Energy-Aware Speculative Execution in Vehicular Edge Computing Systems

Tayebeh Bahreini
Wayne State University
tayebeh.bahreini@wayne.edu

Marco Brocanelli
Wayne State University
brok@wayne.edu

Daniel Grosu
Wayne State University
dgrosu@wayne.edu

Abstract

We address the problem of energy-aware optimization of speculative execution in vehicular edge computing systems, where multiple copies of a workload are executed on a number of different nodes to ensure high reliability and performance. The objective is to minimize the energy consumption over multiple time periods while minimizing the latency for each of the periods. We prove that the problem is NP-hard and propose a greedy algorithm to solve it in polynomial time. We evaluate the performance of the proposed algorithm by conducting an extensive experimental analysis. The experimental results indicate that the proposed algorithm obtains near optimal solutions within a reasonable amount of time.

CCS Concepts • **Networks → Mobile networks**;

Keywords Vehicular edge computing, energy consumption, speculative execution.

ACM Reference format:
Tayebeh Bahreini, Marco Brocanelli, and Daniel Grosu. 2019. Energy-Aware Speculative Execution in Vehicular Edge Computing Systems. In *Proceedings of EdgeSys '19: International Workshop on Edge Systems, Analytics and Networking, Dresden, Germany, March 25, 2019 (EdgeSys '19),* 6 pages.
https://doi.org/10.1145/3301418.3313940

1 Introduction

Modern vehicles employ several sensors and embedded systems that collect data and provide new services necessary for improving the performance of the engine and infotainment system (e.g., navigation, video streaming), reducing emissions, and assisting the driver by processing camera images (e.g., recognize the road speed limit). These workloads need a substantial amount of processing power in order to meet their performance requirements. Offloading workload to remote cloud-based computing systems is often limited by a high communication latency, which significantly degrades performance. In order to solve this problem, computational nodes have been moved from the data centers to the *edge* of the cloud, forming a Vehicular Edge Computing (VEC) system. In a VEC system, computational nodes can be deployed in cell towers, Road-Side Units (RSUs), and within connected vehicles so that local data and workloads can be processed with a much lower latency compared to using the cloud nodes. Communication among vehicles and RSUs are based on DSRC technology [1], which is characterized by a relatively low energy consumption and latency. On the other hand, these edge nodes have often limited computing capacities and energy budgets (e.g., in electric vehicles). A typical GPU for smart vehicles such as the Nvidia Drive Px 2 can consume up to 80Wh [10] while the energy consumption of an electric vehicle such as the Tesla Model 3 is on average 240Wh/mile [11]. Thus, for every mile worth of energy consumed by the vehicle, the computing system can consume 0.33 miles worth of energy. These characteristics make the problem of ensuring good performance while minimizing the energy consumption in VEC systems a challenging task.

Some of the existing challenges in VEC systems have been addressed from different perspectives and using different methods. Zheng et al. [18] addressed the variability of the resources in a VEC system. Yu et al. [16] studied the mobility of vehicles and proposed a VM migration mechanism. A variety of algorithms have been also proposed to efficiently offload workload to nearby service providers for reducing the energy consumption of mobile devices [9, 14]. However, the above solutions do not consider that multiple service requesters may interfere with each other when accessing shared service providers, which may unexpectedly increase the overall energy consumption and latency. Several studies [7, 12, 15, 17] consider the interference problem for opportunistic mobile cloud computing. However, most of those solutions have a single point of failure, i.e., if the selected service provider fails, the workload has to be migrated or offloaded again, which increases the latency. To solve this problem, various research studies [5, 13, 16, 19] have proposed the use of *speculative execution* to execute multiple copies of a workload on different nodes. However, to the best of our knowledge, none of the above solutions consider the optimization of energy consumption when employing speculative execution in VEC systems.

In this paper, we propose a *replica manager algorithm*, which is an energy-aware optimization algorithm for the speculative execution, which minimizes the workload execution time and the energy consumption of electric vehicles in VEC systems. *Our main intuition is that connected vehicles in close proximity can help each other to reduce their computational energy consumption while maintaining the desired Quality of Service (QoS).* In particular, energy savings can be achieved because of the discrete characteristics of the power management in computing systems: given a certain setting of computational power and QoS for a provider vehicle, there could be leftover computing capacity under a similar power/energy budget. This redundant computing capacity can be used to host workloads of requester vehicles with minimal impact on the energy consumption of the provider, so that requesters can achieve high energy savings. Similarly, those providers can then save energy by becoming requesters at a later time.

Figure 1 shows a speculative execution scenario in a VEC system. The replica manager runs on a local RSU, which manages all the electric vehicles in its close proximity. In this scenario, (1) vehicle A

EdgeSys '19, March 25, 2019, Dresden, Germany

ACM ISBN 978-1-4503-6275-7/19/03…$15.00
https://doi.org/10.1145/3301418.3313940

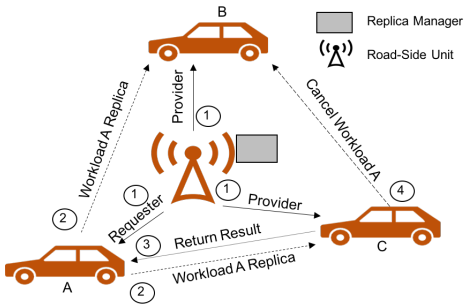


Figure 1. Speculative Execution in VEC Systems.

is a service requester while vehicles B and C are service providers. (2) The replica manager coordinates the replicas deployment so that vehicle A can save energy. (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 return, vehicle A, at a later time, will host the workload of vehicles B and C to help them lower their energy consumption. Thus, all the vehicles can save energy over a sufficiently long period of time.

Our main contributions are as follows: (1) develop a Mixed Integer Linear Programming (MILP) formulation of the energy-aware speculative execution problem in VEC systems, (2) prove that the problem is NP-hard, (3) design an efficient algorithm to solve it, and (4) evaluate the performance of the proposed algorithm. Our experiments show that proposed algorithm achieves near-optimal performance and requires a very small execution time.

The rest of the paper is organized as follows. Section 2 defines the problem and studies its complexity. Section 3 describes the proposed algorithm. Section 4 discusses the experimental results. Section 5 concludes the paper and suggests possible future directions.

2 Speculative Execution Problem

In this section, we formulate the *Speculative Execution Problem* (SEP) in VEC systems. We assume that cell towers run a *cell partitioner* that periodically runs at a coarse time scale to assign vehicles within its cell to *edge-cells* managed by RSUs so that each vehicle remain 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 1) at a finer time scale than the cell partitioner, an *energy manager*, which decides the vehicles' state (i.e., requester or provider) and the number of replicas for each requester's workload, 2) at a finer time scale than the energy manager, a *replica manager*, which determines the exact allocation of the selected requesters' workload replicas on the selected providers. These three tasks run at different time scales and can be treated as three separate problems. In this paper, we focus on the design of the replica manager and leave as future work the design of the energy manager and cell partitioner. Note, the cell partitioner takes the vehicles' speed into consideration to form edge-cells, thus we can assume that vehicles will remain in the assigned edge-cell during multiple replica manager periods. In addition, the replica manager and energy manager are distributed across RSUs and thus the above described structure can handle vehicles moving across edge-cells.

Given the above described architecture, the inputs of the replica manager are a fixed set of M providers and N requesters, the average distances between requesters and providers during the entire duration of a replica manager period, and the workload characteristics. Requester i's workload is denoted by $\{d_i, K_i, R_i\}$, where d_i is the size of the task that is to be offloaded, K_i is the number of replicas, and $R_i = \{r_{1i}, \dots r_{Qi}\}$ denotes the resources requested for the Q available resource types. We consider three main resources (i.e., $Q = 3$) indexed by h: CPU time ($h = 1$), memory space ($h = 2$), and storage space ($h = 3$). Each provider j has a limited capacity C_{hj} for each resource type. Thus, the amount of type-h resources requested from provider j should not exceed the provider's capacity:

$$\sum_{i=1}^{N} \sum_{k=1}^{K_i} r_{hi} x_{ijk} \leq C_{hj} \tag{1}$$

where x_{ijk} is a binary decision variable that is 1 if the k^{th} replica is assigned to provider j, and 0 otherwise. We assume that the providers' CPU time capacity is equivalent to the replica manager's period, i.e., all the workload must finish by the end of each period.

Table 1. Notation

Notation	Description
N	Number of of requests.
M	Number of edge providers.
Q	Number of resource types.
K_i	Set of replicas of request i.
d_i	Size of the request i.
r_{hi}	Amount of resource of type h needed for the request i
l_{ij}	Distance between requester i and provider j.
b_{ij}	Average bandwidth between requester i and provider j.
τ_{ij}	Time needed to offload request i on provider j.
θ_{ij}	Time needed to run request i on provider j.
C_{hj}	Capacity of provider j for resource of type h.
B_j	Power budget of provider j.
c_{ij}	Time to run a unit of request i on provider j.
x_{ijk}	Binary decision variable associated with the allocation of the kth replication of request i to provider j

The power consumption of computing systems is mainly characterized by two components, the offload power (due to the wireless interface) and the computational power. We assume to have a mechanism similar to that proposed in [2, 3] to analyze the workloads and find the ones that are worthy to be offloaded, i.e., those for which the computational power is much higher than the offload power. Thus, we consider only the computational power consumption of the vehicles, which is mainly influenced by the CPU utilization [8]. Thus, the power consumption of provider j is:

$$P_j = P_j^{idle} + u_j \pi_j \tag{2}$$

where P_j^{idle} is the idle power consumption, u_j is the CPU utilization, and π_j is the peak power consumption when the CPU is fully utilized. The total power consumption of provider j should not exceed its power budget, B_j, which is estimated by the RSU to ensure energy savings when the providers will become requesters. The CPU utilization u_j can be calculated as the ratio of the total CPU time requested from provider j (i.e., $\sum_{i=1}^{N} \sum_{k=1}^{K_i} r_{1i} x_{ijk}$) and the CPU capacity C_{1j} of provider j. Using this in Equation 2, we relate the decision variables x_{ijk} and the provider j's power budget:

$$P_j^{idle} + \pi_j \frac{\sum_{i=1}^{N} \sum_{k=1}^{K_i} r_{1i} x_{ijk}}{C_{1j}} \leq B_j \tag{3}$$

The total time needed to execute a request on a provider's node is calculated as the sum of the offload time and the computation time. The time τ_{ij} needed to offload request i to provider j is calculated as the ratio of the request size d_i and the average bandwidth b_{ij}, i.e., $\tau_{ij} = \frac{d_i}{b_{ij}}$. In particular, the average bandwidth b_{ij} between requester i and provider j is inversely proportional to the distance

between each other l_{ij}, i.e., $b_{ij} = \frac{\mu}{l_{ij}}$, where μ is a constant estimated parameter. The time θ_{ij} to run request i on provider j is calculated based on the requested CPU time r_{1i} and an estimated parameter c_{ij}, which depends on the type of request (e.g., CPU bounded, IO bounded), load on the provider, and the CPU clock rate of the provider. Specifically, $\theta_{ij} = r_{1i} \cdot c_{ij}$. Thus, the objective function of SEP is the total execution time of the N requests on the M providers:

$$\sum_{i=1}^{N}\sum_{j=1}^{M}\sum_{k=1}^{K_i}\alpha_j(\tau_{ij}+\theta_{ij})x_{ijk} \tag{4}$$

where α_j is the *fairness ratio*, which ensures a fair usage of the providers' energy and resources. In the first replica manager period, $\alpha_j = 1$ for $j = 1, \ldots, M$, so that we give the same weight to each provider. For a subsequent period t, α_j is calculated based on the history of normalized resource load distribution $A_j^{(t)}$, i.e., the ratio between the total resources allocated on provider j and the total resources allocated to the M providers until period $t-1$. Thus:

$$\begin{cases} \alpha_j = \frac{A_j^{(t)}}{\sum_{j=1}^{M} A_j^{(t)}} \\ A_j^{(t)} = A_j^{(t-1)} + \sum_{h=1}^{Q} \frac{r_{hi}}{\sum_{j=1}^{M} C_{hj}} r_{hi}^{(t-1)} x_{ijk}^{(t-1)} \end{cases}$$

The fairness ratio controls the load on each provider by weighting their execution time in the optimization objective. In practice, a provider that has had a higher request load in the previous periods will have a higher α and thus a longer execution time in the optimization objective than another provider that has received a lower request load. Thus, the provider with the high α value will likely receive less workload than the provider with the lower α value. Table 1 summarizes the notation used so far.

Next, we formulate the energy-aware speculative execution problem in VEC systems as an MILP (i.e., SEP-MILP):

$$\text{Minimize} \quad \sum_{i=1}^{N}\sum_{k=1}^{K_i}\sum_{j=1}^{M}\alpha_j(\tau_{ij}+\theta_{ij})x_{ijk} \tag{5}$$

subject to:

$$\sum_{k=1}^{K_i} x_{ijk} \le 1 \quad \forall i, \forall j \tag{6}$$

$$\sum_{j=1}^{M} x_{ijk} = 1 \quad \forall i, \forall k \tag{7}$$

$$\sum_{i=1}^{N}\sum_{k=1}^{K_i} r_{hi}x_{ijk} \le C_{hj} \quad \forall j, \forall h \tag{8}$$

$$P_j^{idle} + \pi_j \frac{\sum_{i=1}^{N}\sum_{k=1}^{K_i} r_{1i}x_{ijk}}{C_{1j}} \le B_j \quad \forall j \tag{9}$$

$$x_{ijk} \in \{0,1\} \quad \forall i, \forall j, \forall k \tag{10}$$

Constraints (6) ensure that K_i replicas of a request are executed by K_i different providers. Constraints (7) ensure that each replica of request i is allocated. Constraints (8) and (9) ensure that the providers' resource capacity and power budget are not exceeded. Constraints (10) guarantee the integrality of the decision variables.

Algorithm 1 G-SEP

Input: User requests: $\{d_i, K_i, R_i\}$, $i = 1, \ldots, N$
Providers capacity: $C = \{C_1, \ldots, C_M\}$, $C_j = \{C_{1j}, \ldots, C_{Qj}\}$

$X \leftarrow 0$
$\bar{X} \leftarrow 0$
$\bar{C} \leftarrow C$
Sort requests in non-increasing order of $K_i \sum_{h=1}^{Q} \frac{r_{hi}}{\sum_{j=1}^{M} C_{hj}}$
for $i = 1, \ldots, N$ **do**
  $count \leftarrow 0$
  Sort providers in non-decreasing order of $\alpha_j(\tau_{ij} + \theta_{ij})$
  for $k = 1, \ldots, K_i$ **do**
    for $j = 1, \ldots, M$ **do**
      if available($\bar{C}_j$, i) **then**
        $\bar{x}_{ijk} \leftarrow 1$
        $count \leftarrow count + 1$
        for $h = 1, \ldots, Q$ **do**
          $\bar{C}_{hj} \leftarrow \bar{C}_{hj} - r_{hi}$
        break;
  if $count = K_i$ **then**
    for $j = 1, \ldots, M$ **do**
      for $k = 1, \ldots, K_i$ **do**
        $x_{ijk} \leftarrow \bar{x}_{ijk}$
    for $h = 1, \ldots, Q$ **do**
      $C_{hj} \leftarrow \bar{C}_{hj}$
  else
    for $h = 1, \ldots, Q$ **do**
      $\bar{C}_{hj} \leftarrow C_{hj}$

Output: X

2.1 Complexity of SEP

Here we prove that SEP is an NP-hard problem, that is, it is not solvable in polynomial time, unless P = NP. For this purpose, we show that SEP has a known NP-hard problem as a special case.

Theorem 2.1. *SEP is NP-hard.*

Proof. Let us consider a special case of SEP in which there is only one type of resources ($Q = 1$) and only one replica for each request ($K_i = 1$). We also consider that the capacity of all providers is the same ($C_j = C$), and that each provider has an unlimited power budget. We call this problem SEP-SR in which SR stands for single replica. Note that since the indexes k and h have a fixed value ($k = 1$, $h = 1$), for the sake of simplicity, we ignore them from the parameters and variables.

To prove he theorem, we show that even finding a feasible solution for SEP-SR is NP-hard. In fact, finding a feasible solution for the SEP-SR problem is equivalent to solving the problem of minimizing the makespan of a set of jobs $\{1, \ldots, N\}$ on a set of parallel identical machines $\{1, \ldots, M\}$. Each job must be run on a single machine ($\sum_{j=1}^{M} x_{ij} = 1$) for a certain amount of time, r_i. The objective is to assign jobs to machines such that the makespan of each machine does not exceed C (i.e., $\sum_{i=1}^{N} r_i x_{ij} \le C$). This problem is a well-known NP-hard problem [6]. Therefore, even finding a feasible solution for SEP-SR is NP-hard. Thus, we can conclude that SEP which is the general case of SEP-SR is NP-hard. □

3 A greedy algorithm for SEP

Because SEP is NP-hard, it is not possible to find a feasible solution in polynomial time unless $P = NP$. Thus, we design G-SEP, a greedy algorithm that operates in two steps to solve SEP in polynomial time. First, G-SEP sorts request loads and providers. Second, it allocates replicas on providers one by one. In particular, to ensure a high number of allocated requests, G-SEP sorts the requests in non-increasing order of their normalized loads. Then, starting from

the request with the highest load, it tries to allocate each request one by one by finding the first provider that has enough capacity to host the request, starting from the provider that has the lowest weighted (i.e., for fairness) execution time. These steps allow to lower the overall requests' response time and to ensure a fair load distribution across providers. Next, we explain the details of G-SEP and then study its complexity.

G-SEP is given in Algorithm 1. The input of G-SEP is the vector of the requests and the capacity of the providers. The output is the allocation matrix $X = \{x_{ijk}\}$. First, G-SEP initializes the allocation matrix and the auxiliary matrices $\bar{X}$ and $\bar{C}$ (Lines 1-3). Matrices $\bar{X}$ and $\bar{C}$ are used to store the temporary values of X and C. G-SEP sorts the requests in non-increasing order of their normalized loads (line 4). The request for each type of resource is normalized with respect to the total capacity of that resource type. Then, G-SEP iteratively allocates resources to the requests. Starting from the request with the highest load, it sorts the providers in non-decreasing order of the weighted execution time (Line 7) to minimize the total communication and computation time while guaranteeing a fair allocation. Based on the sorted list of the providers, it assigns replicas one by one to providers. For each provider, it calls the function *available* to check if the provider has enough capacity and power budget for the current replica (i.e., Constraints 8 and 9). If so, then G-SEP updates the temporary capacity and allocation matrices (Lines 10-15). Then, if it finds that all the replicas of request i have been allocated to different providers (i.e., Constraints 6 and 7), it updates the allocation matrix X (Lines 16-21). Otherwise, it sets the entries of auxiliary matrix $\bar{C}$ to their previous values (Lines 22-24).

The main part of G-SEP consists of the loop in Lines 5-24, which executes N times. In each iteration, sorting the providers takes $O(M \log M)$. Then, providers are visited one by one for each replica of the current request (Lines 8-15), which takes $O(K_i MQ)$. Finally, the time complexity of Lines 16-24 is $O(K_i M + Q)$. We can ignore K_i and Q here because they generally have much smaller values compared to N and M. Thus, the time complexity of G-SEP is $O(NM \log M)$, which is polynomial in M and N.

4 Experimental Analysis

4.1 Experimental setup

We generate several problem instances with different numbers of requesters and providers. Table 2 shows the parameters that we use to generate instances in our analysis, where $U[x, y]$ indicates the uniform distribution within the interval $[x, y]$. Note that the numbers in Table 2 are chosen to represent various realistic heterogeneous hardware and software characteristics, so that we can diversify the scenarios tested without having to test just one particular hardware and software testbed. To compare the algorithms, we use the average response time as the performance metric. The average response time is defined as the total latency of the system over the total number of allocated replicas. An allocation algorithm is completely fair if the value of the fairness ratio α_j is the same for all the providers. To evaluate the fairness of the algorithms, we determine the Coefficient of Variation (CV) over fairness ratios of the providers. A lower value of CV indicates a more fair distribution of requests. CV is defined as the ratio of the standard deviation of α_j over the average fairness ratio across providers $\bar{\alpha}$:

$$CV = \frac{\sqrt{\frac{1}{M}\sum_{j=1}^{M}(\alpha_j - \bar{\alpha})^2}}{\bar{\alpha}} \tag{11}$$

Table 2. Distribution of parameters

Parameter	Distribution	Parameter	Distribution
r_{hj}	$U[1, 25]$	μ	10
K_i	$U[1, 5]$	B_j	$U[200, 400]$
d_i	$U[5, 100]$	π_j	$U[100, 300]$
C_{1j}	50	c_{ij}	$U[10, 1000]$
C_{2j}	$U[5, 100]$	P_j^{idle}	$U[20, 80]$
C_{3j}	$U[5, 100]$		

G-SEP is implemented in C++ and the experiments are conducted on an Intel 1.6GHz Core i5 with 8 GB RAM system. For solving SEP-MILP, we use the CPLEX 12 solver provided by IBM ILOG CPLEX optimization studio for academics initiative [4].

4.2 Experimental results

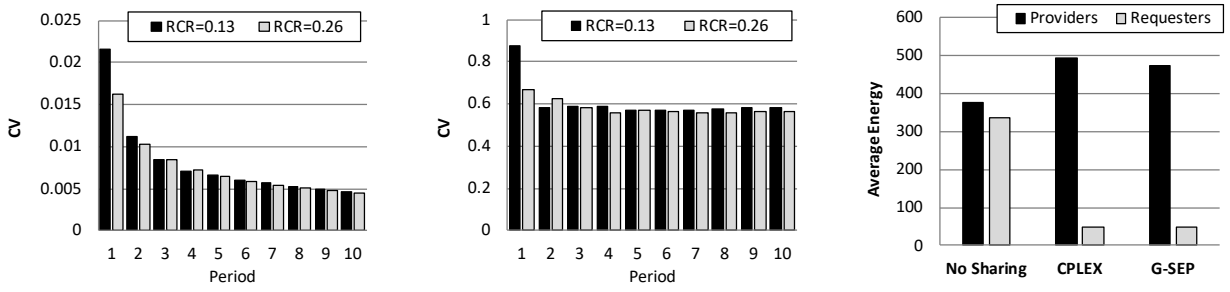
In this section, we study the fairness and the energy consumption of the vehicles and compare the performance and the scalability of G-SEP and CPLEX-based solution for problem instances with varying number of requesters and providers.

Fairness and energy consumption. We first investigate how the fairness ratio changes over several periods of time and then study the vehicles' energy consumption. Intuitively, the ratio between the total amount of requested resources and the total available capacity plays an important role on the results. For example, it may be more difficult to achieve complete fairness when the number of requests is much smaller than the number of providers. In order to characterize the relationship between the requested resources and the capacity, we define the Request to Capacity Ratio (RCR) as:

$$RCR = \frac{\sum_{i=1}^{N}\sum_{h=1}^{Q} r_{hi}}{\sum_{i=1}^{N}\sum_{h=1}^{Q} C_{hj}} \tag{12}$$

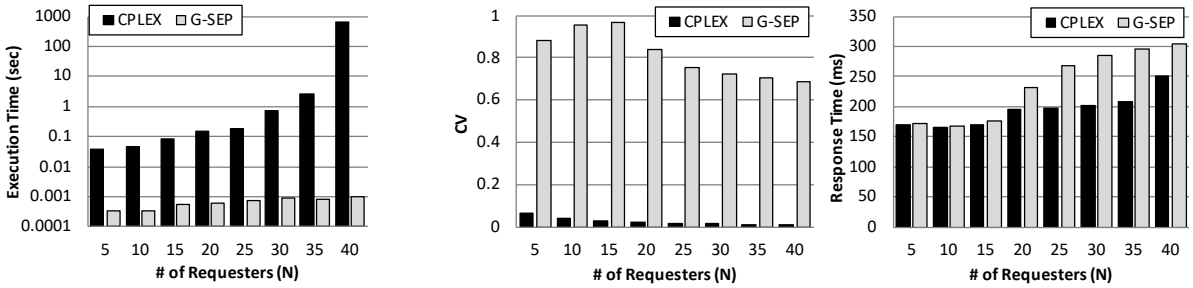
We consider two problem instances. In both instances, the number of requests is the same, $N = 20$. The number of providers in the first set is $M = 15$, while in the second set is $M = 30$, i.e., $RCR = 0.26$ and $RCR = 0.13$, respectively. We run CPLEX and G-SEP for both problem instances for ten time periods.

Figures 2(a) and 2(b) show the Coefficient of Variance, CV, obtained by CPLEX and G-SEP, respectively, for the two problem instances. By comparing these two figures, we make two observations. First, the instance with the lower RCR has a higher CV for both algorithms, which means that the allocation algorithms have more difficulties on ensuring fairness when the number of requests is not sufficiently high. This is because some of the providers may have no assignment in some periods. However, we observe that, for both instances and algorithms, generally the CV decreases over time, which means that the algorithms can improve the fairness over multiple executions. Second, CPLEX can generally achieve a better fairness, i.e., a lower CV value, compared to G-SEP. For example, the CV for CPLEX is always smaller than 0.025 during all the periods while the CV for G-SEP is always smaller than 0.9. Although distributions with a CV lower than 1 are generally considered as low-variance distributions, i.e., G-SEP achieves good fairness, this difference in achieved fairness between the two algorithms is due to the fact that CPLEX generally considers more allocation options compared to G-SEP. However, as we show later,



(a) CPLEX **(b)** G-SEP **(c)** Energy

Figure 2. Coefficient of Variance (*CV*) of fairness ratio over time for (a) CPLEX and (b) G-SEP. The Replica Manager distributes the workload of requesters to providers to make sure that, compared to the baseline *No Sharing*, the extra energy consumption of the providers is lower than the energy saved by requesters (c). Thus, those providers can achieve energy savings by becoming requesters in future executions.



(a) **(b)** **(c)**

Figure 3. The effect of the number of requests (*N*) on (a) Execution time of the placement algorithm, (b) Coefficient of Variance (*CV*) of fairness ratio, and (c) average workload response time.

CPLEX has a much higher execution time compared to G-SEP, which means that CPLEX cannot be used in real scenarios.

We now investigate the computational energy consumption of the vehicles. In general, it is desired to have the extra amount of energy used by the providers much smaller than the amount of energy saved by the requesters. This would mean that, in the next periods, those providers will be able to actually save energy when they will become requesters, even after using some energy to host the workload of other vehicles. For this study, we use a problem instance with an *RCR* of 0.12, $N = 10$, and $M = 25$, to test the energy consumption when it is more difficult to achieve a fair load distribution across providers (due to space limitations, we omit the results for other combinations of N and M). We run CPLEX and G-SEP ten times to simulate ten periods of replica manager and show the average energy consumption of provider and requester vehicles. As the baseline for comparison, we calculate the vehicles' energy consumption when they do not share their workload, i.e., *No Sharing*. Figure 2(c) shows the results. On average, CPLEX and G-SEP achieve similar results: sharing the resources allow requester vehicles to save 85% of their computational energy consumption compared to the No Sharing baseline, while providers have an average of 28% increase in energy consumption. Thus, those providers will be able to achieve energy savings when they will become requesters in future executions.

Performance and scalability with respect to the number of requesters. We now test the performance of CPLEX and G-SEP by varying the number of requests. We assume a fixed number of providers ($M = 40$) and vary the number of requests from 5 to 40. We stop at 40 because, with more requests, there is no feasible solution that satisfies Constraints 7-9 of the SEP-MILP.

In Figure 3(a), we compare the execution time of CPLEX and G-SEP. By increasing the number of requests, the execution time of CPLEX increases exponentially, thus making CPLEX not scalable to a large number of requesters. For example, CPLEX finds the optimal solution for an instance with $N = 5$ in less than 0.1 seconds while, to solve an instance with 40 requests, CPLEX requires 619 seconds. This is because, by increasing the number of requests, it is harder for CPLEX to find a feasible solution. On the other hand, the execution time of G-SEP for fixed M increases linearly with the number of requests (as proved in Section 3) and its execution time is smaller than 1 millisecond in the worst case, while allocating more than 85% of the requests. Thus, compared to CPLEX, G-SEP is more scalable with respect to the number of requester vehicles.

Figure 3(b) shows the *CV* obtained by CPLEX and G-SEP. For both algorithms, the value of *CV* generally decreases by increasing the number of requests because, when there are a few requests in the system, many providers may have no assignment, which leads to a high value of *CV*. The value of *CV* obtained by CPLEX is always lower than 0.05. Although CPLEX shows a better fairness, the *CV* of G-SEP is always lower than one for all instances, which still indicates a good distribution across providers. Figure 3(c) shows the average request response time obtained with each algorithm. By increasing the number of requests, we observe an increment in the average response time because each request has less options for the assignment. By comparing the results of the two algorithms, we can observe that CPLEX, although it needs a long execution time

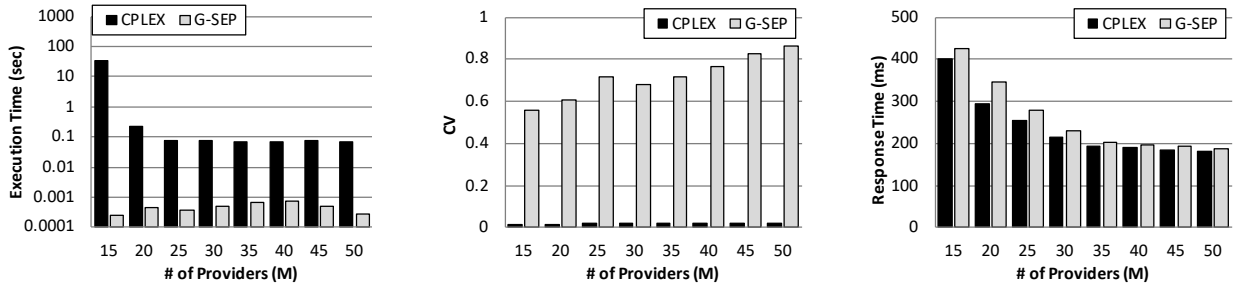


Figure 4. The effect of the number of providers (M) on (a) execution time of the placement algorithm, (b) Coefficient of Variance (CV) of fairness ratio, and (c) average workload response time.

for a high number of requests (see Figure 3(a)), achieves only a 17% better response time for requesters. Thus, G-SEP, other than being more suitable for a real-world scenario by having low execution times, achieves a near optimal allocation performance and response time for varying number of requesters.

Performance and scalability with respect to the number of providers. Finally, we analyze the effect of the number of providers on the performance of CPLEX and G-SEP. We consider a fixed number of requests ($N = 20$) and vary the number of providers from 15 to 50. We stop at 15 because, with fewer providers, there is no feasible solutions that satisfies the constraints of the SEP-MILP.

Figure 4(a) shows the execution time obtained by CPLEX and G-SEP. We observe that by increasing the number of providers, the execution time of CPLEX decreases while the execution time of G-SEP increases. The reason is that, by increasing the number of providers, there is a lower level of competition between requests in order to obtain the optimal assignment. Therefore, finding the optimal solution is simpler for CPLEX. On the other hand, the time complexity of G-SEP depends on the number of providers. Thus, by increasing the number of providers, the execution time of G-SEP increases. On the other hand, the execution time of CPLEX ($<$ 32 seconds) is significantly higher than that of G-SEP ($<$ 0.7 milliseconds), which allocates more than 85% of the requests.

Figure 4(b), shows the CV obtained by CPLEX and G-SEP. For both algorithms, the CV generally increases with the increase in the number of providers because some of the providers will have no assignment. Figure 4(c) shows the average response time obtained by each algorithm. By increasing the number of providers, the average response time decreases because each request has more allocation options. Although CPLEX achieves a better fairness, G-SEP shows only a 5.6% increase in response time compared to CPLEX. Thus, G-SEP, despite having a much lower execution time compared to CPLEX, achieves a near optimal allocation performance and response time for varying number of providers.

5 Conclusions and Future Work

In this paper, we addressed the problem of energy-efficient speculative execution in VEC systems. We proved that the problem is NP-hard and developed a greedy algorithm (G-SEP) to solve it. We evaluated the performance of G-SEP and compared it with the optimal solution. Our results showed that G-SEP achieves near optimal performance with a considerably lower execution time. As future work, we plan to develop an *energy manager* that can make decisions on the vehicles states (requester or provider) and guarantee energy savings for all the participating vehicles over multiple executions of energy manager and replica manager.

References

[1] Autotest. 2018. DSRC Technology. https://www.auto-talks.com/technology/dsrc-technology/. (2018).

[2] Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, Mayur Naik, and Ashwin Patti. 2011. CloneCloud: Elastic Execution Between Mobile Device and Cloud. In *EuroSys '11*. ACM, New York, NY, USA, 301–314.

[3] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. 2010. MAUI: Making Smartphones Last Longer with Code Offload. In *MobiSys '10*. ACM, New York, NY, USA, 49–62.

[4] IBM. 2009. IBM ILOG CPLEX V12.1 User's Manual. (2009). ftp://ftp.software.ibm.com/software/websphere/ilog/docs/

[5] Zhiyuan Jiang, Sheng Zhou, Xueying Guo, and Zhisheng Niu. 2018. Task Replication for Deadline-Constrained Vehicular Cloud Computing: Optimal Policy, Performance Analysis, and Implications on Road Traffic. *IEEE Internet of Things Journal* 5, 1 (2018), 93–107.

[6] Jan K Lenstra and AHG Rinnooy Kan. 1979. Computational complexity of discrete optimization problems. In *Annals of Discrete Mathematics*. Vol. 4. Elsevier, Banff, Alta, and Vancouver, B.C. Canada, 121–140.

[7] Lan Li, Xiaoyong Zhang, Kaiyang Liu, Fu Jiang, , and Jun Peng. 2018. An Energy-Aware Task Offloading Mechanism in Multiuser Mobile-Edge Cloud Computing. *Mobile Information Systems* 2018 (April 2018), 3–15.

[8] Tridib Mukherjee, Georgios Varsamopoulos, Sandeep KS Gupta, and Sanjay Rungta. 2007. Measurement-based power profiling of data center equipment. In *Cluster Computing*. IEEE, Austin, TX, USA, 476–477.

[9] O. Muñoz, A. Pascual-Iserte, and J. Vidal. 2015. Optimization of Radio and Computational Resources for Energy Efficiency in Latency-Constrained Application Offloading. *IEEE Trans. on Vehicular Technology* 64, 10 (Oct 2015), 4738–4755.

[10] Usman Pirzada. 2016. Nvidia Drive PX 2 Uses Integrated and Discrete Pascal GPU Cores âĂŞ 24 DL TOPS, 8 TFLOPs and Up To 4GB GDDR5. https://wccftech.com/nvidia-drive-px2-pascal-gtc-2016/. (2016).

[11] Usman Pirzada. 2017. Tesla Model 3 battery packs have capacities of 50 kWh and 75 kWh, says Elon Musk. https://electrek.co/2017/08/08/tesla-model-3-battery-packs-50-kwh-75-kwh-elon-musk/. (2017).

[12] S. Sardellitti, G. Scutari, and S. Barbarossa. 2015. Joint Optimization of Radio and Computational Resources for Multicell Mobile-Edge Computing. *IEEE Trans. on Signal and Information Processing over Networks* 1, 2 (June 2015), 89–103.

[13] Yuxuan Sun, Jinhui Song, Sheng Zhou, Xueying Guo, and Zhisheng Niu. 2018. Task Replication for Vehicular Edge Computing: A Combinatorial Multi-Armed Bandit based Approach. *arXiv preprint arXiv:1807.05718* abs/1807.05718 (2018), 7.

[14] H. Trinh, D. Chemodanov, S. Yao, Q. Lei, B. Zhang, F. Gao, P. Calyam, and K. Palaniappan. 2017. Energy-Aware Mobile Edge Computing for Low-Latency Visual Data Processing. In *FiCloud*. IEEE, Prague, Czech Republic, 128–133.

[15] H. Viswanathan, E. K. Lee, I. Rodero, and D. Pompili. 2015. Uncertainty-Aware Autonomic Resource Provisioning for Mobile Cloud Computing. *IEEE Trans. on Parallel and Distributed Systems* 26, 8 (Aug 2015), 2363–2372.

[16] Rong Yu, Yan Zhang, Stein Gjessing, Wenlong Xia, and Kun Yang. 2013. Toward cloud-based vehicular networks with efficient resource management. *IEEE Network* 27, 5 (2013), 48–55.

[17] K. Zhang, Y. Mao, S. Leng, Q. Zhao, L. Li, X. Peng, L. Pan, S. Maharjan, and Y. Zhang. 2016. Energy-Efficient Offloading for Mobile Edge Computing in 5G Heterogeneous Networks. *IEEE Access* 4 (2016), 5896–5907.

[18] Kan Zheng, Hanlin Meng, Periklis Chatzimisios, Lei Lei, and Xuemin Shen. 2015. An SMDP-based resource allocation in vehicular cloud computing systems. *IEEE Trans. on Industrial Electronics* 62, 12 (2015), 7920–7928.

[19] Chao Zhu, Giancarlo Pastor, Yu Xiao, Yong Li, and Antti Ylae-Jaeaeski. 2018. Fog Following Me: Latency and Quality Balanced Task Allocation in Vehicular Fog Computing. In *SECON*. IEEE, Hong Kong, 1–9.