Multiobjective Optimization-driven Task Scheduling in Vehicular Cloud Environments

Joahannes B. D. da Costa*, Allan M. de Souza†, Denis Rosário‡, Christoph Sommer§, Leandro Villas†
*Universidade de São Paulo (USP), Escola Politécnica (Poli), São Paulo, Brazil
†Universidade Estadual de Campinas (UNICAMP), Campinas, Brazil
‡Federal University of Pará (UFPA), Belém, Brazil
§TU Dresden, Faculty of Computer Science, Germany
Contact: joahannes@usp.br, denis@ufpa.br, {allanms, lvillas}@unicamp.br, https://cms-labs.org/people/sommer

***Abstract*—Vehicular Edge Computing (VEC) has emerged as a promising paradigm to address the growing demand for low-latency computation in vehicular applications, driven by the increasing number of connected vehicles and the massive volume of generated data. However, the highly dynamic nature of VEC environments poses significant challenges for efficient task scheduling. To meet these challenges, this work proposes MOMUS, a multiobjective optimization-based scheduler that leverages the Non-dominated Sorting Genetic Algorithm II (NSGA-II) algorithm to balance conflicting objectives, including maximizing the number of tasks completed within their deadlines, minimizing monetary cost, and reducing system latency. Simulation results show that MOMUS outperforms state-of-the-art VEC scheduling approaches, particularly under high-demand scenarios, achieving higher task completion rates while reducing monetary cost and maintaining acceptable latency.**

I. Introduction

The number of vehicles worldwide continues to grow, along with the increasing adoption of connected vehicles capable of communicating with other entities, such as smart traffic lights, surrounding vehicles, roadside infrastructure, and the Internet [1]. It is estimated that the number of connected vehicles will exceed 367 million by 2027 [2]. This growth brings challenges such as traffic congestion and a massive volume of data generated by sensors and onboard cameras in vehicles. Several safety and traffic efficiency applications, such as collision warnings, cooperative navigation, and entertainment for passengers, require fast processing and low latency [3]. The traditional model of sending all data to a remote cloud is not always suitable for these applications due to latency constraints and potential network overload [4].

In this context, Vehicular Edge Computing (VEC) emerges as an interesting approach to apply the edge computing paradigm in order to bring computing and storage resources closer to the data generation location, i.e., vehicles [5]. VEC creates a flexible infrastructure called Vehicular Cloud (VC), which leverages: *(i)* resources from the vehicles (such as processing units, memory, and storage) to dynamically form clusters; and *(ii)* edge servers deployed at fixed locations near roads, such as Roadside Units (RSUs) for vehicular communication or the cellular network Base Stations (BSs).

A VC consists of a set of vehicles and/or communication infrastructures, where their computational resources are aggregated and made available in the network for use by other vehicular users. Typically, this usage follows the traditional pay-as-you-go model in cloud computing infrastructures [6]. In other words, a user may need computational resources to process a task but does not have sufficient resources, and thus the VEC controller receives the task and decides where it will be processed. The user is only concerned with paying for the system usage time. VEC is responsible for deciding where the task will be processed (vehicles and/or BS) and for pricing the use of these resources.

In a dynamic and complex scenario such as VEC, one of the central questions is the task scheduling problem: *how to efficiently decide which computational task should be processed where and when?* In this context, three aspects are fundamental to achieve effective scheduling in a highly dynamic vehicular environment, namely: i) *Vehicular Mobility*: the scheduling mechanism must account for mobility to avoid task failures due to disconnections, as vehicle movement continuously changes network topology and resource availability; ii) *Deadline Constraints*: tasks completed after their deadlines become useless, since many vehicular applications, particularly safety-related ones, require results to be delivered within strict time limits; iii) *Monetary Cost*: the processing cost is an important factor for end users. Therefore, designing scheduling strategies that balance deadline satisfaction and cost efficiency while serving as many requests as possible remains a key challenge.

In this paper, we introduce a multiobjective optimization-driven task scheduling in vehicular cloud environments, called **MOMUS**. MOMUS formulates task scheduling as a multiobjective optimization problem to improve operational efficiency. The adopted optimization generates a set of solutions that satisfy the established criteria, organized in a Pareto Frontier. Among the available options, one solution is selected and applied to the system. For instance, MOMUS seeks to jointly minimize the monetary cost associated with the use of computational resources and the task processing latency. In this sense, MOMUS reduces the need for resource rescheduling by simultaneously considering a set of solutions that satisfy the objective functions, reducing the total cost. The obtained results indicate that MOMUS is capable of scheduling a larger number of tasks, in addition to minimizing resource utilization costs and reducing system latency, when compared to other approaches in the literature.

© 2026 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

104th IEEE Vehicular Technology Conference (VTC2026-Fall)

This paper is organized as follows. Section II discusses the related works. Section III presents the system model, problem definition, and MOMUS's operation in detail. Section IV discusses the performance evaluation and results obtained. Finally, Section V presents the conclusion and future works.

II. Related Work

Several works have addressed task scheduling in VEC using multi-criteria decision-making and optimization. Su and Liu [7] proposed a task scheduling scheme for VEC scenarios under a cloud-edge collaborative architecture. The authors designed a task priority model based on Analytic Hierarchy Process (AHP), which considers task latency tolerance, computational demand, and storage demand, and employed a Deep Reinforcement Learning (DRL) approach to optimize the scheduling rate. However, relying on DRL introduces computational overhead, which may hinder real-time deployment in resource-constrained scenarios. Da Costa *et al.* [8] introduced a context-aware scheduler called FARID, which applies probabilistic VC selection to improve performance and fairness in vehicular resource usage. FARID adopts a dual-criteria strategy to obtain the Pareto set by minimizing processing time and deadline violations, reducing task waiting time and prioritizing tasks with restricted deadlines. FARID also maximizes the number of scheduled tasks through an approximation heuristic for the Bin Covering Problem (BCP) problem.

Bio-inspired optimization has also been explored. Surayya *et al.* [9] formulated a multi-objective problem to minimize delay and cost, solving it with a Particle Swarm Optimization (PSO)-based algorithm. Although it produces a Pareto frontier, its convergence time may be unsuitable for real-time decisions. Moreover, parameter refinement is often required to improve solution accuracy, limiting applicability in highly dynamic environments. Similarly, Lieira *et al.* [5] introduced LOPRIVE, a priority-oriented scheduler based on the Lion Optimization Algorithm (LOA). By iteratively evaluating task "strength" against available resources, it improves scheduling efficiency while prioritizing critical tasks. However, it relies on static category weights and does not account for variations in user profiles or heterogeneous resource-sharing incentives.

Due to the highly dynamic nature of VEC, machine learning approaches have gained attention. Yang *et al.* [10] integrated computational demand prediction with scheduling decisions, enabling flexible service pre-caching at RSUs. Task execution is handled via an online learning method based on a Multi-Armed Bandit (MAB), allowing vehicles to learn edge-node performance and schedule multiple tasks without full network topology knowledge, aiming to minimize total delay. However, MAB-based exploration may degrade performance during the learning phase, especially in highly dynamic environments.

Overall, most existing solutions address scheduling only partially, often neglecting the joint impact of network dynamics, resource heterogeneity, diverse computational demands, and user profiles. Furthermore, assumptions such as static parameters, greedy decisions, or high convergence times limit their applicability in highly dynamic VEC scenarios. To the best of our knowledge, only MOMUS considers a multi-objective optimization to solve such a problem in an integrated scheduler.

III. System Overview

This section is organized into three parts: a system overview, the formal problem definition, and the operation of MOMUS.

A. Network and System Model

Figure 1 shows the system architecture composed of vehicles, BSs, and VEC controllers. VEC controllers are responsible for coordinating $|R|$ predefined city regions, managing both VC formation and task scheduling. VC formation consists of aggregating idle resources from vehicles and infrastructure, whereas scheduling concerns the use of such resources to process tasks. Since MOMUS focuses exclusively on task scheduling, we simplify the VC formation process, without loss of generality, to the association between vehicles and BSs during vehicular mobility. While a vehicle remains within the coverage area of a BS, the VEC system can exploit its idle resources [8]. The exchange of information between BSs is enabled by an *Xn* interface.

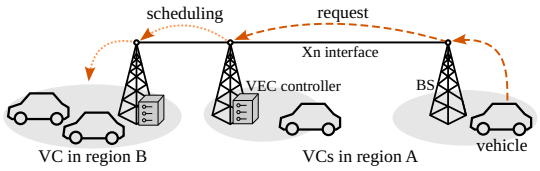

Fig. 1. The architecture employed by MOMUS [8].

The VEC scenario considers a set of x vehicles, denoted by $u_i \in U = \{u_1, u_2, \ldots, u_x\}$, and a set of p BSs deployed in the city, denoted by $b_y \in B = \{b_1, b_2, \ldots, b_p\}$. Each region $r \in R$ contains at least one BS and exactly one VEC controller. A set of vehicular clouds $v_j \in V = \{v_1, v_2, \ldots, v_m\}$ is available for processing, where each vehicular cloud $v_j \in V$ satisfies $v_j \subseteq U \cup B$. Each VC v_j has a total Central Processing Unit (CPU) capacity Ω_j, defined as the sum of the processing capacities of its members, including both vehicles (ω_{u_i}) and BSs (ω_{b_y}), measured in Million of Instructions Per Second (MIPS). In addition, each VC has a total storage capacity Φ_j, measured in Megabyte (MB).

B. Problem Definition

The scheduling problem addressed can be formalized as follows. Consider a set of tasks $T = \{t_1, t_2, \ldots, t_n\}$, where each task $t_l \in T$ is represented by the tuple $\langle id_l, s_l, w_l, D_l \rangle$. In this tuple, id_l is a unique identifier, s_l denotes the input data size in MB, w_l represents the number of instructions required for processing in Million of Instructions (MI), and D_l denotes the maximum completion deadline in seconds.

In this cooperative scenario, the resources of a VC are shared among all tasks assigned to it. Accordingly, the effective processing capacity available to each task in VC v_j is given by $\Psi_j = \frac{\Omega_j}{|T'_j|}, j \in V$, where Ω_j denotes the total processing capacity of v_j and $|T'_j|$ is the number of tasks assigned to it.

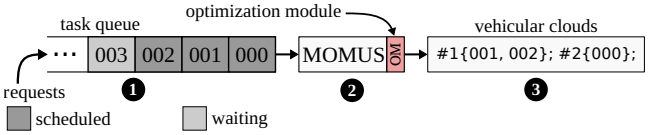

Fig. 2. System task processing pipeline.

Also, the processing time d_{lj} of task t_l is defined as $d_{lj} = \frac{w_l}{\Psi_j}, \forall\, t_l \in v_j$. A task is considered successfully served if its processing time does not exceed its deadline, i.e., $d_{lj} \leq D_l$.

In addition, executing a task incurs a monetary cost associated with the consumed CPU cycles and the processing time. This cost, denoted by C_{t_l}, is modeled as the product of the processing time and the computational workload, multiplied by a predefined unit price[1]. The unit price depends on whether the task is executed on communication infrastructure resources (BS b_y) or vehicular resources (u_i), as defined in Equation (1):

$$C_{t_l} = \begin{cases} d_{lj} \times (w_l \times 10.34592), & \text{if } t_l \text{ uses } b_y, \\ d_{lj} \times (w_l \times 5.17296), & \text{if } t_l \text{ uses } u_i. \end{cases} \quad (1)$$

In summary, the scheduler aims to select a subset of tasks $T' \subseteq T$ to be processed by the available VCs, maximizing the number of tasks completed within their deadlines while simultaneously minimizing the total monetary cost of resource usage. This is inherently a multiobjective optimization problem. In this way, the main constraints considered are: (1) each task deadline must be satisfied; (2) the sum of the sizes of tasks allocated to a VC cannot exceed its storage capacity; and (3) the total required CPU cycles cannot exceed the processing capacity of the VC.

C. MOMUS's operation

Figure 2 illustrates the task scheduling process of MOMUS, which consists of three main stages. First, multiple requests arrive at the system, corresponding to user tasks whose vehicles lack sufficient computational resources for local execution. Tasks are queued in the system (Stage ❶) and may assume two states: initially *waiting* (*i.e.*, the task is still awaiting execution) or *scheduled* (*i.e.*, the task has been assigned to a VC). The task only moves from the initial *waiting* state when it is selected by a scheduling mechanism (Stage ❷) and dispatched for processing. Finally, tasks are executed on the available VCs (Stage ❸). The entire process is monitored by VEC controllers, and a task is removed from the queue only when its processing is completed or its deadline D_l expires.

MOMUS evaluates the tasks in the queue and makes scheduling decisions based on a joint analysis of multiple criteria, such as computational resource requirements and *deadline*. The core idea is to balance the priority of tasks with shorter deadlines, *i.e.*, those that cannot tolerate long waiting times, with the amount of computational resources required for their execution. As discussed previously, this selection process involves multiple objectives, and thus MOMUS must consider multiobjective optimization algorithms. To this end, MOMUS includes an Optimization Module (OM).

Three well-known multiobjective optimization algorithms were evaluated to select the most suitable one for this specific context, namely: Particle Swarm Optimization (PSO), Non-dominated Sorting Genetic Algorithm II (NSGA-II), and NSGA-III [11]. After a preliminary performance analysis (see in Section IV-B1), we selected NSGA-II as the decision-making algorithm for MOMUS. Specifically, NSGA-II is an evolutionary algorithm widely used to solve multiobjective optimization problems, especially in the presence of conflicting objectives, such as minimizing monetary cost and maximizing the number of tasks completed within their deadlines. Its operation is based on the evolution of a population of candidate solutions over multiple generations. At each generation, solutions are evaluated with respect to all objectives and organized using a non-dominated sorting process, which identifies the solutions that belong to the Pareto frontier, that is Pareto-optimal solutions.

In this case, *a candidate solution represents a complete schedule, i.e., an assignment of tasks from set T to VCs, respecting deadline and capacity constraints*. Also, the algorithm employs the crowding-distance metric to preserve diversity among solutions, preventing the search from concentrating on a single region of the solution space. Genetic operators, such as crossover and mutation, are applied to generate new solutions, enabling the exploration of different task allocation possibilities until a set of efficient schedules is obtained.

To integrate MOMUS into the VEC environment, this work presents a system operation cycle (Algorithm 1) to manage task execution. The algorithm operates in discrete time intervals $k \in K$, during which new tasks are added to the waiting queue (lines 1 and 2). When pending tasks exist, MOMUS is triggered to schedule them, generating the set $\mathcal{S}$ based on the available VCs in V (lines 3 and 4). The set $\mathcal{S}$ contains all the tasks that have been scheduled. During execution, completed tasks have their cost computed according to Equation (1), are removed from the queue, and their results are returned to the user (lines 6, 7, 8, and 9); otherwise, their progress estimate is updated (line 11). This workflow highlights the scheduler as a key component for dynamic task management in VEC.

Algorithm 1: VEC operation with MOMUS integrated.

```
1  foreach time interval k ∈ K do
2      Add new tasks to the waiting queue
3      if the task queue is not empty then
4          S ← MOMUS(T, V)          ▷ Scheduling mechanism
5          foreach s ∈ S do
6              if s has completed execution then
7                  Compute cost using Equation (1)
8                  Remove task from the waiting queue
9                  Return result to the user
10             else
11                 Update progress estimate of s
```

[1]Price values are based on commercial Amazon EC2 instances in the *us-east-2* region (*c8a.48xlarge* and *c8a.metal-24xl*), used as references for the costs of b_y and u_i, respectively.

Algorithm 2 describes how MOMUS adapts the NSGA-II algorithm for task scheduling in the VEC environment. The algorithm takes as input the set of tasks T and the set of available VCs V. Initially, task deadlines D_l are extracted (line 1), and estimated processing times PT are computed for each task–VC combination (line 2). A population P of `POP` feasible solutions is then initialized, with feasibility ensured by a repair operator that enforces CPU and storage constraints (line 3). An evolutionary process is executed over `GENS` generations (line 4). At each generation, a child population P_{child} is constructed (line 5) by selecting parents via binary tournament selection (line 7). Selected parents undergo crossover with probability `CXPROB` (line 8) and mutation with probability `MUTPROB` (line 9), generating new solutions. These solutions are repaired to maintain feasibility (line 10) and added to the child population (line 11). Parent and child populations are then combined (line 12), ranked using non-dominated sorting (line 13), and evaluated using the crowding-distance metric (line 14). The best `POP` individuals are selected for the next generation (line 15). After the final generation, the first Pareto frontier F_1 is identified (line 16), and the solution s^* with the lowest total monetary cost is selected (line 17). The resulting set of scheduled tasks T' is obtained (line 18) and returned as the output of MOMUS (line 19). The evolutionary parameters were set to `POP` = 30, `GENS` = 10, `CXPROB` = 0.9, and `MUTPROB` = 0.15.

Algorithm 2: Pseudocode of MOMUS.

```
   Input: set of tasks T and set of VCs V
   Output: subset of scheduled tasks T'
1  Extract deadlines D ← {D_l | t_l ∈ T}
2  Compute PT ← {time(t_l, v_j) | t_l ∈ T, v_j ∈ V}
3  Initialize population P with POP feasible solutions
4  for gen = 1 to GENS do
5      P_child ← ∅
6      while |P_child| < POP do
7          Select parents p_1, p_2 from P via binary tournament
8          Apply crossover to p_1, p_2 with probability CXPROB
9          Apply mutation to offspring with probability MUTPROB
10         Repair offspring to ensure feasibility
11         P_child ← P_child ∪ {offspring}
12     P ← P ∪ P_child
13     Classify P into Pareto frontiers (non-dominance)
14     Compute crowding distance for each solution
15     Select the best POP individuals for the next generation
16 Identify the first Pareto frontier F_1
17 Select s* ∈ F_1 with the lowest total monetary cost
18 T' ← tasks allocated according to s*
19 return T'
```

IV. Performance Evaluation

This section describes the methodology and metrics used to evaluate the efficiency of MOMUS.

A. Scenario description and Methodology

The experiments were carried out using Simulation of Urban Mobility (SUMO) 1.18.0, and the scheduling algorithms were implemented in Python 3.10. The algorithms interact with SUMO through the TraCI interface, enabling a real-time cycle of data collection and decision-making. To increase simulation granularity, the simulation time step (`--step-length`) was set to 0.1 s. All simulations were executed on a 12th Gen Intel i7-12700H (3.50-4.70 GHz) running Ubuntu 22.04 LTS.

The scenario is based on a subset of the TAPASCologne mobility trace[2], covering a total area of approximately 114 km^2. During the simulation, which lasts 700 seconds (with the first 100 seconds allocated to the warm-up phase), the number of active vehicles varies over time, reaching a peak of 350 vehicles. This dynamic creates a challenging scenario for resource management in the vehicular environment, as resource demand is consistently higher than the available supply.

The scenario includes 16 deployed communication infrastructures, corresponding to 5G network BSs. Each infrastructure provides a communication range of 2000 meters and a processing capacity of 15 MIPS. The city comprises four regions, each managed by a VEC controller. Vehicles, with a communication range of 250 meters, voluntarily contribute 1 MIPS of processing capacity and 1 MB of storage each, forming dynamic VCs. These MIPS assignments for BSs and vehicles are intended solely to differentiate these elements, without loss of generality.

The workload generated by vehicles consists of Bag-of-Tasks (BoT) tasks, which arrive at the system according to a Poisson process [6]. To assess the robustness of the algorithms, the arrival rate of tasks λ is 15 tasks/second. Each task is characterized by an input size s_l ranging from 1 to 10 MB, a computational demand w_l ranging from 1 to 30 MI, and a maximum deadline D_l. Four deadline levels were considered, namely $D_l = 0.5, 1, 5,$ and 7 seconds, enabling the analysis of performance from latency-sensitive applications to more delay-tolerant ones.

We compared MOMUS with state-of-the-art VC task scheduling mechanisms, namely: *First-Come, First-Served (FCFS)*, a baseline approach that schedules tasks strictly in their order of arrival; *MAB*, which adopts a reinforcement learning-based strategy to balance exploration of new resources and exploitation of previously successful scheduling decisions, as proposed in [10]; *LOPRIVE*, a priority-oriented bio-inspired approach based on the Lion Optimization Algorithm (LOA) that considers task requirements and VC capacities [5]; and *FARID*, which combines an approximation heuristic for the BCP problem with Pareto optimization to jointly minimize monetary cost and processing time [8].

Performance evaluation was conducted using four main metrics to provide a comprehensive assessment of the efficiency and practicality of each solution: i) *Scheduled Tasks (%)*: percentage of tasks completed within their deadlines; ii) *Monetary Cost ($)*: aggregated cost of processing resources used by vehicles and BSs; iii) *System Latency (s)*: end-to-end latency, including queue waiting and execution times, reflecting perceived Quality of Service (QoS); iv) *CPU Time*: The CPU time consumed by the scheduling algorithm on the machine running the simulation.

[2]https://sumo.dlr.de/docs/Data/Scenarios/TAPASCologne.html

B. Simulation Results

This section presents the simulation results in two parts: the first discusses the selection of the multiobjective optimization algorithm adopted in MOMUS, and the second compares MOMUS with other task scheduling approaches.

1) Optimization: Table I summarizes the obtained results of NSGA-II, NSGA-III, and PSO as multiobjective optimization algorithm to be integrated into MOMUS, using the metrics and parameters defined in the previous subsection. By analyzing the percentage of scheduled tasks, we observe that NSGA-II and NSGA-III consistently outperform PSO across all evaluated deadlines. Under the most restrictive settings (*i.e.*, $D_l = 0.5$ s), the scheduling rate decreases for all methods. However, both NSGA-based algorithms remain substantially more effective, indicating greater robustness under stringent timing constraints. As deadlines become more relaxed, NSGA-II and NSGA-III achieve near-complete scheduling rates (above 95%), whereas PSO remains significantly less effective.

In terms of monetary cost, NSGA-III yields the lowest values across all cases, followed closely by NSGA-II. PSO incurs considerably higher costs, with an increasing trend as deadlines increase. System latency results also favor NSGA-based methods, which maintain low and stable delays for all deadline configurations. In contrast, PSO exhibits a pronounced increase in latency as deadlines are relaxed.

Finally, CPU time highlights the trade-off between computational overhead and solution quality. Although PSO requires the lowest CPU time, it achieves poor scheduling performance, higher costs, and increased latency. NSGA-III presents the highest computational overhead, whereas NSGA-II provides a more balanced profile, achieving competitive performance with substantially lower CPU time than NSGA-III.

TABLE I
PERFORMANCE OF THE MULTIOBJECTIVE OPTIMIZATION ALGORITHMS.

Algorithm	$D_l = 0.5$	$D_l = 1$	$D_l = 5$	$D_l = 7$
Scheduled tasks (%)				
NSGA-II	24.6 ± 0.36	64.2 ± 1.22	96.1 ± 0.27	98.5 ± 0.26
NSGA-III	26.2 ± 0.36	66.8 ± 1.60	95.7 ± 0.27	98.2 ± 0.17
PSO	2.7 ± 0.04	7.3 ± 0.50	18.0 ± 0.54	24.9 ± 0.61
Monetary cost ($)				
NSGA-II	16.9 ± 0.06	16.6 ± 0.18	16.0 ± 0.28	16.0 ± 0.29
NSGA-III	15.7 ± 0.31	15.5 ± 0.14	14.8 ± 0.29	14.6 ± 0.32
PSO	62.2 ± 2.01	66.1 ± 3.24	78.7 ± 1.47	83.3 ± 1.37
System latency (s)				
NSGA-II	0.4 ± 0.00	0.9 ± 0.01	1.4 ± 0.01	1.4 ± 0.02
NSGA-III	0.4 ± 0.00	0.8 ± 0.01	1.3 ± 0.02	1.4 ± 0.02
PSO	0.4 ± 0.00	1.0 ± 0.00	3.7 ± 0.01	5.4 ± 0.02
CPU time (s)				
NSGA-II	73.8 ± 1.61	61.8 ± 2.40	62.4 ± 0.52	64.1 ± 2.10
NSGA-III	139.3 ± 5.10	120.4 ± 6.20	123.1 ± 10.65	121.8 ± 10.23
PSO	31.2 ± 1.06	20.1 ± 1.40	47.3 ± 5.54	54.4 ± 6.61

The cells with the best and worst values are highlighted in green and red, respectively. In the event of a tie, both are marked the same color.

In summary, NSGA-II offers the best compromise between computational feasibility and solution quality under a workload considered, supporting its selection as the multiobjective optimization core of MOMUS.

2) Task scheduling: We compared the performance of MOMUS considering NSGA-II as a multiobjective optimization algorithm against other scheduling approaches using the same metrics and scenario described previously. Figure 3 summarizes the results in terms of scheduled tasks, monetary cost, system latency, and CPU time.

Figure 3(a) presents the percentage of scheduled tasks under a fixed arrival rate of $\lambda = 15$. MOMUS clearly achieves the highest scheduling rates for all deadline configurations, reaching close to 100% when deadlines are relaxed ($D_l \geq 5$ s). Even under the most restrictive condition ($D_l = 0.5$ s), MOMUS remains substantially superior to the baselines, indicating robustness under severe time constraints. In contrast, FCFS and LOPRIVE schedule only a limited fraction of tasks, showing that static or priority-based heuristics are insufficient when resources are highly contested. MAB and FARID improve performance compared to these baselines, but they still fail to match the consistency of MOMUS, suggesting that their decision processes are less effective in exploiting the available VEC resources as deadlines become more restricted.

Figure 3(b) reports system latency. MOMUS consistently yields the lowest latency values and remains stable even as deadlines increase, which indicates that its scheduling decisions avoid overloading specific VCs and reduce the accumulation of tasks waiting in the queue. FARID presents the second-best latency performance for most deadlines, suggesting that its heuristic also promotes balanced allocations. However, its latency becomes higher than MOMUS as deadlines relax, possibly because it prioritizes low-cost resources that may be computationally weaker or more congested. FCFS, LOPRIVE, and MAB exhibit a pronounced latency increase for larger deadlines, which suggests that these methods allow queue accumulation instead of proactively distributing tasks across available resources. This behavior is particularly critical in vehicular applications, since increased latency directly impacts QoS and may lead to service degradation even when deadlines are not immediately violated.

Monetary cost results are shown in Figure 3(c). MOMUS achieves the lowest cost values, followed closely by FARID, while FCFS and LOPRIVE incur the highest costs across all deadlines. This behavior is expected since FARID explicitly targets cost reduction and employs a heuristic to approximate low-cost allocations. However, FARID does not translate this advantage into a proportional increase in the number of scheduled tasks, indicating that its low-cost decisions may sacrifice feasibility or task admission. In contrast, MOMUS maintains a competitive cost profile while achieving significantly higher service rates, which indicates that it is able to jointly optimize cost and feasibility. Moreover, MAB shows moderate cost values but with less stability, likely due to its exploration phase, which may temporarily select inefficient resources before converging to better policies.

Finally, Figure 3(d) presents the CPU time consumed by each scheduling mechanism. As expected, MOMUS incurs the highest computational overhead due to its NSGA-II evolutionary search, which requires iterative evaluation of multiple candidate schedules. FCFS and LOPRIVE remain computationally lightweight, while MAB and FARID show

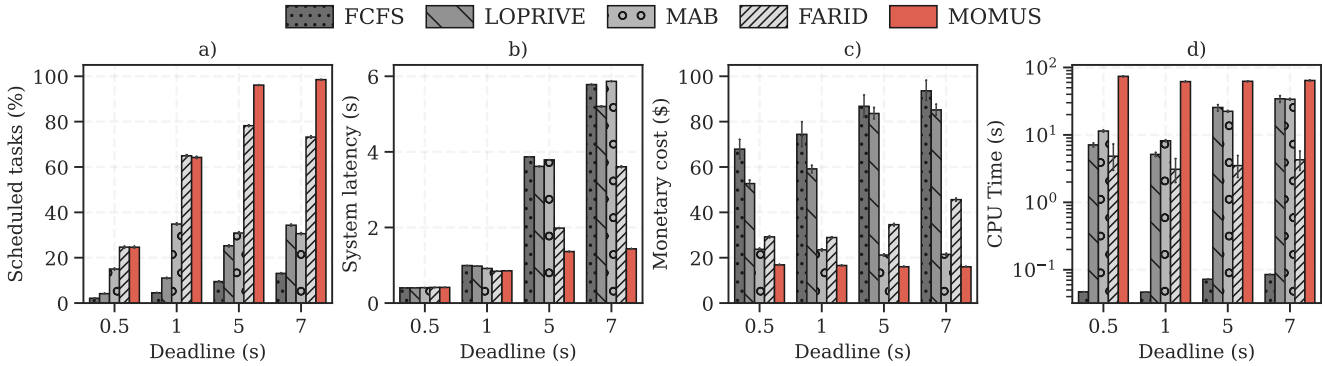

Fig. 3. Results with a task arrival rate $\lambda = 15$.

intermediate overhead. Although FARID achieves low CPU time, its reduced exploration capability may explain its weaker scheduling performance. Therefore, the results highlight a clear trade-off: MOMUS requires higher computation, but this cost enables substantially better decisions, reflected in higher task completion rates and lower latency.

In summary, the results demonstrate that MOMUS provides the best overall balance between service efficiency and QoS among the evaluated methods. While FARID achieves competitive monetary cost and reduced CPU time, it fails to deliver the same level of task scheduling efficiency and latency minimization. In contrast, MOMUS consistently maintains near-optimal (considering that the solutions used lie on the Pareto frontier) task servicing while keeping monetary cost competitive, indicating that its multiobjective optimization process effectively balances conflicting requirements. These findings validate the use of NSGA-II as the decision core of MOMUS for task scheduling in dynamic VEC environments.

V. Conclusion

This paper proposed MOMUS, a multiobjective optimization-driven scheduler that employs the NSGA-II algorithm to balance conflicting objectives, including maximizing the number of tasks completed within their deadlines and minimizing monetary cost. We implemented and evaluated MOMUS in a realistic simulation environment based on a vehicular mobility trace under different workload conditions. The results show that MOMUS consistently outperforms state-of-the-art scheduling approaches, particularly under high load and restrictive deadlines. The proposed mechanism achieves high task completion rates while simultaneously reducing monetary cost and system latency. Overall, the adoption of Non-dominated Sorting Genetic Algorithm II (NSGA-II) as the optimization core proves effective, providing a favorable trade-off between scheduling performance and computational feasibility.

Future work includes investigating adaptive mechanisms to dynamically tune the parameters of NSGA-II, exploring a greater diversity of workloads, and incorporating additional relevant metrics, such as load balancing and fairness indexes, to provide a more comprehensive evaluation. Although mobility is a crucial factor, MOMUS does not use this information for decision-making. Therefore, mobility information could be leveraged to further enhance its robustness.

Acknowledgments

São Paulo Research Foundation (FAPESP), grants #2018/16703-4, #2021/13780-0, and #2024/21006-1. The brazilian Ministry of Science, Technology and Innovations, with resources from Law nº 8,248, of October 23, 1991, within the scope of PPI-SOFTEX, coordinated by Softex and published Arquitetura Cognitiva (Phase 3), DOU 01245.003479/2024-10.

References

[1] D. Baumann, M. Sommer, F. Dettinger, T. Rösch, M. Weyrich, and E. Sax, "Connected Vehicle: Ontology, Taxonomy and Use Cases," in *2024 IEEE International Systems Conference*, IEEE, 2024, pp. 1–6.

[2] A. Weissberger, *Juniper Research: 5G connectivity opportunity for the connected car market*, IEEE ComSoc Technology Blog, Jan. 2023. [Online]. Available: https://techblog.comsoc.org/2023/01/09/juniper-research-5g-connectivity-opportunity-for-the-connected-car-market/.

[3] X. Zhang *et al.*, "Vehicle-to-Everything Communication in Intelligent Connected Vehicles: A Survey and Taxonomy," *Automotive Innovation*, pp. 1–33, 2025.

[4] L. Bai, J. Cao, M. Zhang, and B. Li, "Collaborative Edge Intelligence for Autonomous Vehicles: Opportunities and Challenges," *IEEE Network*, vol. 39, no. 2, pp. 52–60, 2025.

[5] D. D. Lieira, M. S. Quessada, R. E. De Grande, and R. I. Meneguette, "LOPRIVE: LOA-Based Priority-Driven Task Allocation in the Vehicular Edge," in *IEEE Symposium on Computers and Communications (ISCC)*, IEEE, 2025, pp. 1–6.

[6] J. B. da Costa, A. M. de Souza, R. I. Meneguette, E. Cerqueira, D. Rosário, C. Sommer, and L. Villas, "Mobility and Deadline-Aware Task Scheduling Mechanism for Vehicular Edge Computing," *IEEE Transactions on Intelligent Transportation Systems*, vol. 24, no. 10, pp. 11 345–11 359, 2023.

[7] J. Su and Y. Liu, "Task offloading decision making for IoV based on deep reinforcement learning," *Scientific Reports*, vol. 15, p. 38 586, 2025.

[8] J. B. da Costa, A. M. de Souza, W. Lobato, D. Rosário, C. Sommer, and L. Villas, "Improving Fairness and Performance in Resource Usage for Vehicular Edge Computing," in *IEEE 98th Vehicular Technology Conference (VTC-Fall)*, IEEE, 2023, pp. 1–6.

[9] A. Surayya, M. M. Hussain, V. D. Reddy, A. Abdul, and F. Gazi, "Evolutionary Algorithms for Edge Server Placement in Vehicular Edge Computing," *IEEE Access*, vol. 13, pp. 79 030–79 052, 2025.

[10] J. Yang, K. Yang, X. Dai, Z. Xiao, H. Jiang, F. Zeng, and B. Li, "Service-Aware Computation Offloading for Parallel Tasks in VEC Networks," *IEEE Internet of Things Journal*, vol. 12, no. 3, pp. 2979–2993, 2024.

[11] K. Deb and H. Jain, "An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part I: Solving Problems With Box Constraints," *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 4, pp. 577–601, 2013.