Energy-Aware Resource Management in Vehicular Edge Computing Systems

Tayebeh Bahreini
Dept. of Computer Science
Wayne State University, USA
Email: tayebeh.bahreini@wayne.edu

Marco Brocanelli
Dept. of Computer Science
Wayne State University, USA
Email: brok@wayne.edu

Daniel Grosu
Dept. of Computer Science
Wayne State University, USA
Email: dgrosu@wayne.edu

***Abstract*—The low-latency requirements of connected electric vehicles and their increasing computing needs have led to the necessity to move computational nodes from the cloud data centers to edge nodes such as road-side units (RSU). However, offloading the workload of all the vehicles to RSUs may not scale well to an increasing number of vehicles and workloads. To solve this problem, computing nodes can be installed directly on the smart vehicles, so that each vehicle can execute the heavy workload locally, thus forming a vehicular edge computing system. On the other hand, these computational nodes may drain a considerable amount of energy in electric vehicles. It is therefore important to manage the resources of connected electric vehicles to minimize their energy consumption.**

In this paper, we propose an algorithm that manages the computing nodes of connected electric vehicles for minimized energy consumption. The algorithm achieves energy savings for connected electric vehicles by exploiting the discrete settings of computational power for various performance levels. We evaluate the proposed algorithm and show that it considerably reduces the vehicles' computational energy consumption compared to state-of-the-art baselines. Specifically, our algorithm achieves 15-85% energy savings compared to a baseline that executes workload locally and an average of 51% energy savings compared to a baseline that offloads vehicles' workloads only to RSUs.

***Keywords*-Resource Management, Vehicular Edge Computing, Energy Management.**

I. Introduction

The future increase in the amount of data and workloads (e.g., image recognition, infotainment) generated on connected electric vehicles leads to the necessity to move the computational nodes from the cloud data center closer to the vehicles [13]. In such *edge* environment, computational nodes can be deployed in edge nodes such as Road-Side Units (RSUs) so that heavy workloads of nearby vehicles can be processed with a much lower latency compared to using the cloud nodes. However, this system may have issues of scalability to an increasing number of vehicles and workloads. Due to the limited resource availability in RSUs, some vehicles may experience poor performance or even failure. To solve this problem, powerful computing nodes such as the Nvidia Drive Px 2 can be installed on each vehicle to execute most of the workload locally. In addition, vehicles can communicate with each other and with RSUs using the Dedicated Short Range Communication (DSRC) technology [1]. A system that connects the computing resources of vehicles, RSUs, and cloud is called a *Vehicular Edge Computing (VEC) system*. On the other hand, the computational energy consumption can affect the driving range of vehicles. For example, a computing node consisting of one CPU of type Intel Xeon E5-2630 and three GPUs of type NVIDIA TitanX can reduce the driving range of a Chevy Bolt by 6% [7]. However, by considering the whole system and including storage and cooling overhead, the reduction is about 11.5% [7]. Given the above challenges in VEC systems, it is desired to coordinate the available computing nodes to minimize the vehicles' energy consumption.

Previous studies have proposed various solutions for VEC systems. Unfortunately, they have at least one of two problems. First, they are *unaware of the limited energy availability*. Many previous studies on VEC systems focused on ensuring high Quality of Service (QoS) without considering the limited energy availability of electric vehicles [14], [16]. In addition, in order to minimize the risk of failure, some studies [5], [10], [17] used task replication. However, they do not balance the number of replicas with the energy consumption: having a high number of replicas may lead to a small improvement in robustness to failure while causing energy waste on vehicles. Second, they are *unaware of moving service providers*. Several solutions have been proposed to trade off between latency and energy consumption in mobile edge computing systems [6], [8], [9], [11], [12], [15]. However, most of them do not consider the case of VEC systems where the service providers, i.e., the vehicles, can quickly change their location. Without a proper consideration of the vehicles' moving pattern, the offloading mechanism may lead to a poor QoS and a higher risk of failure. In addition, as also experimented in some of the above related work (e.g., [14]), the limited computing resources at RSU level leads to blocked or dropped vehicular workloads, which inevitably leads to poor QoS or higher energy consumption for some vehicles. To the best of our knowledge, none of the above solutions consider at the same time the problems of (a) coordinating the computing resources of moving vehicles, and (b) deciding the number of replicas for vehicular workloads to minimize the vehicles' energy consumption without violating the desired QoS levels.

In this paper, we propose an energy-aware resource management algorithm for VEC systems. Rather than relying on the limited capacity of RSU nodes, the algorithm coordinates the computing resources of the vehicles to achieve energy savings. The **key-intuition for achieving energy savings** is to exploit the discrete power/performance settings of computing resources such as CPUs and GPUs. For example, CPUs commonly have a fixed number of selectable configurations for voltage-frequency levels and number of cores to trade off power consumption and performance. Each configuration leads to a maximum number of instructions that can be executed within a certain time period. If the local workload exceeds that maximum, then the system must select a new configuration that increases the performance at the cost of a higher power consumption, e.g., activate more cores or increase the voltage-frequency level. However, this selected *default* configuration may not be fully utilized by the local workload, i.e., the number of instructions executed with the local workload is lower than the maximum achievable by the default configuration. Thus, requester vehicles can achieve energy savings by offloading workloads on the providers' nodes and by using the providers' leftover capacity without changing their default configuration, i.e., without affecting the provider's power consumption. Then, at a later time, the providers can become requesters to achieve energy savings.

Based on the above intuition, an energy manager runs periodically on the local RSU to decide (a) the state of each nearby vehicle, i.e., requester or provider, and (b) the number of replicas for the requesters' workloads that minimizes the energy consumption of all the vehicles. We formulate this Energy-aware Resource Management Problem as a Mixed Integer Non-Linear Program (i.e., ERMP-MINLP). ERMP-MINLP is robust to uncertainties of vehicles' locations. However, it is also a chance-constrained optimization problem, which is not solvable in polynomial time. Thus, we propose a greedy algorithm called G-ERMP to find a solution in polynomial time. G-ERMP considers a set of probable future locations for each vehicle to minimize their energy consumption while ensuring a low risk of failure and good QoS.

In summary, this paper makes the following contributions:

- Formulate the energy-aware resource management problem (ERMP) as an MINLP.
- Design an efficient greedy algorithm (G-ERMP) to solve the ERMP problem in polynomial time.
- G-ERMP achieves 15-85% energy savings compared to a baseline that executes workload locally and an average of 51% energy savings compared to a baseline that offloads workloads only to RSUs.

The rest of the paper is organized as follows. Section II formulates the ERMP problem. Section III provides the details of our proposed G-ERMP algorithm. Section IV describes the experimental setup and results. Finally, Section V concludes the paper and discusses our future work.

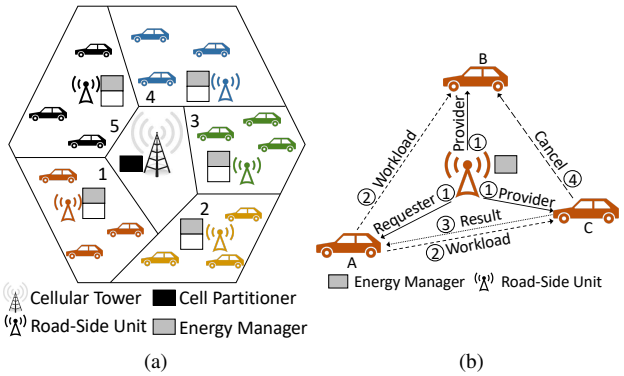


Figure 1: Overview of a VEC system: (a) the cell partitioner creates 5 edge-cells. (b) each edge-cell runs the energy manager.

II. Energy-aware Resource Management in VEC

In this section, we formulate the Energy-aware Resource Management Problem (ERMP) in VEC systems. We assume that cell towers run a *cell partitioner*, which takes the vehicles' speed and direction into consideration to periodically (at a coarse time scale) assign vehicles within its cell to *edge-cells* managed by RSUs, **so that each vehicle remains in the assigned edge-cell for most of the cell partitioner time period**. We also consider that, within each edge-cell, the local RSU periodically runs, at a finer time scale than the cell partitioner, the *energy manager*, which decides the vehicles' state (i.e., requester or provider), the number of replicas for each requester's workload, and the replicas' allocation on provider vehicles. Cell partitioner and energy manager run at different time scales and can be treated as separate problems. Figure 1(a) shows a hierarchical architecture of a VEC system. In the example of the figure, there are several vehicles and RSUs that are co-located within the cellular area of a cellular tower. Based on the information of the vehicles (i.e., speed and direction), the cell partitioner creates five edge-cells of cars. Each edge-cell is managed by a local RSU that runs the energy manager periodically. Figure 1(b) shows a single edge-cell with three vehicles and a local RSU. In this example, the energy manager (1) selects vehicle A as a service requester and vehicles B and C as service providers. It also determines two replicas for the request of vehicle A to be assigned on vehicles B and C. (2) The energy manager coordinates the replicas deployment. (3) When one of the providers returns the computation result to A (e.g., vehicle C), (4) the other providers stop the computation and wait for other workloads.

In this paper, we focus on the design of the energy manager and leave as future work the design of the cell partitioner. Note that the energy manager is distributed across RSUs and thus the above described structure can

handle vehicles moving across edge-cells after each cell-partitioner period.

Energy-Manager Problem Statement. Given the above described architecture, within each cell-partitioner period, the inputs of the energy manager are a fixed set of V vehicles, the historic location of vehicles until the current time, and the workload characteristics. We assume that the RSU runs the energy manager for F_e periods within each cell partitioner period. The length of each period is fixed and is denoted by T_{em}, which can be set to a value that satisfies the Quality of Service (QoS) for the workloads execution, i.e., every workload should execute within T_{em} units of time. Thus, the energy manager, at each one of the F_e invocations, decides the vehicle's state and the number of replicas for each selected requester so that, after F_e periods, the vehicles' energy consumption is minimized without violating the QoS requirements. In other words, we want to ensure that each vehicle, by participating in sharing computational resources, can save some energy before changing the edge-cell assignment at the next cell partitioner invocation. Next, we identify two important constraints for ERMP, define the energy model of vehicles, and formulate the optimization problem.

Capacity Constraint. We characterize the workload of vehicle j by $R_j = \{r_{1j}, \ldots, r_{Qj}\}$, where Q is the number of resource types and r_{ij} is the amount of resource of type i needed to complete the execution of the workload. We consider three resource types (i.e., $Q = 3$) indexed by h: CPU ($h = 1$), memory ($h = 2$), and storage ($h = 3$). Thus, r_{1j} is the number of CPU instructions (in millions), r_{2j} is the amount of memory, and r_{3j} is the amount of storage needed by workload of vehicle j. Each vehicle j selected as a provider has a limited capacity C_{hj} for each resource h, thus the total amount of resources requested cannot exceed the available capacity:

$$\sum_{i=1}^{V} x_{ij} \cdot r_{hi} \leq x_{jj} \cdot C_{hj} \quad \forall j, \forall h \tag{1}$$

where V is the total number of vehicles in the edge-cell and x_{ij} is a binary variable that is 1, if a replica of vehicle i is assigned to vehicle j, and 0, otherwise. The state of vehicle j is associated with the value of variable x_{jj}. Vehicle j is a requester if it does not run its workload locally, i.e., $x_{jj} = 0$, otherwise it is a provider. Thus, the above constraint also guarantees that no replica will be assigned to requester vehicles. As we describe later, we use x_{ij} to decide the number of providers and the number of requesters' workload replicas during each one of the F_e periods. However, the optimized variable x_{ij} also gives an initial placement of replicas to providers, which can be either enforced or optimized at a finer time grain for a higher QoS.

A challenge to overcome is how to calculate the CPU capacity C_{1j} in Constraint (1). We define the CPU capacity based on the Millions of Instructions per Second (MIPS) that can be executed for a certain CPU frequency and number of cores. This relation can be approximated as follows:

$$MIPS_j = (\vartheta_j \cdot f_j + \theta_j) \cdot n_j \tag{2}$$

where ϑ_j and θ_j are estimated parameters, n_j is the number of cores, and f_j is the CPU frequency of each core (assuming the same frequency for all the cores for simplicity). In order to ensure a good QoS, the required time to run the local workload r_{1j} must be shorter than the energy manager period duration T_{em}, which can be set as desired. Thus, vehicle j can satisfy this QoS requirement at the minimum energy consumption for its own computation by selecting as default the minimum frequency level that satisfies the following inequality:

$$f_j \geq \frac{r_{1j}}{\vartheta_j \cdot n_j \cdot T_{em}} - \frac{\theta_j}{\vartheta_j} \tag{3}$$

As a result, the total CPU capacity $C_{1,j}$ of Equation (1) can be calculated as follows:

$$C_{1j} = (\vartheta_j \cdot f_j + \theta_j) \cdot n_j \cdot T_{em} \tag{4}$$

For each vehicle j, the leftover capacity that can be used for requester workloads can be calculated as $MIPS_j \cdot T_{em} - r_{1j}$, where $MIPS_j$ is calculated using the default frequency f_j described above. This capacity constraint enables the energy manager to place extra workload on provider vehicles without affecting their default CPU power consumption. Thus, the energy manager can achieve energy savings for all the V vehicles in the edge-cell over multiple energy manager periods.

Risk Factor Constraint. Despite the cell partitioner efforts to provide a static vehicle set within each partitioner period, it may happen that some vehicles may change location in any of the F_e energy manager periods. Thus, because the future vehicle locations can only be predicted, we need to ensure that, with some level of confidence defined by a risk factor, each requester has a good connection with at least one provider during each period. To formulate this constraint, we first need to find the minimum distance between each requester and providers. In particular, for every selected requester, we want to have at least one provider within a reliable distance $\delta > 0$. On the other hand, the location of vehicles is non-deterministic and thus it may be affected by estimation errors. As a result, we must make sure that the probability of having at least one provider in a reliable distance is greater than a satisfaction factor $(1 - \alpha)$. This constrain can be expressed as follows:

$$p\left\{\min_{j \in \{1,\ldots,V\}} \{l_{ij} \cdot x_{ij} + M \cdot (1 - x_{ij})\} \leq \delta\right\} \geq 1 - \alpha \ \forall i \tag{5}$$

where l_{ij} is the average distance between vehicle i and vehicle j in the current period and M is a sufficiently large

number. The second term $M \cdot (1 - x_{ij})$ is needed to handle the cases in which the first term is 0 either because $i = j$ and i is a requester (i.e., $l_{ij} \cdot x_{ij} = 0$ where $l_{ij} = 0$ and $x_{ij} = 0$), or because $i \neq j$ where j is a provider not serving requests of i (i.e., $l_{ij} \cdot x_{ij} = 0$ where $l_{ij} > 0$ and $x_{ij} = 0$).

Energy Consumption. The computing system energy consumption for vehicle j is mainly characterized by two components, i.e., the computational and the transmission energy consumption. The computational energy consumption E_j^{comp} can be estimated with a linear model of the CPU clock frequency f_j and the number of cores n_j:

$$E_j^{comp} = T_{em} \cdot \left[P_j^{idle} + n_j \cdot (\lambda_j \cdot f_j + \gamma_j) \cdot x_{jj}\right] \quad (6)$$

where P_j^{idle} is the idle power consumption, λ_j and γ_j are estimated parameters. Note that when the CPU is idle because vehicle j is a requester, i.e., $x_{jj} = 0$, the computational energy consumption of the vehicles is equivalent to $T_{em} \cdot P_j^{idle}$. Therefore, the energy savings of vehicle j, if it is selected as a requester, are calculated as follows:

$$E_j^{save} = T_{em} \cdot (1 - x_{jj}) \cdot (\lambda_j \cdot f_j + \gamma_j) \cdot n_j \quad (7)$$

Note that, if vehicle j is a provider, i.e., $x_{jj} = 1$, $E_j^{save} = 0$.

The transmission energy consumption of vehicle j to receive a request from vehicle i is calculated as the ratio of the request size d_i and the average bandwidth b_{ij}, i.e., $\omega_j \cdot \frac{d_i}{b_{ij}}$. The parameter ω_j is the energy consumption of vehicle j to receive one unit of data. Also, the average bandwidth b_{ij} between vehicle i and vehicle j is proportional to their average distance l_{ij}, i.e., $b_{ij} = \frac{\mu}{l_{ij}}$, where μ is a constant estimated parameter. Therefore, the total energy consumption of vehicle j to receive requests from other vehicles, i.e., E_j^{rec}, is:

$$E_j^{rec} = T_{em} \cdot \left[\sum_{i=1}^{V} x_{ij} \cdot \omega_j \cdot \frac{d_i}{b_{ij}}\right] \quad (8)$$

Similarly, the energy consumption of vehicle j to send its request to other vehicles is:

$$E_j^{send} = T_{em} \cdot \left[\sum_{i=1}^{V} x_{ji} \cdot \psi_j \cdot \frac{d_j}{b_{ji}}\right] \quad (9)$$

where ψ_j is the energy consumption of vehicle j to send one unit of data to the network. Note, Equations (8) and (9) guarantee that $E_j^{rec} = 0$ and $E_j^{send} = 0$ if vehicle j is selected as a requester and a provider, respectively.

In order to keep track of the total energy saved and extra energy spent by each vehicle when selected as requesters or providers, respectively, we define the *energy balance*. In each energy manager period, the energy balance of vehicle j, E_j^{blnc}, is calculated based on the energy balance $E_j^{blnc'}$ obtained from the previous periods, the transmission energy, and the energy savings in the current period. In practice, *a negative energy balance means energy savings compared to always executing the workload locally*. Given the above models, the energy balance is calculated as follows:

$$E_j^{blnc} = E_j^{blnc'} + T_{em} \cdot (x_{jj} - 1) \cdot (\lambda_j \cdot f_j + \gamma_j) \cdot n_j + \sum_{i=1}^{V} x_{ji} \cdot \psi_j \cdot \frac{d_j}{b_{ji}} + \sum_{i=1}^{V} x_{ij} \cdot \omega_j \cdot \frac{d_i}{b_{ij}} \quad (10)$$

Table I: Notation

Notation	Description
V	Total number of vehicles in the edge-cell
α	Risk factor
x_{ij}	Binary decision variable for replica assignments
r_{hj}	Amount of type h resource requested by vehicle j
l_{ij}	Average distance between vehicle i and j
C_{hj}	Available capacity of resource of type h h
$E_j^{blnc'}$	Energy balance of vehicle j from previous period
P_j^{idle}	Idle power consumption of vehicle j
f_j	Default CPU frequency of vehicle j
d_j	Size of the request of vehicle j
b_{ij}	Average bandwidth between vehicle i and vehicle j
ψ_j	Transmission Energy to send one unit of data
ω_j	Transmission Energy to receive one unit of data
$\lambda_j, \theta_j, \gamma_j, \vartheta_j$	Estimated parameters

Problem Formulation. The main objective of the energy manager is to minimize the vehicles' energy balance. In fact, it would be desired to have a negative energy balance for all the vehicles before the end of the current cell partitioner period. Table I summarizes the notation used in the formulation. The Mixed Integer Non-Linear Program (MINLP) for ERMP denoted by ERMP-MINLP is defined as follows:

$$\text{Min} \quad z \quad (11)$$

s.t.:

$$\sum_{i=1}^{V} x_{ij} \cdot r_{hi} \leq x_{jj} \cdot C_{hj} \quad \forall j, \forall h \quad (12)$$

$$p\left\{\min_{j \in \{1,\ldots,V\}} \{l_{ij} \cdot x_{ij} + M \cdot (1 - x_{ij})\} \leq \delta\right\} \geq 1 - \alpha \quad \forall i \quad (13)$$

$$E_j^{blnc'} + T_{em} \cdot (x_{jj} - 1) \cdot (\lambda_j \cdot f_j + \gamma_j) + \sum_{i=1}^{V} x_{ji} \cdot \psi_j \cdot \frac{d_j}{b_{ji}} + \sum_{i=1}^{V} x_{ij} \cdot \omega_j \cdot \frac{d_i}{b_{ij}} \leq z \quad \forall j \quad (14)$$

$$x_{ij} \in \{0, 1\} \quad \forall i, \forall j \quad (15)$$

The objective function of ERMP-MINLP is to minimize the maximum energy balance of each vehicle. This is equivalent to minimizing an auxiliary variable z that is an upper bound for the energy balance of each vehicle (Constraint (14)). Constraint (15) guarantees the integrality of the decision variables. Finally, Constraints (12) and (13), as described in the previous paragraphs, enforce the available resource capacity and the desired risk factor, respectively.

III. A GREEDY ALGORITHM FOR ERMP

Due to Constraint (13), ERMP-MINLP can be classified as a chance-constrained optimization problem. As a result, it is robust to location uncertainties of the vehicles. However, solving chance-constrained optimization problems optimally usually requires computationally expensive algorithms (e.g., CPLEX [2]) that are not suitable for our setting. Therefore, we develop an efficient greedy algorithm called G-ERMP that finds a feasible solution to ERMP-MINLP in polynomial time.

G-ERMP operates in two phases using various *scenarios* to handle the chance constraint. Each scenario assumes a deterministic (i.e., known) location for vehicles. In the first phase, the algorithm picks a random sample of scenarios generated based on a set of probable locations for the vehicles. Therefore, the algorithm solves the deterministic version of ERMP-MINLP to obtain a solution for each scenario. Because the location within a scenario is known, this solution provides a single replica for each requester vehicle. Then, in the second phase, based on the assignments obtained for each scenario, the algorithm determines the number of replicas for each vehicle as well as their replica assignment so that Constraint (13) is satisfied with a probability higher than $(1-\alpha)$. However, the problem solved in the first phase of G-ERMP belongs to the class of packing problems, which are known to be NP-hard. Therefore, it is not solvable in polynomial time, unless P=NP. For this purpose, we first develop a greedy algorithm called GD-ERMP, that solves the problem associated with the first phase of G-ERMP in polynomial time for a selected scenario. Then, we describe the complete G-ERMP algorithm, which examines the solutions provided by GD-ERMP to finalize the selection of the providers and the number of replicas.

A. The GD-ERMP Algorithm

In order to minimize the maximum energy balance of vehicles, GD-ERMP analyzes the energy balance of each vehicle at the beginning of each period: vehicles with a low energy balance are more likely to be selected as the providers for the current period. However, making this decision based only on energy balance may lead to an unnecessarily high number of irregularly distributed providers. To solve this problem, our algorithm needs to consider both the location of the vehicles and their energy balance.

Algorithm 1 shows the pseudo-code of GD-ERMP. It considers both the energy balance and the location of vehicles to decide the set of providers S and the number of replicas for requesters. The input is the vector of vehicles with their request type R_j, capacity C_j, their current energy balance $E_j^{blnc'}$, and their location (in a given scenario) A_j. The output is the allocation matrix $X = \{x_{ij}\}$. In order to determine the providers, GD-ERMP first picks a vehicle with the minimum energy balance, and puts it in the set of providers S (Lines 1-2). Then, in an iterative manner, other providers are added to S. For each vehicle i that is not selected as a provider, the algorithm calls nearest-provider to find the nearest provider that has *enough capacity* to serve the vehicle (Line 7). If that provider is within a reliable distance, i.e., $l_{ij} \leq \delta$, the requester is assigned to that provider and the allocation vector y is updated (Lines 8-9). If the algorithm cannot allocate a request within a reliable distance, the current set of providers is not enough to satisfy all the requests. Hence, the algorithm needs to add another vehicle to the set of providers. As discussed in the examples above, the next provider is chosen so that it has a relatively low energy balance and it is far away from the already selected providers, which helps covering more requesters with a minimum number of providers. This strategy is implemented in Lines 11-14, where the algorithm picks a vehicle that has the maximum value of $\left(\frac{l(i,S)}{\bar{L}} - \frac{E_i^{blnc'}}{\bar{E}}\right)$, where $l(i,S)$ is the distance of vehicle i from the selected providers, i.e., $l(i,S) = \min_{j \in S} l_{ij}$. $\bar{L}$ and $\bar{E}$ are the average distance over vehicles and the average energy balance over vehicles, respectively.

The above procedure is repeated until all requests are allocated to providers that are within a reliable distance. Then, the algorithm updates the allocation matrix X based on the allocation obtained for vector y in the last iteration of the algorithm (Lines 15-18).

Complexity Analysis. The time complexity of GD-ERMP is $O(V^3)$. The main part of GD-ERMP consists of the loop in Lines 4-14, which executes $|S|$ times. In each iteration, for each non-provider vehicle, finding the nearest provider among j providers will take $O(j)$ time. Therefore, the total time complexity of GD-ERMP is $\sum_{j=1}^{|S|}(V-j)\cdot j = O(V^3)$

Algorithm 1 GD-ERMP

Input: Set of vehicles : $\mathcal{V}$
1: $j \leftarrow \text{argmin}_{i \in \mathcal{V}} E_i^{blnc'}$
2: $S \leftarrow \{j\}$
3: $stop \leftarrow$ false
4: **while not** $stop$ **do**
5: $\quad stop \leftarrow$ true
6: $\quad$ **for** $i \in \mathcal{V} - S$ **do**
7: $\quad\quad j \leftarrow$ nearest-provider(i, S, X)
8: $\quad\quad$ **if** $l_{ij} \leq \delta$ **then**
9: $\quad\quad\quad y_i \leftarrow j$
10: $\quad\quad$ **else**
11: $\quad\quad\quad j \leftarrow \text{argmax}_{i \in \mathcal{V}-S}(\frac{l(i,S)}{\bar{L}} - \frac{E_i^{blnc'}}{\bar{E}})$
12: $\quad\quad\quad S \leftarrow S \cup \{j\}$
13: $\quad\quad\quad stop \leftarrow$ false
14: $\quad\quad\quad$ **break**
15: **for** $i \in S$ **do**
16: $\quad x_{ii} \leftarrow 1$
17: **for** $i \in \mathcal{V} - S$ **do**
18: $\quad x_{iy_i} \leftarrow 1$
Output: X

Algorithm 2 G-ERMP

Input: Set of vehicles : $\mathcal{V}$
Set of scenarios :ξ

```
1:  S ← ∅
2:  for each j ∈ V do
3:      σ_j ← 0
4:  Γ ← ∅
5:  for each ε ∈ ξ do
6:      Z^ε ← GD-ERMP(V,ε)
7:  for each j ∈ V do
8:      for each i ∈ V do
9:          w_ij ← Σ_{ε∈ξ} z^ε_ij
10:     Indeg_j ← Σ_{i∈V} w_ij
11: while |Γ| < V do
12:     j ← argmax_{i∈V−Γ} Indeg_i
13:     S ← S ∪ {j}
14:     x_jj ← 1
15:     Γ ← Γ ∪ {j}
16:     for each i ∈ S do
17:         if x_ji = 1 then
18:             x_ji ← 0
19:             for each k ∈ V − Γ do
20:                 if w_ki > 0 and available(k,i,X) then
21:                     x_ki ← 1
22:                     σ_k ← σ_k + w_ki
23:                     if σ_k > (1 − α) · |ξ| then
24:                         Γ ← Γ ∪ {k}
25:     for each g ∈ V − Γ do
26:         if w_gj > 0 and available(g,j,X) then
27:             x_gj ← 1
28:             σ_g ← σ_g + w_gj
29:             if σ_g > (1 − α) · |ξ| then
30:                 Γ ← Γ ∪ {g}
```

Output: X

B. The G-ERMP Algorithm

Algorithm 2 shows the pseudo-code of G-ERMP. The algorithm has as input the set of scenarios, ξ, the vector of vehicles with their request size, R_j, and their capacity, C_j. The output is the allocation matrix $X = \{x_{ij}\}$. The main idea of G-ERMP is to create a graph based on the allocation matrices $Z^{\varepsilon} = \{z^{\varepsilon}_{ij}\}$, where z^{ε}_{ij} is 1 if vehicle i is assigned to vehicle j in scenario ε, and 0, otherwise. Each vertex of this graph represents a vehicle; each edge of the graph indicates a requester i to provider j assignment, weighted by the number of scenarios in which a request of i has been allocated to j by the GD-ERMP algorithm. Then, the algorithm partitions this graph into the set of providers and the set of requesters, and determines the number of replicas for each requester. This partitioning is done so that, for each vehicle, Constraint (13) is satisfied for more than $(1-\alpha)\cdot|\xi|$ scenarios.

G-ERMP starts with an empty set of providers S (Line 1). In each iteration of the algorithm, this set will be updated. Also, we define vector $\sigma = \{\sigma_j\}$ to store the number of scenarios in which Constraint (13) is satisfied for each vehicle j (Lines 2-3). We define Γ as a set of vehicles for which Constraint (13) is satisfied. G-ERMP initializes Γ with the empty set (Line 4). Matrix Z is used to save the allocation obtained for each scenario (Lines 5-6). Based on the allocation matrix Z, the algorithm creates a graph with V vertices. Each vertex represents a vehicle and each edge indicates a request-provider assignment. The weight of an edge from vertex i to vertex j is denoted by w_{ij} and is defined as the number of scenarios in which vehicle i is assigned to vehicle j. The indegree of vertex j, i.e., the total weight of edges adjacent to vertex j, is stored in vector $Indeg = \{Indeg_j\}$ (Lines 7-10).

In order to find the minimum number of providers, in each iteration, G-ERMP selects the vehicle as the provider that has received the maximum number of requests from the various scenarios. Therefore, it choses the vertex with the maximum indegree as a provider (Line 12). Then, it updates the set of providers, and the allocation matrix (Line 13-14). When a vehicle is selected as a provider, it runs its requests locally, which means that, for this vehicle, Constraint (13) is automatically satisfied. Thus, it adds the current provider to the set Γ (Line 15). The algorithm then updates the replica assignment of vehicles in two steps. In the first step, since vehicle j is selected as a provider, the algorithm removes all the previous assignments from vehicle j on any provider. In fact, the algorithm checks if a request from vehicle j has been assigned to a provider i, it removes that assignment and resets the corresponding allocation variable (Lines 16-18). Furthermore, since the remaining capacity of vehicle i is increased, it might be able to serve more requests. Thus, for each vehicle $k \in V - \Gamma$ willing to be assigned to vehicle i, the algorithm updates the assignment if vehicle i has enough capacity. It also updates σ for vehicle k. If σ_k is greater than $(1-\alpha)\cdot|\xi|$, the algorithm adds vehicle k to Γ. Therefore, the algorithm will not generate any further replica for that vehicle (Lines 19-24). In the second step, the algorithm assigns requests from each vehicle g willing to be assigned to vehicle j. It updates the assignment if vehicle i has enough capacity. It also updates σ for vehicle g. If σ_g is greater than $(1-\alpha)\cdot|\xi|$, the algorithm adds vehicle g to Γ (Lines 25-30). The algorithm continues this procedure until all the vehicles are added to set Γ. Due to space limitations, we refer to Sections 4 and 5 of our technical report [3] for illustrative examples to show how the proposed algorithm works.

Complexity Analysis. To investigate the time complexity of G-ERMP, we analyze the time complexity of the two main parts of the algorithm. In the first part, G-ERMP calls GD-ERMP for each scenario. Therefore, as analyzed in the previous section, the time complexity of the first part is $O(|\xi| \cdot V^3)$. In the second part, G-ERMP builds a graph based on the solution obtained in the first part. The time complexity of the second part mainly depends on the loop in

Lines 11-29, which executes $O(V)$ times. The main part of the loop consists of the loop in Lines 16-24 which executes $O(|S| \cdot (|V - |\Gamma|))$ times. Therefore, the time complexity of the second part is $O(V^3)$. As a result, the total time complexity of G-ERMP is $O(|\xi| \cdot V^3 + V^3) = O(|\xi| \cdot V^3)$

IV. Experimental analysis

A. Experimental setup

Edge-Cell Setup. We consider a time slotted system in which the location of vehicles may change from one time slot to another. Vehicles are located within a 4-way road. An RSU is located at the intersection. We assume that the coverage range of the RSU is 1 km, which is similar to the radius of DSRC technologies used on connected vehicles [1]. The RSU runs the energy manager for 10 periods. The duration of each period is 10 time slots, i.e., $T_{em} = 10$. Based on the traffic and the average speed of vehicles on each road, we initialize the location of vehicles so that they do not leave the coverage area of the RSU during the 10 energy manager periods. In our setup, we assume that the traffic on North-South road is two times heavier than the traffic on East-West road. The direction of each vehicle can be toward the intersection or away from the intersection. We also assume that the speed of vehicles on the East-West road is 20 meters per second while that of the vehicles on the North-South road is 10 meters per second. We consider that vehicles moving towards the intersection may turn to the other road with a probability calculated based on the traffic statistics. Therefore, the location of vehicles during the current energy manager period, depends on the initial location, the direction, and the speed of the vehicles on the road. Given this setup, we estimate that there could be up to 400 vehicles on the road within the considered edge-cell.

Computing Setup. Table II shows the parameters that we use to generate instances in our analysis. In this table, $U[x, y]$ indicates the uniform distribution within the interval $[x, y]$, and $N(x, y)$ indicates the normal distribution with mean x and variance y. We assume that for each type of resources, the capacity of vehicles is in the same range and does not vary significantly. Therefore, we use normal distribution for the memory and storage capacity of vehicles. The capacity of CPU depends on the default frequency of vehicles (See Equation (4)). We use as an example CPU the ARM Cortex A57. The CPU has 13 frequency levels from 700 MHz to 1,900 MHz and the difference of frequency between two consequent levels is about 100 MHz. We profile the Cortex A57 in terms of MIPS and power consumption for each frequency level to get the model parameters in Equations (4) and (6). Specifically, the maximum leftover capacity of a provider over T_{em} time slots does not exceed 30,732 million instructions. Therefore, to have a reasonable problem settings, we vary the workload of vehicles between 500 million instructions and 20,000 million instructions.

Table II: Distribution of parameters

Parameter	Distribution/Value	Parameter	Value
	low: $U[100, 5000]$	P_j^{idle}	0.1
r_{h1}	medium: $U[5000, 10000]$	ϑ_j	7.683
	heavy: $U[10000, 20000]$	μ	10
r_{h2}	$U[100, 1000]$	θ_j	-4558.52
r_{h3}	$U[100, 1000]$	F_e	10
C_{2j}	$N[5000, 1000]$	T_{em}	10
C_{3j}	$N[5000, 1000]$	γ_j	-0.741625
α	0.1	λ_j	0.00125
ω_j	0.2	ϕ_j	0.2

According to Equation (3), the default frequency of vehicles varies between 700 MHz and 900 MHz, which gives an estimated capacity between 8,198 and 94,259 million instructions to execute within a T_{em} period. We estimate the transmission energy parameters ω_j and ϕ_j based on the analysis provided in [4].

Performance Metrics. The performance of the proposed algorithm is evaluated by computing the percentage of total energy saving, which is defined as the ratio between the total energy saving of vehicles and the total baseline energy consumption (when all the vehicles run their requests locally).

$$\text{ES}(\%) = 100 \cdot \frac{\sum_{i=1}^{V}(a_i \cdot f_i + b_i) \cdot n_i \cdot (1 - x_{jj})}{\sum_{i=1}^{V}(a_i \cdot f_i + b_i) \cdot n_i} \quad (16)$$

To evaluate the fairness of the algorithms, we determine the Coefficient of Variation (CV) over energy balance of the vehicles. A lower value of CV means a more fair distribution of requests. CV is defined as the ratio of the standard deviation of E_j^{blnc} over the average energy balance across vehicles $\bar{E}$,

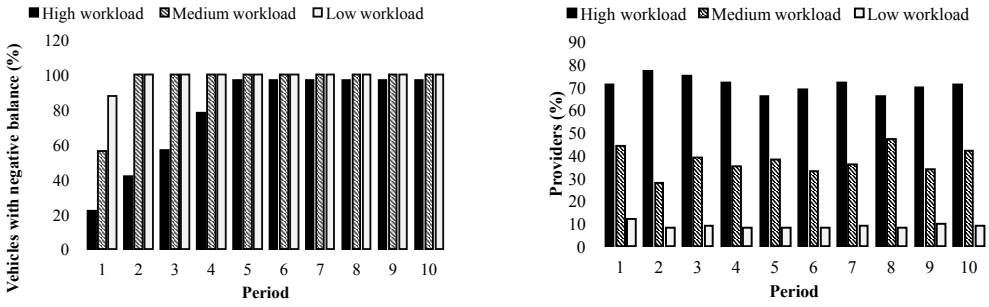
$$CV = \frac{\sqrt{\frac{1}{V}\sum_{j=1}^{V}(E_j^{blnc} - \bar{E})^2}}{\bar{E}} \quad (17)$$

G-ERMP is implemented in C++ and the experiments are conducted on an Intel 1.6GHz Core i5 with 8 GB RAM.

B. Experimental results

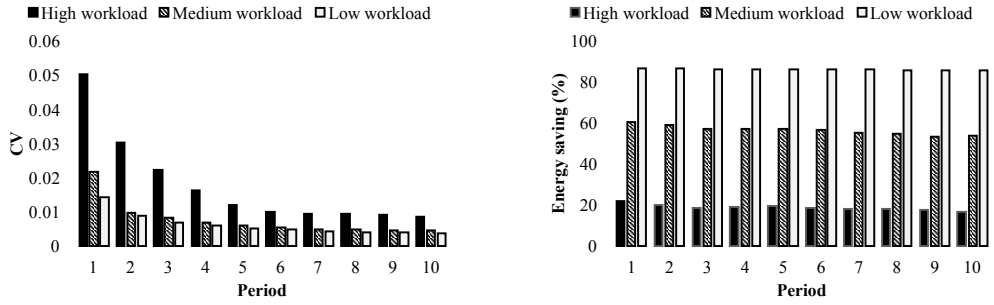
In this section, we first investigate the performance of G-ERMP compared to the baseline that executes workload of each vehicle locally for the low, medium, and high workload instances and a fixed number of vehicles. Then, we investigate the scalability of G-ERMP compared to a baseline that only offloads vehicles' workload to the local RSU while changing the number of vehicles and using random workload instances.

Fixed Number of Vehicles. Here we analyze the performance of G-ERMP by considering a fixed number of vehicles, i.e., $V = 100$, and run the algorithm in 10 energy-manager periods. We consider three sets of workload instances: low, medium, and high. In these instances, the size of transmission data of vehicles is low and is randomly chosen from $[200, 500]$ KB (see our Tech Report [3] for the tests on the effect of data transmission). The average



(a) (b)

Figure 2: G-ERMP performance with low, medium, and high workloads: (a) Percentage of vehicles with negative balance, (b) Percentage of vehicles selected as provider.



(a) (b)

Figure 3: G-ERMP performance with low, medium, and high workloads: (a) Coefficient of Variance (CV) of energy balance, and (b) Percentage of energy saving.

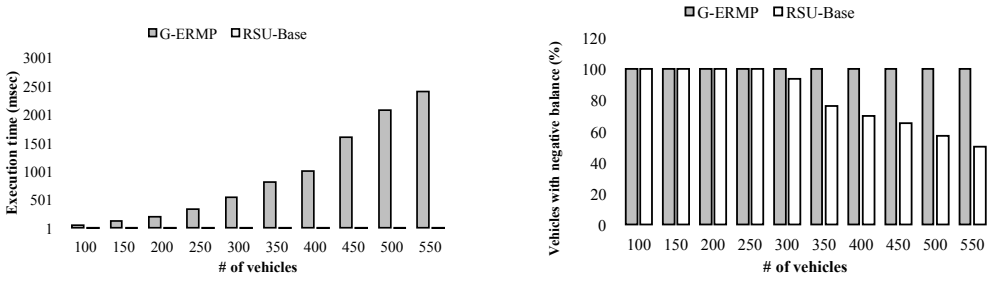
execution time of G-ERMP for each problem instance is about 0.06s, which is negligible compared to the execution period of the requests ($T_{em} = 10$s).

As Figure 2(a) shows, the percentage of vehicles with a negative balance increases over the periods. For problem instances with low and medium workload, after two periods, all vehicles obtain energy savings. However, for the heavy workload instances, only 97% of the vehicles are able to obtain energy savings because heavy workloads are more difficult to place on providers without changing their default frequency. High workloads have two consequences. First, as Figure 2(b) shows, the percentage of providers increases compared to the low workloads, and second, as Figure 3(a) shows, the CV value of high workloads is generally higher than that of the low workloads because more vehicles have to process their requests locally. This leads to having fewer vehicles achieving energy savings. On the other hand, as Figure 3(b) shows, even for the high-workload case the vehicles can achieve about 15% more energy savings compared to the baseline while the medium and low workloads achieve 50% and 85% energy savings, respectively.

These experiments show that the G-ERMP allows vehicles to achieve energy savings for their computational nodes.

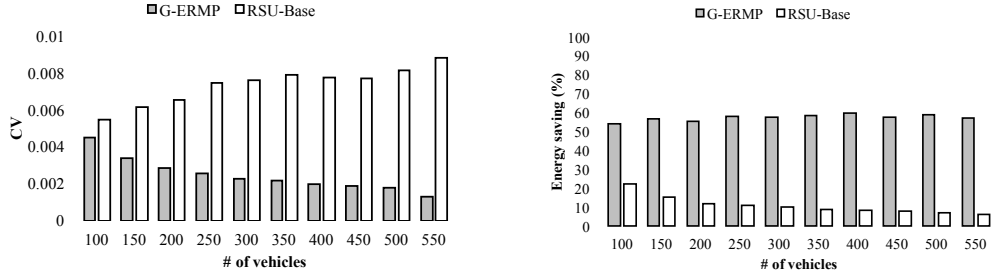
Scalability. Here, we investigate the scalability of G-ERMP to the number of vehicles and compare it to a baseline called RSU-Base, which uses the local RSU to run the request of the vehicles. RSU-Base orders the vehicle requests in descending order based on the vehicles' energy balance and then, starting from the one with highest balance, it allocates as many requests as possible on the RSU's resources.

As discussed in Section IV-A, the estimated number of vehicles that can be hosted in the edge-cell is 400. Thus, here we vary the number of vehicles from 100 to 550 so that we account also for estimation error on speed and vehicle sizes. The vehicles' workload is randomly chosen from the low, medium, and high workload. As Figure 4(a) shows, by increasing the number of vehicles, the execution time of G-ERMP increases polynomially (see analysis in Section III). However, on average, its execution time is less than 10% of the energy-manager period duration and thus is acceptable. On the other hand, as Figure 4(b) shows, G-ERMP allows all the vehicles to obtain energy savings after 10 periods while RSU-Base, because of its limited resources, cannot achieve energy savings for all the vehicles when there are more than 300 vehicles in the edge-cell. As a result, as Figure 5(a) shows, G-ERMP has a fair distribution of the energy savings (decreasing CV value) while RSU-Base has an unbalanced savings distribution (increasing CV value) due to the limited number of vehicles that can



(a) (b)

Figure 4: The effect of the number of vehicles on (a) the execution time of the algorithm, (b) the percentage of vehicles with negative balance.



(a) (b)

Figure 5: The effect of the number of vehicles on (a) the Coefficient of Variance (CV) of energy balance, and (b) the energy savings.

offload their workload. This behavior, as Figure 5(b) shows, translates in a stable 50% energy savings with G-ERMP and a decreasing amount of savings for RSU-Base with an increasing number of vehicles.

These experiments show that the proposed algorithm, G-ERMP, allows vehicles to achieve energy savings independently from the number of vehicles to manage.

V. Conclusions and Future Work

In this paper, we have proposed G-ERMP, an energy-aware resource management algorithm for VEC systems. We have evaluated G-ERMP by performing an extensive experimental analysis on several problem instances. The results have shown that the proposed algorithm has a reasonable execution time and allows vehicles to achieve 15-85% computational energy savings compared to a baseline that executes workload locally and 51% more energy savings compared to a baseline that offloads vehicles' workloads only to RSUs. In our future research we plan to develop the cell partitioner, which partitions the vehicles and assigns them to the edge-cells guaranteeing that each vehicle remains in the assigned edge-cell for most of the cell partitioner time period. In addition, we plan to improve G-ERMP to (i) formally guarantee the minimum computational energy consumption for any data transmission size, (ii) further reduce its execution time, (iii) explore the possibility of splitting large requester workloads, and (iv) allow provider vehicles to temporarily increase their default computing frequency level.

Acknowledgment

This work was supported by the NSF under grant no. IIS-1724227.

References

[1] DSRC Technology. https://www.auto-talks.com/technology/dsrc-technology.

[2] IBM ILOG CPLEX V12.1 user's manual. ftp://ftp.software.ibm.com/software/websphere/ilog/docs/, 2009.

[3] T. Bahreini, M. Brocanelli, and D. Grosu. Technical report. https://www.dropbox.com/s/v39r0z4lf68l4vo/Tech_Report.pdf?dl=0.

[4] E. Björnson and E. G. Larsson. How energy-efficient can a wireless communication system become? In *ACSSC*, 2018.

[5] Z. Jiang, S. Zhou, X. Guo, and Z. Niu. Task replication for deadline-constrained vehicular cloud computing: Optimal policy, performance analysis, and implications on road traffic. *IEEE Internet of Things Journal*, 5(1):93–107, 2018.

[6] L. Li, X. Zhang, K. Liu, F. Jiang, , and J. Peng. An energy-aware task offloading mechanism in multiuser mobile-edge cloud computing. *Mobile Information Systems*, 2018:3–15, April 2018.

[7] S.-C. Lin, Y. Zhang, C.-H. Hsu, M. Skach, M. E. Haque, L. Tang, and J. Mars. The architectural implications of autonomous driving: Constraints and acceleration. In *ACM SIGPLAN Notices*, volume 53, pages 751–766. ACM, 2018.

[8] O. Muñoz, A. Pascual-Iserte, and J. Vidal. Optimization of radio and computational resources for energy efficiency in latency-constrained application offloading. *IEEE Transactions on Vehicular Technology*, 64(10):4738–4755, 2015.

[9] S. Sardellitti, G. Scutari, and S. Barbarossa. Joint optimization of radio and computational resources for multicell mobile-edge computing. *IEEE Transactions on Signal and Information Processing over Networks*, 1(2):89–103, 2015.

[10] Y. Sun, J. Song, S. Zhou, X. Guo, and Z. Niu. Task replication for vehicular edge computing: A combinatorial multi-armed bandit based approach. *arXiv preprint arXiv:1807.05718*, 2018.

[11] H. Trinh, D. Chemodanov, S. Yao, Q. Lei, B. Zhang, F. Gao, P. Calyam, and K. Palaniappan. Energy-aware mobile edge computing for low-latency visual data processing. In *FiCloud*, 2017.

[12] H. Viswanathan, E. K. Lee, I. Rodero, and D. Pompili. Uncertainty-aware autonomic resource provisioning for mobile cloud computing. *IEEE Trans. on Parallel and Distributed Syst.*, 26(8):2363–2372, 2015.

[13] R. Yu, Y. Zhang, S. Gjessing, W. Xia, and K. Yang. Toward cloud-based vehicular networks with efficient resource management. *IEEE Network*, 27(5):48–55, September 2013.

[14] R. Yu, Y. Zhang, S. Gjessing, W. Xia, and K. Yang. Toward cloud-based vehicular networks with efficient resource management. *IEEE Network*, 27(5):48–55, 2013.

[15] K. Zhang, Y. Mao, S. Leng, Q. Zhao, L. Li, X. Peng, L. Pan, S. Maharjan, and Y. Zhang. Energy-efficient offloading for mobile edge computing in 5g heterogeneous networks. *IEEE Access*, 4:5896–5907, 2016.

[16] K. Zheng, H. Meng, P. Chatzimisios, L. Lei, and X. Shen. An smdp-based resource allocation in vehicular cloud computing systems. *IEEE Transactions on Industrial Electronics*, 62(12):7920–7928, 2015.

[17] C. Zhu, G. Pastor, Y. Xiao, Y. Li, and A. Ylae-Jaeaeski. Fog following me: Latency and quality balanced task allocation in vehicular fog computing. In *SECON*, 2018.