part of the Dantzig-Wolfe decomposition method is increased. Nevertheless, the execution time compared to the MILP is extremely incomparable. Note that, for all the input sizes, the deviation of the Dantzig-Wolfe from the optimal solution (MILP) is almost the same. Also, it should be noted that the performance of the Dantzig-Wolfe method can be improved by increasing the number of iterations, if the system can afford the time given for tasks scheduling.

From the study and analysis conducted in this section, we conclude that the Danzig-Wolfe method is the more suitable and scalable choice for task scheduling in terms of both execution time and task rejection rate. Hence, in the next section, to evaluate the performance of our system through simulation, we use this method for the scheduling stage. But first, it is interesting to show through an example how the Danzig-Wolfe method converges to an optimal solution as proceeding with the number of iterations. Figure 2 shows how the feasible solution converges for a 15-task input. In the figure, the upper-bound (i.e., the unfeasible solutions) and the lower-bound (i.e., the best obtained feasible solutions for the master problem after certain iterations) are shown by red thick and black strip lines respectively. As it can be depicted from the figure, the optimal solution is obtained after 52 iterations which takes one second. The upper bound is obtained through subtracting the reduced costs coefficients of the new columns from the current master objective value. The result is an objective value that can be achieved only if the master accepts all the new arrvied columns. A scenario that can occur in very special conditions.

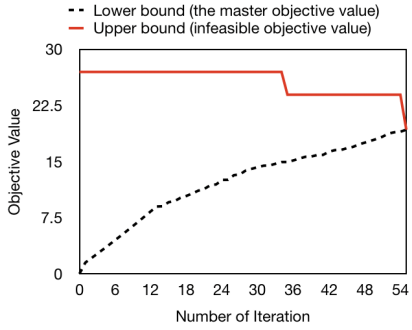

Fig. 2. The convergence of the columns generation algorithm

B. System Evaluation

In this subsection, we study the performance of our system by simulating the traffic arrival process using the well-known urban traffic simulator SUMO. In order to get random traces, SUMO requires to specify what is called the vehicle *Emitting Probability (EP)* per second per lane (SUMO is a discrete time simulator and this emitting probability is the geometric distribution parameter for each lane). In addition, it assumes that the vehicles speed follows a truncated Gaussian distribution (refer to Table II). The event driven simulation starts by running SUMO for 1 minute (this is the simulation

TABLE III
COMPARISON BETWEEN THE ALGORITHMS PERFORMANCE IN TERMS OF REJECTION RATE AND COMPUTATION TIME.

Number of Vehicles	Number of Tasks	MILP		Dantzig-Wolfe			Greedy		
		Rej. Rate	Time (s)	Rej. Rate	Dev.	Time (s)	Rej. Rate	Dev.	Time (s)
5	50	0.44%	8	18.70%	19.93%	2	86.70%	86.65%	0.38
	75	2.23%	197	20.35%	18.53%	4	92.31%	92.132%	1
	100	4.34%	66435	23.51%	18.73	47	95.23%	95.21%	4
15	50	0.45%	23.7	19.07%	19.32%	13.25	86.73%	86.66%	1
	75	0.48%	156.85	22.89%	23.24%	29.45	91.11%	91.07%	2.6
	100	0.50%	1378	20.26%	19.95%	70	93.08%	93.04%	6

time not the actual program execution time) in order to let it reach the steady state response. Then, we start collecting the information of the available vehicles in the RSU range for one minute. The information collected here contains the vehicles speed, their coordinate per unit time (which is chosen as one millisecond), their arrival and departure time to and from the RSU range. In addition, we assign each vehicle a certain computation capacity following the distribution specified in Table II. Consequently, we reset the time of the simulator and start generating tasks request events for each vehicle and device following an exponential distribution for the the inter-arrival time (refer to table II), and add the events to a sorted data structure (i.e., red-black tree). For each certain time period, we generate a scheduling event, and eventually we add it to the data structure as well. Afterwards, the event execution process begins. For each batch of tasks, we start the scheduler, and once we get the scheduling result, we update the status of the resources. All the results shown in this subsection are averaged over 20 samples.

Fig. 3 shows the system performance in terms of weighted rejection rate compared to the different tasks arrival rate per vehicle. Three EP values are considered here (0.1, 0.15 and 0.2). As shown in the figure, from 0.2 to 0.8 task arrival rate, the curves with higher EP values result in lower rejected rate. Consider here that a vehicle in our system represents a load, as well as a computation resource. Now, when the tasks arrival rate is low, along with the higher arrival vehicles, the overall system capacity maintains a low rejection rate because any vehicle arrival increases the overall system capacity. On the other hand, when the tasks arrival rate is more than 1 task/s, the system's behaviour changes as the arrived vehicles generates load high enough to make the system uses its maximum capacity. Hence, in this situation, the rejection rate increases by increasing the vehicles arrival rate even when the computational capacity of each vehicle is high.

For a point-to-point comparison, from figure 3, we can observe that the plot of Ep=0.2 starts with a lowest rejection rate. For instance, with a tasks arrival rate of 0.2 tasks/s, the rejection rates of EP=0.1, EP=0.15, and EP=0.2 are around 8.5%, 9.6% and 10% respectively. But, as we increase the tasks arrival rate, the plot of EP=0.2 inflates faster than the others, since with a higher vehicles arrival, more tasks will be generated and hence the system will be overwhelmed faster. The behaviour of the EP=0.15 curve is similar to the EP=0.2 but with a lower steep. From the tasks arrival rate of 1 task/s onwards, the behaviour of the two EPs is inverted, and the differences in number of rejection rates increases with the increase of the tasks arrival rate. For example, at the tasks arrival rate of 2 tasks/s (or 5 tasks/s), the rejection rate differences between the EP=0.1 and EP=0.15, and between EP=0.15% and EP=0.2% both are 2% (or 4%). As mentioned earlier, the critical point of the system is when the tasks arrival rate is 1 tasks/s. Clearly, we can push this point to get higher than 1 task/s by increasing the system capacity (i.e., increasing the average server size). In practice, this can only be done by increasing the capacity of the RSU server, since the system operator has no control over the vehicles computational capacity (either to increase or decrease their capacity).

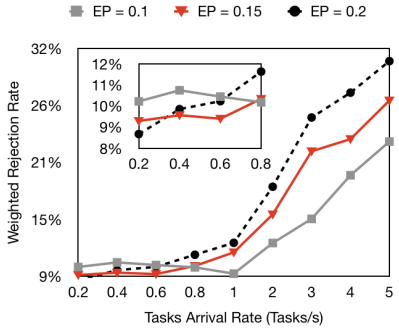

Fig. 3. Rejection rate versus tasks arrival rate per vehicle.

Figure 4 plots the weighted rejection rate of the system versus the emitting probability of the vehicles arrival per lane. The figure shows the system performance for two different tasks arrival rates per vehicles (i.e., 2 tasks/s and 4 tasks/s). While both rejection rates increase at higher EP, the differences in weighted rejection rate between the two tasks arrival rates increase as well. For instance, the relative increase in EP between 0.15 and 0.2 for the tasks arrival rate of 2 tasks/s is 22%, while for the tasks arrival rate of 4 tasks/s is 25%. Consequently, the weighted rejection rate at higher vehicles EP increases linearly. For example, at EP=0.25, the weighted rejection rate for the tasks arrival rate of 2 tasks/s is 14%, while the tasks arrival rate of 4 tasks/s is 40%. This is due to the fact that the 4 tasks/s arrival rate is reaching the system full-potential usage with higher emitting probability.

Note that the arrival load of the system can be increased either by increasing the arrival rate or by intensifying the tasks requirements. Figure 5 demonstrates the system response in terms of weighted rejection rate as we vary the average upload data size. For example, when the task data size is 1.5 MB, the weighted rejection rate of the system is around 24%, whereas, when the data size of the task is 2 MB, the weighted

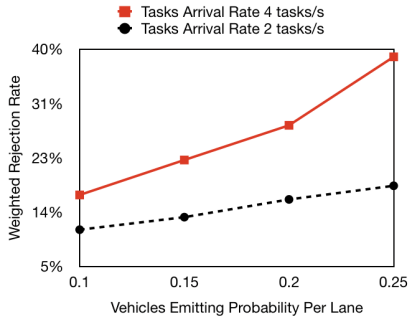

Fig. 4. Rejection rate versus vehicles arrival rate.

rejection rate is 26%. This proves that by increasing the data size of arrival tasks upto the wireless bandwidth capacity, the percentage of the weighted number of rejected tasks increases in a nonlinear manner. For instance, the relative increase in weighted rejection rate from 1.5 MB tasks data size to 2 MB is around 6%, while the increase is 10% from 2 MB to 2.5 MB data size.

Fig. 5. Rejection rate versus average upload data size per task.

In Figure 6, we calculate the rejection rate for two different average server sizes (2.5 GHz and 3.0 GHz). As depicted in the figure, for a small average number of cycles, the system is capable of admitting almost all tasks, hence, the performance of the two server sizes are almost the same. As the number of cycles per task increases, the difference between the two server sizes starts to increase. For example, for 20 million cycles, the difference between the two server sizes is around 1% while for 30 cycles, the difference reaches more than 2%.

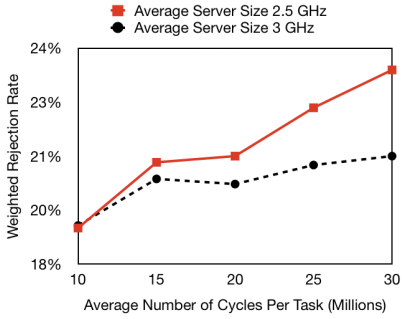

Fig. 6. Rejection rate versus average number of cycles per task.

Finally, We study the system performance by increasing the average OBU's server size. Figure 7 shows the system performance by increasing the average server size for two different average number of requires cycles (i.e., 10 million cycles and 30 million cycles). As it can be seen from the figure, for smaller server sizes, the difference between the two different number of cycles is relatively large. For example, the difference between the two is around 24% for 1 GHz average server size. By increasing the OBU's server size, the difference between the two different cycles decreases. For instance, with 2 GHz average server size, the difference between the two proposed cycles in system rejection rate is 6%, while for 3 GHz average server size, the difference is 4%. This reduction in the difference of rejection rate between the two different average cycles is due to the fact that both average load sizes can be handled easily when the size of the resource is sufficient enough to handle most of the arriving load.

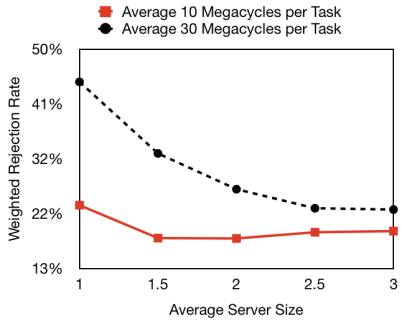

Fig. 7. Rejection rate versus OBU server average size.

VII. Conclusion

Fog computing is a promising paradigm that will allow an efficient utilization of computation resources available and accessible through wireless communication. 5G technologies with its low latency and high reliability promises can provide a platform to enable fog computing establishment upon various kinds of resources including smart vehicles. In this work, we proposed a system that can handle computation requests over vehicular network through an efficient scheduling scheme that considers the available vehicle OBU computation servers and the limited radio spectrum in order to efficiently allocate them for the requested computational tasks. The problem was formulated as an MILP with the objective to maximize the weighted number of admitted tasks. Although the problem is an NP-hard one, we proposed a scalable decomposition scheme based on Dantzig-Wolfe scheme, which resulted in polynomial-time solvable subproblems and a linear master problem. The approach showed an efficient and scalable performance compared to a greedy heuristic and MILP. Several parameters were considered in the evaluation demonstrating the robustness of our approach to solve various kinds of the problem instances.

References

[1] S. Abdelhamid, H. S. Hassanein, and G. Takahara, "Vehicle as a resource (vaar)," *IEEE Network*, vol. 29, no. 1, pp. 12–17, 2015.

[2] J. Z. K. H. Y. Mao, C. You and K. B. Letaief, "A surveyon mobile edge computing: The communication perspective," *IEEE CST*, vol. 19, no. 4, 2017.

[3] A. D. JoSEP, R. KAtz, A. KonWinSKi, L. Gunho, D. PAttERSon, and A. RABKin, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, 2010.

[4] E. El Haber, T. M. Nguyen, and C. Assi, "Joint optimization of computational cost and devices energy for task offloading in multi-tier edge-clouds," *IEEE Transactions on Communications*, vol. 67, no. 5, pp. 3407–3421, 2019.

[5] A. Kiani and N. Ansari, "Toward hierarchical mobile edge computing: An auction-based profit maximization approach," *IEEE Internet of Things Journal*, vol. 4, no. 6, pp. 2082–2091, 2017.

[6] P. S. Khodashenas, C. Ruiz, J. F. Riera, J. O. Fajardo, I. Taboada, B. Blanco, F. Liberal, J. G. Lloreda, J. Pérez-Romero, O. Sallent *et al.*, "Service provisioning and pricing methods in a multi-tenant cloud enabled ran," in *2016 IEEE Conference on Standards for Communications and Networking (CSCN)*. IEEE, 2016, pp. 1–6.

[7] Y. Xiao and C. Zhu, "Vehicular fog computing: Vision and challenges," in *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*. IEEE, 2017, pp. 6–9.

[8] C. Yu, B. Lin, P. Guo, W. Zhang, S. Li, and R. He, "Deployment and dimensioning of fog computing-based internet of vehicle infrastructure for autonomous driving," *IEEE Internet of Things Journal*, vol. 6, no. 1, pp. 149–160, 2018.

[9] S. Memon and M. Maheswaran, "Using machine learning for handover optimization in vehicular fog computing," in *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*. ACM, 2019, pp. 182–190.

[10] S. Zhou, Y. Sun, Z. Jiang, and Z. Niu, "Exploiting moving intelligence: Delay-optimized computation offloading in vehicular fog networks," *arXiv preprint arXiv:1902.09401*, 2019.

[11] Z. Ning, J. Huang, and X. Wang, "Vehicular fog computing: Enabling real-time traffic management for smart cities," *IEEE Wireless Communications*, vol. 26, no. 1, pp. 87–93, 2019.

[12] J. C. Nobre, A. M. de Souza, D. Rosario, C. Both, L. A. Villas, E. Cerqueira, T. Braun, and M. Gerla, "Vehicular software-defined networking and fog computing: integration and design principles," *Ad Hoc Networks*, vol. 82, pp. 172–181, 2019.

[13] H. M. Birhanie, M. A. Messous, S.-M. Senouci, E.-H. Aglzim, and A. M. Ahmed, "Mdp-based resource allocation scheme towards a vehicular fog computing with energy constraints," in *2018 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2018, pp. 1–6.

[14] Z. Zhou, P. Liu, J. Feng, Y. Zhang, S. Mumtaz, and J. Rodriguez, "Computation resource allocation and task assignment optimization in vehicular fog computing: A contract-matching approach," *IEEE Transactions on Vehicular Technology*, 2019.

[15] J. Xie, Y. Jia, Z. Chen, and L. Liang, "Mobility-aware task parallel offloading for vehicle fog computing," in *International Conference on Artificial Intelligence for Communications and Networks*. Springer, 2019, pp. 367–379.

[16] X. Xiao, X. Hou, X. Chen, C. Liu, and Y. Li, "Quantitative analysis for capabilities of vehicular fog computing," *Information Sciences*, 2019.

[17] S. Yan, X. Zhang, H. Xiang, and W. Wu, "Joint access mode selection and spectrum allocation for fog computing based vehicular networks," *IEEE Access*, vol. 7, pp. 17 725–17 735, 2019.

[18] Z. Zhou, J. Feng, B. Gu, B. Ai, S. Mumtaz, J. Rodriguez, and M. Guizani, "When mobile crowd sensing meets uav: Energy-efficient task assignment and route planning," *IEEE Transactions on Communications*, vol. 66, no. 11, pp. 5526–5538, 2018.

[19] P. Duan, Y. Jia, L. Liang, J. Rodriguez, K. M. S. Huq, and G. Li, "Space-reserved cooperative caching in 5g heterogeneous networks for industrial iot," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 6, pp. 2715–2724, 2018.

[20] 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.

[21] K. Zhang, Y. Mao, S. Leng, Y. He, and Y. ZHANG, "Mobile-edge computing for vehicular networks: A promising network paradigm with predictive off-loading," *IEEE Vehicular Technology Magazine*, vol. 12, no. 2, pp. 36–44, June 2017.

[22] Q. Hu, C. Wu, X. Zhao, X. Chen, Y. Ji, and T. Yoshinaga, "Vehicular multi-access edge computing with licensed sub-6 ghz, ieee 802.11p and mmwave," *IEEE Access*, vol. 6, pp. 1995–2004, 2018.

[23] J. Wu, M. Dong, K. Ota, J. Li, W. Yang, and M. Wang, "Fog-computing-enabled cognitive network function virtualization for an information-centric future internet," *IEEE Communications Magazine*, vol. 57, no. 7, pp. 48–54, 2019.

[24] P. Brucker, *Scheduling Algorithms*. SpringerVerlag, 2004.

[25] L. Wolsey, *Integer Programming*, ser. Wiley Series in Discrete Mathematics and Optimization. Wiley, 1998. [Online]. Available: https://books.google.ca/books?id=x7RvQgAACAAJ

[26] M. Behrisch, L. Bieker, J. Erdmann, and D. Krajzewicz, "Sumo–simulation of urban mobility," in *The Third International Conference on Advances in System Simulation (SIMUL 2011), Barcelona, Spain*, vol. 42, 2011.