Escalonamento de Tarefas Ciente de Contexto para Computação de Borda Veicular

Joahannes B. D. da Costa[1], Allan M. de Souza[1], Rodolfo I. Meneguette[2], Eduardo Cerqueira[3], Denis Rosário[3], Leandro A. Villas[1]

[1]Instituto de Computação (UNICAMP), Campinas – São Paulo – Brasil
[2]Universidade de São Paulo (USP), São Carlos – São Paulo – Brasil
[3]Universidade Federal do Pará (UFPA), Belém – Pará – Brasil

{joahannes, allanms}@lrc.ic.unicamp.br, meneguette@icmc.usp.br
{cerqueira, denis}@ufpa.br, leandro@ic.unicamp.br

Abstract. *Vehicular Edge Computing is a promising paradigm that provides cloud computing services closer to vehicular users. Vehicles and communication infrastructures can cooperatively attend vehicular services with low latency constraints through the vehicular clouds formation and use of these clouds' computational resources, where this last process is called task scheduling. An efficient task scheduler needs to decide which cloud will run the tasks, considering vehicular mobility and tasks' requirements. This is important to minimize processing time and the monetary cost of using computing power. However, the literature solutions do not consider these contextual aspects together, degrading the overall system efficiency. In this way, this work presents CARONTE, a task scheduler mechanism that considers contextual aspects in its decision process. The results have shown that CARONTE is able to schedule more tasks while minimizing monetary cost, computation delay, and queuing time when compared to state-of-the-art solutions.*

Resumo. *A Computação de Borda Veicular é um paradigma promissor que fornece serviços de computação em nuvem próximo dos usuários veiculares. Veículos e infraestruturas de comunicação podem cooperativamente atender serviços veiculares com restrições de baixa latência através da formação de nuvens veiculares e utilização dos recursos computacionais dessas nuvens, sendo este último processo chamado de escalonamento de tarefas. Um escalonador de tarefas eficiente precisa decidir qual nuvem veicular executará as tarefas, levando em consideração aspectos como mobilidade veicular e requisitos das tarefas. Isso se faz importante para minimizar o tempo de processamento e o custo monetário na utilização do poder computacional. No entanto, as soluções da literatura não consideram esses aspectos contextuais conjuntamente, o que pode degradar a eficiência geral do sistema. Assim, apresenta-se o CARONTE, um escalonador de tarefas que considera aspectos contextuais em seu processo de decisão. Os resultados mostraram que o CARONTE é capaz de escalonar mais tarefas enquanto minimiza custo monetário, atraso de computação e tempo de fila, quando comparado com abordagens do estado da arte.*

1. Introdução

Os veículos estão se tornando cada vez mais inteligentes e conectados, direcionando-nos para um sistema de transporte seguro, ecológico e eficiente [Guidoni et al. 2020, de Souza et al. 2020]. A demanda por serviços baseados em veículos vai crescer cada vez

mais e aumentar significativamente o consumo de largura de banda, ocasionando congestionamentos na rede e falhas na entrega dos dados [Meneguette et al. 2021]. Nesse contexto, é essencial garantir baixos níveis de latência para atender demandas de aplicações que possuem requisitos de tempo real, tais como, mecanismos baseados em aprendizagem de máquina ou serviços orientados à Inteligência Artificial para veículos autônomos conectados. No entanto, atender essas restrições não é uma tarefa simples por conta da dinamicidade da rede [Luo et al. 2021].

Levando isso em consideração e tomando proveito da capacidade de comunicação presente nos veículos provida pelas Redes Veiculares (*Vehicular Networks - VANETs*), além do poder computacional que também se faz presente nos modernos veículos conectados, surge o paradigma Computação de Borda Veicular (*Vehicular Edge Computing - VEC*). Tal paradigma visa utilizar e disponibilizar recursos computacionais disponíveis nos veículos e infraestruturas como serviços de nuvem, trazendo assim poder computacional para próximo dos usuários veiculares [Boukerche and Soto 2020]. Em uma arquitetura de VEC, existe um controlador de rede para cada região pré-definida da cidade, chamado controlador VEC. Tal controlador constrói e mantém conhecimento sobre os veículos por meio da comunicação entre veículos e Estações Base (*Base Stations - BSs*), permitindo regras de controle mais dinâmicas e precisas [Meneguette et al. 2021]. Desta forma, o controlador VEC forma Nuvens Veiculares (*Vehicular Clouds - VCs*) com recursos computacionais disponíveis nos veículos e infraestruturas.

O controlador VEC deve realizar a Formação de VCs e o Escalonamento de Tarefas [Luo et al. 2021]. O primeiro processo diz respeito ao agrupamento e gerenciamento dos recursos computacionais de veículos e infraestruturas, especificamente recursos de processamento e armazenamento. O segundo processo, que será o foco deste trabalho, se refere ao uso desses recursos computacionais mantidos pela VEC. Além disso, esse uso de recursos funciona tal qual o empregado por provedores de computação em nuvem, havendo um custo monetário associado a ele. Em suma, o escalonamento de tarefas possibilita a utilização de recursos de forma transparente por aplicações/serviços que exigem poder computacional acima do suportado localmente [Boukerche and Soto 2020].

Devido aos conhecidos problemas das VANETs (*i.e.*, alta mobilidade veicular, conexões intermitentes e desconexões), realizar o escalonamento de tarefas em VEC têm alguns desafios. Assim, pelo menos dois pontos devem ser levados em consideração para um escalonamento eficiente em VEC: (1) *Mobilidade veicular*: esse fator pode causar desconexões inesperadas entre os veículos que processam tarefas e o controlador VEC, ocasionando interrupções e atrasos no escalonamento. Com isso, informações de mobilidade dos veículos podem ser utilizadas na seleção de VCs mais estáveis para o processo de escalonamento. A estabilidade de uma VC refere-se à sua variabilidade de recursos, causada pela mudança no número de veículos na VC. (2) *Restrições de prazo (deadline)*: após o processamento da tarefa em uma VC, o resultado obtido deve ser devolvido antes de um *deadline* especifico e curto. Caso contrário, esse resultado torna-se inútil [Wang et al. 2020]. Portanto, o escalonamento devem considerar as restrições de prazo das tarefas em seu processo de decisão para aumentar suas taxas de atendimento.

Nesse contexto, considerar aspectos de mobilidade e restrições de prazo das tarefas em conjunto é um ponto em aberto na literatura. Ou seja, observar aspectos da mobilidade veicular, já que parte considerável do poder computacional da VEC é provida por veículos, é fundamental para estimar a disponibilidade futura de recursos das VCs. Na mesma direção, os *deadlines* das tarefas podem ser melhor observados e considerados a partir de uma estimativa da duração da disponibilidade dos recursos nas VCs, o que

possibilita inferir o tempo de processamento e a probabilidade de atendimento.

Considerando os desafios mencionados, este artigo apresenta um mecanismo para es**C**alon**A**mento de ta**R**efas ciente de c**ON**texto para compu**T**ação de borda v**E**icular, chamado **CARONTE**. O mecanismo é executado em controladores VEC dispostos na cidade e seleciona as tarefas para escalonamento com base em uma heurística de aproximação. O CARONTE considera o contexto referente à mobilidade dos veículos que fornecem recursos para a VEC e requisitos das tarefas. Além disso, o escalonamento empregado pelo CARONTE se preocupa em minimizar o tempo de processamento nas VCs, reduzindo o tempo de utilização dos recursos e, consequentemente, seu custo monetário. Portanto, os objetivos do CARONTE incluem a maximização do número de tarefas escalonadas nas VCs e minimização dos custos monetários. O CARONTE foi comparado à outras abordagens da literatura e os resultados indicam sua capacidade de escalonar um maior número de tarefas, minimizar o custo monetário de utilização dos recursos, minimizar o tempo de processamento das tarefas e minimizar o tempo de fila das tarefas. Em resumo, as principais contribuições deste trabalho podem ser destacadas da seguinte forma: *(i)* introdução de um mecanismo para escalonamento de tarefas que maximiza o número de tarefas escalonadas sem aumentar o custo monetário de utilização dos recursos computacionais e *(ii)* utilização eficiente de aspectos contextuais (mobilidade e requisitos das tarefas) no processo de tomada de decisão.

O restante deste artigo está organizado como se segue. A Seção 2 apresenta os trabalhos relacionados à esta pesquisa. A Seção 3 descreve o CARONTE. A Seção 4 apresenta a metodologia de avaliação e análise dos resultados. Por fim, a Seção 5 apresenta conclusões e trabalhos futuros.

2. Trabalhos Relacionados

Hattab *et al.* propuseram um escalonamento de tarefas em VCs com base no Problema de Atribuição Linear [Hattab et al. 2019]. O algoritmo possui duas etapas, onde na primeira as tarefas são classificadas de acordo com a proporção entre tempos de conclusão e espera. Na segunda etapa, seleciona um subconjunto de tarefas com a menor proporção e resolve uma sequência de Programas Lineares. O trabalho formula um problema de escalonamento cujo objetivo é minimizar o tempo de conclusão das tarefas escalonadas nas VCs. No entanto, os autores não consideram mobilidade veicular na formação da única VC presente no cenário, sendo esta constituída por veículos estacionados.

Da Costa *et al.* apresentaram o MORFEU, um mecanismo para escalonamento de tarefas em VCs baseado em otimização combinatória [da Costa et al. 2020]. Em sua operação, após um processo de formação de VCs, um controlador localizado em um nível superior na rede recebe as requisições por recursos (*i.e.*, as tarefas veiculares) e as escalona para processamento nas VCs disponíveis utilizando um algoritmo pseudo-polinomial para o Problema da Mochila 0/1. No entanto, os autores não consideram aspectos contextuais no processo de decisão de escalonamento. Ou seja, os impactos da mobilidade veicular e dos requisitos temporais das tarefas são desprezados.

Lieira *et al.* introduziram o TOVEC, mecanismo que utiliza a meta-heurística Otimização do Lobo Cinzento (*Grey Wolf Optimizer - GWO*) para seleção do melhor conjunto de tarefas a serem escalonadas nas VCs [Lieira et al. 2021]. O TOVEC tem como objetivo maximar a utilização de recursos veiculares, considerando em seu processo de decisão todos os requisitos das tarefas. Entretanto, o TOVEC considera apenas um controlador VEC, e não define onde o mesmo está inserido na rede. Ou seja, dependendo do nível em que o controlador esteja, as requisições ainda passarão pelo núcleo da rede, po-

dendo causar congestionamentos e, consequentemente, atrasos nos atendimentos. Além disso, o TOVEC não considera informações de mobilidade futura dos veículos.

Luo *et al.* apresentaram uma análise do atraso e custo do escalonamento de tarefas em VCs [Luo et al. 2021]. É estabelecido um *framework* de escalonamento com comunicação e computação para VCs, considerando tarefas com diferentes requisitos. Assim, um problema de otimização multiobjetivo é formulado para minimizar tanto o atraso quanto o custo. Para isso, um algoritmo de escalonamento baseado em Otimização por Enxame de Partículas (*Particle Swarm Optimization - PSO*) é proposto para obter as soluções Pareto-ótimas. No entanto, devido à sua abordagem bio-inspirada, seu tempo de convergência pode afetar o desempenho geral da solução. E além disso, os autores não consideram a mobilidade veicular como requisito no processo de escalonamento.

Wu *et al.* investigaram o problema de escalonamento de tarefas e otimização da alocação de recursos considerando o efeito da mobilidade veicular no ambiente VEC [Wu et al. 2020]. Especificamente, os autores formulam o problema de otimização conjunta em uma perspectiva *Min-Max* para reduzir a latência geral no escalonamento das tarefas. Além disso, considerando os padrões de movimento relativamente estáveis de um veículo em um curto período, é considerada a previsão de mobilidade para projetar um esquema baseado em previsão de mobilidade para obter melhores resultados. No entanto, o modelo de mobilidade não é realista, pois os veículos precisam de aceleração constante durante os processos de escalonamento de tarefas e alocação de recursos.

Com base na análise dos trabalhos relacionados, é possível observar que esses trabalhos, em sua maioria, não consideram a mobilidade do ambiente veicular em seus processos de decisão de escalonamento. E os que consideram, dependem de situações específicas, o que dificulta a implantação em ambientes mais dinâmicos. Além disso, mesmo considerando alguns requisitos das tarefas para tomada de decisão, como tamanho da tarefa e *deadline*, os trabalhos não os consideram em conjunto com outras informações contextuais, tais como a mobilidade veicular que dita a dinâmica dos recursos na rede.

3. Escalonamento de Tarefas em Nuvens Veiculares

Esta seção descreve o CARONTE, o qual considera controladores VEC para a seleção das melhores VCs para o escalonamento de tarefas. O CARONTE utiliza heurística de tempo polinomial para seleção do conjunto de tarefas a ser escalonado, bem como considera informações de mobilidade veicular para estimar os recursos disponíveis em cada VC.

3.1. Modelo de Sistema

A Figura 1 apresenta a arquitetura do sistema composta por veículos, BSs, controladores VEC e nuvem de Internet. Considera-se um cenário composto por um conjunto de x veículos, denotado como $u_i \in U = \{u_1, u_2, \ldots, u_x\}$. Também considera-se um conjunto de BSs implantadas na cidade, denotado como $b_y \in B = \{b_1, b_2, \ldots, b_p\}$, onde p é o número total de BSs. Todas BSs têm comunicação cabeada com a nuvem da Internet, e as BSs podem fornecer recursos de processamento e armazenamento na borda da rede, diminuindo o tempo de resposta para algumas aplicações [Liu et al. 2019].

Considerando a presença da rede 5G, cada BS possui uma interface Xn, que permite a troca de informações entre BSs vizinhas. Cada veículo possui Unidade de Bordo (*On-Board Unit - OBU*) que possibilita a comunicação entre veículos vizinhos (*Vehicle-to-Vehicle - V2V*) ou com BSs (*Vehicle-to-Infrastructure - V2I*). Por exemplo, os veículos podem se associar a uma BS e se comunicar com um Servidor Remoto (SR) na nuvem para acessar a Internet e solicitar recursos para processamento de dados.

Para associação dos veículos a uma BS, utiliza-se o método de associação baseado na Relação Sinal-Interferência-Ruído Máxima (Max-SINR). Neste caso, um veículo se associará a uma BS que fornece o Max-SINR. Em resumo, para um veículo receptor i localizado em algum ponto do espaço sob cobertura de pelo menos uma BS y, seu SINR(i, y) correspondente é dado por SINR$(i, y) = \frac{P_i}{I_i+N}$, onde P_i é a potência do sinal da BS no veículo receptor i, I_i é a potência de interferência dos outros sinais e N é a densidade espectral de potência do ruído gaussiano branco [Li et al. 2021].

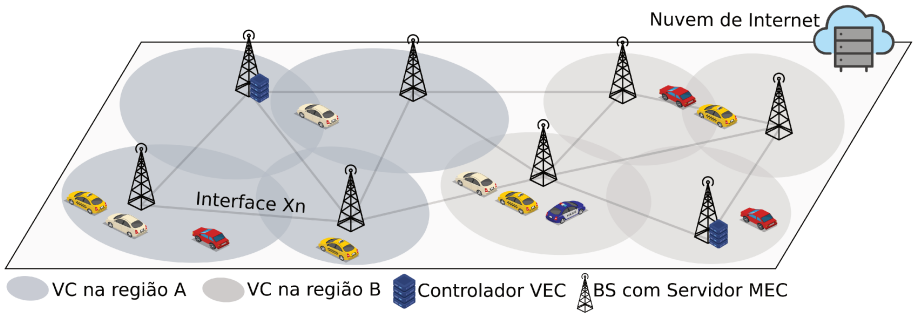


Figura 1. Representação da Arquitetura do Sistema

Nesse cenário, a cidade é dividida em R regiões e cada região possui pelo menos uma BS. Além disso, considera-se um conjunto de controladores VEC, denotados como $c_o \in C = \{c_1, c_2, \ldots, c_e\}$, onde $e = |R|$ é o número total de controladores, e está diretamente relacionado ao número de regiões R. A disposição ótima dos controladores no cenário não é o foco deste trabalho. Sendo assim, após a associação entre veículos e BS, a BS envia essa informação para o SR. As informações da BS são atualizadas à medida que o número de veículos em sua cobertura é alterado. Desta forma, os controladores VEC podem ser implantados em um subconjunto de BSs, e cada controlador é responsável por gerenciar as BSs em sua região da cidade. Esse gerenciamento inclui os processos de formação de VCs e escalonamento de tarefas.

No processo de formação de VCs, o controlador VEC precisa solicitar ao SR as informações de BSs e veículos para construir seu conhecimento regional. Nesse caso, considera-se o paradigma *Publish/Subscribe* para obter as informações relevantes sem inserir tráfego indesejado na rede. Considerando que o número de VCs é o mesmo número de BSs no cenário, o conjunto de VCs pode ser denotado por $v_j \in V = \{v_1, v_2, \ldots, v_m\}$, onde m é o número total de VCs. Logo, $m = p$. Em resumo, uma VC consiste em um conjunto de nós (*i.e.*, veículos e BS) capazes de compartilhar recursos de processamento ω (frequência de ciclo de CPU em MIPS[1]) e armazenamento ϕ em Megabytes (MB).

Neste contexto, ω_i denota a frequência de ciclo de CPU do veículo (MIPS), ϕ_i denota a capacidade de armazenamento do veículo (MB), ω_y denota a frequência de ciclo de CPU do BS (MIPS) e ϕ_y denota o capacidade de armazenamento do BS (MB). Portanto, cada VC é representada por uma tupla $\{id_j, veiculos_j, bs_j, total_j\}$, onde id_j é a identificação única da VC, $veiculos_j$ são os recursos dos veículos (ω_i e ϕ_i), bs_j são os recursos da BS (ω_y e ϕ_y), e $total_j$ é a soma dos recursos na VC (Ω_j e Φ_j). Ou seja, a quantidade total de recursos de processamento Ω_j e armazenamento Φ_j de cada VC v_j é a soma dos recursos compartilhados de veículos e BS que compõem essa VC.

Para garantir estabilidade nas VCs, em relação à variabilidade de recursos ao longo

[1]Milhões de Instruções por Segundo

do tempo, se faz necessária a utilização de informações de mobilidade veicular para estimar o tempo de permanência de veículos na cobertura da BS. Dessa forma, utiliza-se neste trabalho uma abordagem de predição de mobilidade ótima, pois a predição de mobilidade não é o foco deste trabalho. Ou seja, informações de mobilidade futura dos veículos são coletadas do *dataset* veicular considerando uma janela de tempo K e utilizadas no processo de decisão de escalonamento. Nesse sentido, como agora se tem uma estimativa dos recursos disponíveis nas VCs em $k \in K$ instantes de tempo, pode-se representar os recursos disponíveis nas VCs por Ω_{jk} e Φ_{jk}.

Ressalta-se que a abordagem considerada pode ser substituída por qualquer outra técnica de predição de mobilidade da literatura, considerando a relação entre tempo de predição e latência inserida no sistema pela técnica escolhida. No entanto, optou-se por utilizar essa abordagem para demonstrar de maneira simples os benefícios que informações de mobilidade podem proporcionar aos mecanismos de escalonamento.

3.2. Definição do Problema

Cada tarefa $t_l \in T = \{t_1, t_2, \ldots, t_n\}$ é denotada por uma tupla $\{id_l^t,\ s_l^t,\ w_l^t,\ D_l^t\}$ onde id_l^t representa a identificação única da tarefa, s_l^t (em MB) denota o tamanho dos dados de entrada da tarefa, w_l^t (em MI[2]) é o número de ciclos de CPU necessários para processar a tarefa e D_l^t é sua restrição de *deadline*.

De acordo com a literatura, para se calcular o atraso de atendimento para um escalonamento de tarefas em VC, todos os processos que adicionam atraso devem ser considerados [Wang et al. 2020]. No entanto, neste trabalho considera-se apenas o atraso de processamento da tarefa na VC (***i.e.***, tempo de execução de uma tarefa em uma determinada configuração computacional), pois essa métrica simplifica o entendimento da eficiência das soluções de escalonamento de tarefas. O atraso de computação d_{lj}^t pode ser obtido com base no ciclo de CPU necessário w_l^t dividido pela frequência do ciclo de CPU do servidor Ω_j, que neste caso é a VC, conforme a Equação 1.

$$d_{lj}^t = \frac{w_l^t}{\Omega_j}, \forall t_l \in v_j \tag{1}$$

Os recursos computacionais de uma VC são compartilhados entre as diversas tarefas escalonadas nessa nuvem. Assim, o Ω_j considerado para o cálculo de atraso de computação para uma determinada tarefa deve ser atualizado de acordo com o grau de compartilhamento desse recurso dentro da VC. O valor de Ω_j é dividido pelo número de tarefas $|T_j'|$ que foram escalonadas nessa VC, conforme a Equação 2.

$$\Omega_j = \frac{\Omega_j}{|T_j'|}, j \in V \tag{2}$$

Conforme observado, cada tarefa possui uma restrição de prazo D_l^t em sua configuração. Essa restrição representa um limite de tempo que a tarefa pode esperar até ser atendida. Portanto, se $d_{lj}^t \leq D_l^t$, então significa que a tarefa pode ser escalonada e executada na VC v_j. Assim, o sucesso do escalonamento de uma tarefa t_l pode ser obtido por $\alpha_l = \beta_{lj} \times \mathcal{C}_l$, onde β_{lj} é um valor binário. Se a tarefa t_l for processada, $\beta_{lj} = 1$; caso contrário, $\beta_{lj} = 0$. A parcela $\mathcal{C}_l$ representa o custo monetário de utilização do recurso computacional, modelado pela função mostrada na Equação 3.

[2]Milhões de Instruções

$$\mathcal{C}_l = d_{lj}^t \times (s_l^t \times \mathcal{P}(t_l)) \tag{3}$$

Onde d_{lj}^t é o atraso computacional da tarefa t_l na $vc_j \in VC$ e s_l^t é o tamanho da tarefa. $\mathcal{P}(t_l)$ indica o preço do recurso utilizado, definido conforme Equação 4. Os preços considerados baseiam-se nos preços de instâncias disponíveis no Amazon EC2[3].

$$\mathcal{P}(t_l) = \begin{cases} 1.750, & \text{se } t_l \text{ utiliza recursos das BSs} \\ 0.449, & \text{se } t_l \text{ utiliza recursos dos veículos} \end{cases} \tag{4}$$

De forma resumida, quando uma tarefa chega ao sistema, o controlador VEC deve selecionar a melhor VC para processar essa tarefa. Essa seleção deve considerar o poder de processamento da VC e o seu custo monetário associado, além dos requisitos da tarefa. Para coordenar esses objetivos distintos, formulou-se um problema de escalonamento de tarefas que, primariamente, busca maximizar o número de tarefas atendidas considerando restrições que impactam diretamente no custo monetário de utilização dos recursos, conforme mostrado na Equação 5.

A Restrição 6 garante que o deadline da tarefa seja respeitado e, consequentemente, colabora com a redução do custo monetário ao garantir o escalonamento em VCs que conseguem completar o processamento da tarefa, evitando reescalonamentos. Além disso, as Restrições 7 e 8 garantem que os limites de armazenamento e processamento das VCs durante os k intervalos de tempo necessários no processamento sejam respeitados.

$$\text{maximizar} \sum_{l=1}^{n} t_l,\ l \in T, \tag{5}$$

$$\text{sujeito a } d_{lj}^t \leq D_l^t,\ l \in T',\ j \in V, \tag{6}$$

$$\sum_{l=1}^{n} s_l^t \leq \Phi_{jk},\ l \in T',\ j \in V,\ k \in K, \tag{7}$$

$$\sum_{l=1}^{n} w_l^t \leq \Omega_{jk},\ l \in T',\ j \in V,\ k \in K \tag{8}$$

3.3. CARONTE

O problema de escalonar tarefas em VCs precisa levar em consideração mais de um aspecto durante seu processo de decisão. Além disso, a literatura discute que o problema de escalonamento de tarefas em VCs é $\mathcal{NP}$-difícil [Sorkhoh et al. 2019]. Dessa forma, o CARONTE considera uma heurística de aproximação chamada Primeiro Ajuste Decrescente (*First-Fit Decreasing - FFD*) do Problema do Empacotamento (*Bin Packing Problem - BPP*) para resolver esse problema em tempo polinomial. Ressalta-se que o BPP também é $\mathcal{NP}$-difícil, por isso a escolha pela heurística de aproximação.

Em resumo, o BPP resolve o problema do empacotamento de itens com pesos diferentes em um conjunto finito de caixas. Dado um conjunto de itens, o BPP decide

[3] `https://aws.amazon.com/pt/ec2/dedicated-hosts/pricing/`

quantas caixas serão necessárias para guardar tais itens, visando minimizar o número de caixas utilizadas. Com a heurística FFD, antes de colocar os itens nas caixas, eles são ordenados em ordem não crescente de pesos. Cada item tenta ser colocado na primeira caixa que pode acomoda-lo. Se nenhuma caixa for encontrada, uma nova caixa será aberta e o item será colocado nessa caixa. O FFD pode ser implementado para executar em tempo no máximo $\mathcal{O}(n \log n)$, onde n é o número de itens a serem empacotados.

Sendo assim, o Algoritmo 1 exemplifica como o sistema VEC funciona ao longo do tempo. Inicialmente, são criadas as filas de espera Q e execução R (Linhas 1-2). A cada instante de tempo $z \in Z$, onde Z é igual ao tempo total de simulação, um conjunto de tarefas chega ao sistema para ser escalonado, seguindo um processo de Poisson com $\lambda = [1, 2]$ (Linha 3). O conjunto de tarefas é enfileirado e fica em espera para ser escalonado (Linha 4). Para que o processo de escalonamento de tarefas ocorra, é necessário conhecer as VCs disponíveis no cenário. Assim, se z for igual ao intervalo de formação de VCs (K), ocorre o processo de formação e a informação das VCs é mantida até o próximo intervalo de formação de VC (Linhas 5-6). Se z for igual ao intervalo de escalonamento (K) e a fila Q não estiver vazia, as tarefas enfileiradas e o conjunto de VCs são passados para o CARONTE (Linhas 7-9) e o escalonamento será iniciado.

No funcionamento do CARONTE, enquanto existirem tarefas enfileiradas e VCs com recursos disponíveis, o CARONTE prioriza a VC de maior capacidade e a utiliza para resolução do BPP, passando as tarefas T e a VC selecionada (Linhas 10-14). O conjunto solução local S' contém as tarefas que podem ser escalonadas nessa VC. Com isso, para cada tarefa em S', é realizada uma verificação se a VC possuirá recursos suficientes por pelo menos até o deadline da tarefa (Linhas 15-16). Caso negativo, a tarefa é removida de S' e será reescalonada no futuro (Linha 17). Caso positivo, o atraso de computação d_{lj}^t é calculado e é verificado se a VC é capaz de processar essa tarefa por pelo menos até $d_{lj}^t \leq D_l^t$ (Linhas 18-20). Se a VC não for capaz, a tarefa é removida de S' (Linha 21). Caso contrário, as tarefas em S' são enfileiradas para execução em R e são removidas do conjunto de tarefas T. Caso o sistema não possua VCs disponíveis, o processo de escalonamento é encerrado até o próximo intervalo de escalonamento (Linhas 25-26).

Após o processo de escalonamento apresentado, é realizada a verificação periódica a cada intervalo de tempo se as tarefas na fila R atingiram seu tempo de processamento em cada VC (Linhas 27-28). Se o atraso computacional for maior ou igual a z, significa que a tarefa foi executada com sucesso. Desta forma, o custo monetário é calculado e a tarefa é removida das filas Q e R (Linhas 29-32). Caso contrário, a estimativa de execução da tarefa é atualizada, pois outras tarefas podem ter saído do sistema e mais recursos podem estar disponíveis na VC correspondente (Linha 33-34).

A complexidade de tempo do algoritmo CARONTE é analisada da seguinte forma (considerando somente as linhas de 10 a 26, que representam o mecanismo em si). A primeira etapa do mecanismo é executada em tempo $\mathcal{O}(n \log n)$ no pior caso, que é a complexidade de tempo do FFD. Nesta etapa, o algoritmo precisa de $\mathcal{O}(n \log n)$ para realizar a ordenação não crescente de peso das tarefas e percorrer todas elas. Além disso, o número de tarefas diminui com base na VC selecionada, sendo o número de VCs representado por m. A segunda e terceira etapas possuem complexidade linear $\mathcal{O}(n)$ no pior caso, para verificar as restrições de prazo e atraso computacional para cada tarefa $t \in S'$. Por fim, adiciona-se uma constante $\mathcal{F}$ que representa a complexidade computacional da etapa de predição, que pode variar conforme a técnica utilizada. Em resumo, a complexidade de tempo do mecanismo pode ser descrita como $\mathcal{O}(\max\{m\} + n \log n + 2n + \mathcal{F})$ no pior caso, com m sendo o número de VCs e n o número de tarefas. Ou seja, o CARONTE

Algoritmo 1: Abstração do sistema VEC com o CARONTE

```
1  Q ← ∅
2  R ← ∅
3  para cada intervalo de tempo z faça
4      Q.enfileira(T_z)
5      se z = intervaloFormacaoVCs então
6          V ← Forma VCs com base na estratégia de predição apresentada na Subseção 3.1.
7      se z = intervaloEscalonamento então
8          se Q ≠ ∅ então
9              T ← Q
10             enquanto T ≠ ∅ faça
11                 se V ≠ ∅ então
12                     v ← max{V}
13                     V ← V \ {v}
14                     S' ← BPP_FFD({T, v})
15                     para cada t ∈ S' faça
16                         se v NÃO tiver recursos até D_l^t então
17                             S' ← S' \ {t}
18                         senão
19                             d_lj^t ← Equação 1 + z
20                             se d_lj^t > (D_l^t + z) então
21                                 S' ← S' \ {t}
22                             senão
23                                 R ← S'
24                                 T ← T \ S'
25                 senão
26                     retorne
27     se R ≠ ∅ então
28         para cada r ∈ R faça
29             se d_r^t ≥ z então
30                 Calcula custo monetário conforme Equação mostrada na Subseção 3.2.
31                 Q.desenfileira(r)
32                 R.desenfileira(r)
33             senão
34                 Atualiza estimativa de computação de r em v_j ∈ V.
```

executa em tempo polinomial $\mathcal{O}(\max\{m\} + n \log n)$.

4. Avaliação

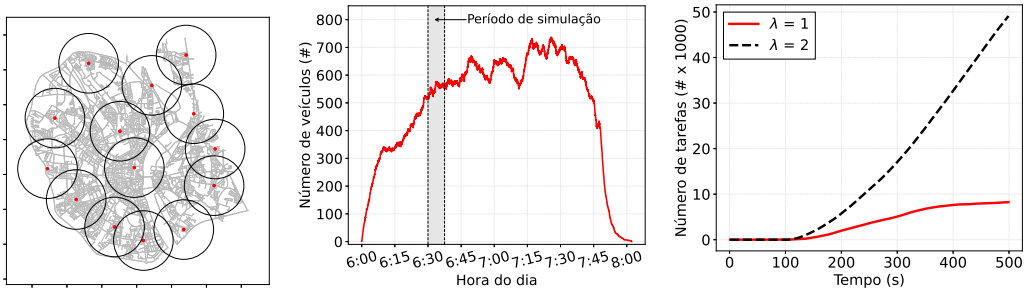
Esta seção descreve a metodologia e métricas utilizadas para avaliar a eficiência do mecanismo CARONTE. Foram realizadas simulações em um *trace* de mobilidade realística do centro da cidade de Colônia, Alemanha, para analisar a eficiência do CARONTE comparado-o à outras abordagens para escalonamento de tarefas da literatura.

4.1. Metodologia

Os experimentos foram realizados com o Simulador de Mobilidade Urbana (*Simulation of Urban MObility – SUMO*), na versão 1.2.0[4]. Os algoritmos foram implementados na

[4] `http://sumo.dlr.de/`

linguagem Python 3.8.10 e conectados ao SUMO através da interface TraCI. Foi utilizado o *trace* de mobilidade do projeto TAPASCologne[5], que reproduz o tráfego de veículos da cidade de Colônia, Alemanha. No entanto, apenas um recorte central de 114 km^2 foi considerado, com 2 horas de registros veiculares e até 700 veículos em seus horários de pico, conforme exibido nas Figuras 2(a) e 2(b). O tempo de simulação foi de 500 segundos, com 100 segundos de *warm-up*, conforme indicado na Figura 2(c). As simulações foram executadas 33 vezes para obtenção de um intervalo de confiança de 95%.



(a) Cenário Cologne (b) Veículos no cenário (c) Tarefas inseridas no sistema

Figura 2. Dados do cenário veicular utilizado na simulação

Foram consideradas aplicações *Bag-of-Tasks*, uma vez que podem ser executadas fora da ordem de submissão. O *deadline* das tarefas varia entre 1 e 9 segundos. Isso é importante para generalizar a representação de possíveis classes de aplicações. Os intervalos de formação de VCs e escalonamento foram de 10 segundos. A taxa de chegada de tarefas λ no sistema varia em 1 e 2 tarefas/segundo, seguindo uma distribuição de Poisson. O comportamento da chegada das tarefas é mostrado na Figura 2(c). As faixas de comunicação dos veículos e BSs foram de 300 e 2000 metros, respectivamente.

Além disso, o peso atribuído às tarefas foi de $s_l^t = \{10, 30\}$ e o número de unidades de processamento por veículo varia em $1, 2$ e 4 CPUs, que sem perda de generalidade representa $1, 2$ e 4 MIPS, respectivamente. A capacidade de armazenamento de cada veículo foi simplificada em $1, 2$ e 4 MB. Foram utilizadas 14 BSs e cada uma delas é capaz de compartilhar recursos de processamento e armazenamento, onde tais valores foram configurados em 25 MIPS e 25 MB, respectivamente. 4 controladores VEC foram considerados e cada um deles pode gerenciar até 4 BSs vizinhas. As BSs foram dispostas no cenário seguindo informações de posicionamento disponibilizadas pelo projeto TAPASCologne[5]. A Tabela 1 resume os parâmetros considerados nas simulações.

As métricas utilizadas para avaliação foram: i) *Tarefas escalonadas*, que representa a porcentagem de tarefas atendidas com sucesso; ii) *Custo monetário* referente ao tempo de utilização dos recursos; iii) *Atraso de computação*, que se refere ao tempo de processamento da tarefa em determinada configuração computacional; e iv) *Tempo de fila*, que diz respeito ao tempo entre a chegada da requisição e o atendimento dela.

O CARONTE foi comparado a três abordagens para escalonamento de tarefas em VCs, sendo: o **TOVEC** [Lieira et al. 2021], que utiliza a meta-heurística GWO no seu processo de decisão; o **MORFEU** [da Costa et al. 2020], que utiliza uma abordagem pseudo-polinomial de otimização combinatória no processo de escalonamento de tarefas; e o **UNC** [Hattab et al. 2019], que implementa a política FIFO sem restrições (*Unconstrained First In, First Out - UNC)* para o escalonamento no ambiente veicular. Por fim,

[5]http://kolntrace.project.citi-lab.fr/

para tornar justa a comparação, todas as abordagens priorizam a VC de maior capacidade a cada rodada para escalonar o maior número de tarefas possível por rodada.

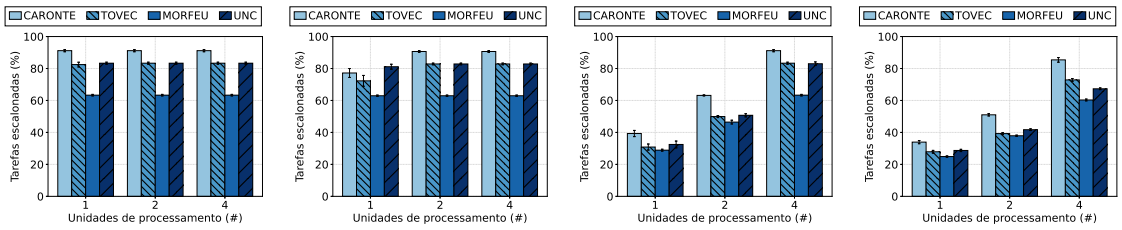
Tabela 1. Parâmetros de simulação

Parâmetro	Valor
Faixa de comunicação veicular (metros)	300
Faixa de comunicação das BSs (metros)	2000
Distribuição para chegada de tarefas	Poisson
Taxa de chegada de tarefas λ (tarefas/segundo)	$\{1, 2\}$
Peso médio das tarefas s_l^t (MB)	$\{10, 30\}$
CPU requerida para processamento w_l^t (MI)	$[1, \ldots, 10]$
Deadline D_l^t (segundos)	$[1, \ldots, 9]$
Unidades de processamento por veículo ω_i (MIPS)	1, 2, 4
Capacidade de armazenamento por veículo ϕ_i (MB)	1, 2, 4
Unidades de processamento por BS ω_y (MIPS)	25
Capacidade de armazenamento por BS ϕ_y (MB)	25
Intervalos de formação de VCs e escalonamento K (segundos)	10

4.2. Análise dos Resultados

A Figura 3 apresenta a porcentagem de tarefas que foram escalonadas e executadas com sucesso em diferentes unidades de processamento (CPUs) disponíveis por veículo. Todos os mecanismos melhoram seu desempenho à medida que o número de CPUs aumenta. No entanto, pode-se notar que todos eles têm dificuldade de escalonamento à medida que o peso das tarefas aumenta (Figuras 3(c) e 3(d)). Em todas as configurações, o CARONTE tem um desempenho superior em comparação com as outras abordagens.

Quando a taxa de chegada é igual a 1, o CARONTE consegue escalonar mais de 90% das tarefas nas configurações com o maior número de núcleos de CPU disponíveis (4 CPUs). TOVEC e UNC operam de forma estatisticamente semelhante nessa avaliação. O MORFEU tem o pior desempenho em todos os cenários. Isso se deve a sua estratégia de decisão, onde se preocupa apenas com o peso da tarefa, desconsiderando aspectos fundamentais como prazo e atraso computacional. É possível ainda observar o impacto que o aumento da taxa de chegada de tarefas causa no sistema. Na configuração mais desafiadora, com $p_i = 30$ e $\lambda = 2$, o CARONTE ainda se mostra superior, porém conseguindo escalonar apenas 36% das tarefas no cenário com 1 CPU disponível.

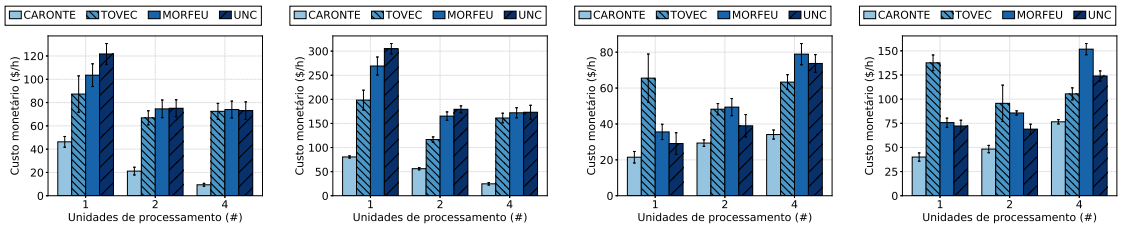


(a) $p_i = 10$, $\lambda = 1$ (b) $p_i = 10$, $\lambda = 2$ (c) $p_i = 30$, $\lambda = 1$ (d) $p_i = 30$, $\lambda = 2$

Figura 3. Tarefas escalonadas e processadas com sucesso.

A Figura 4 apresenta o custo monetário para utilizar os recursos computacionais das VCs. Para minimizar o custo, as abordagens optam por selecionar primeiro os recursos veiculares para o escalonamento. Pode-se notar que o CARONTE minimiza o custo monetário em todas as avaliações realizadas. Nesse caso, é natural que o desempenho

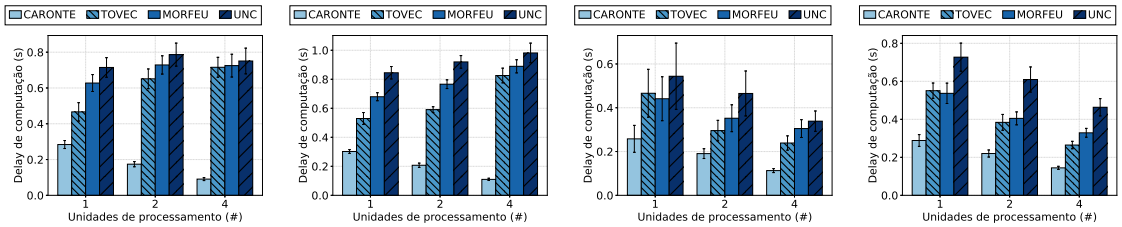
de todos os mecanismos caia devido ao percentual de tarefas escalonadas que também se enquadram em cenários mais desafiadores (Figura 3). O melhor desempenho do CARONTE ocorre devido à seleção das tarefas considerando os recursos futuros disponíveis na VC. A aplicação do BPP permite que um maior número de tarefas seja escalonado na mesma VC, priorizando que tais recursos sejam provenientes de veículos e não da BS. Além disso, quanto mais desafiador o cenário (aumentando a taxa de chegada de tarefas), mais caro se torna o processo de escalonamento. Um ponto interessante é observado nas Figuras 4(c) e 4(d), onde o custo monetário aumenta conforme o aumento de CPUs, isso se dá pelo aumento no número de tentativas sem sucesso, onde os mecanismos tentam escalonar tarefas mais pesadas que não conseguem ser completadas em poucas rodadas.



(a) $p_i = 10, \lambda = 1$ (b) $p_i = 10, \lambda = 2$ (c) $p_i = 30, \lambda = 1$ (d) $p_i = 30, \lambda = 2$

Figura 4. Custo monetário de utilização dos recursos computacionais.

A Figura 5 mostra os resultados referentes ao atraso de computação. Pode-se notar, em todas as avaliações, que o CARONTE reduz o atraso de computação das tarefas nas VCs. Isso se deve principalmente à seleção das VCs, que faz a filtragem com base na taxa de processamento da VC considerando a restrição de *deadline* da tarefa. Além disso, o UNC emprega um maior atraso computacional em todas as avaliações devido à sua estratégia de decisão que considera apenas o peso da tarefa. Especificamente, na configuração $p_i = 10$ e $\lambda = 1$, UNC, TOVEC e MORFEU estabilizam seu desempenho quando os veículos possuem mais CPU disponível. Nesta mesma configuração, o CARONTE opera com uma melhoria de pelo menos 50% em relação aos demais mecanismos. Na configuração com uma taxa de chegada de tarefas igual a 2, o CARONTE é pelo menos 50% melhor que as outras abordagens, em todos os cenários. Além disso, TOVEC e MORFEU operam de forma semelhante no processamento de tarefas quando o número de núcleos de CPU é igual a 2 e 4. Em resumo, o CARONTE gerenciou melhor os recursos computacionais ao considerar conjuntamente aspectos contextuais das tarefas e VCs em seu processo de tomada de decisão.

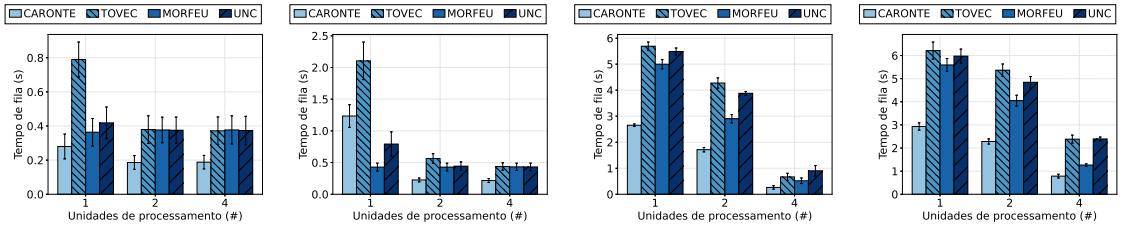


(a) $p_i = 10, \lambda = 1$ (b) $p_i = 10, \lambda = 2$ (c) $p_i = 30, \lambda = 1$ (d) $p_i = 30, \lambda = 2$

Figura 5. Atraso de processamento das tarefas nas VCs.

A Figura 6 mostra os resultados de tempo de fila no processo de escalonamento de tarefas. Essa métrica refere-se ao tempo de espera da tarefa até que ela seja escalonada e

executada nas VCs. Portanto, quando o tempo de fila é maior, significa que as abordagens não estão escalonamento tarefas com eficiência. O CARONTE se mostra superior em todos os cenários avaliados. No cenário $p_i = 10$ e $\lambda = 1$, o CARONTE é estatisticamente semelhante ao MORFEU e UNC, e ambos são pelo menos 50% mais eficientes que o TOVEC. Esse comportamento ocorre quando os VCs possuem menos recursos, justamente por ser um cenário mais desafiador para o processo decisório, como 1 CPU disponível por veículo. É possível observar que em todos os cenários o TOVEC tem um tempo de fila maior. Isso é justificado pela estratégia iterativa de seleção de tarefas para uma determinada VC, impactando o tempo geral de espera da tarefa. Em resumo, pode-se notar que o CARONTE possui um tempo de fila menor em todos os cenários avaliados, pois prioriza a relação entre o atraso de processamento e os recursos disponíveis na VC. No entanto, é fundamental relacionar esses resultados com o percentual de tarefas escalonadas. Em outras palavras, uma abordagem eficiente de escalonamento de tarefas deve maximizar o número de tarefas escalonadas e minimizar o tempo que essas tarefas aguardam até a conclusão de seu processamento.



(a) $p_i = 10$, $\lambda = 1$ (b) $p_i = 10$, $\lambda = 2$ (c) $p_i = 30$, $\lambda = 1$ (d) $p_i = 30$, $\lambda = 2$

Figura 6. Tempo de espera entre a chegada da requisição e o processamento.

5. Conclusão e Trabalhos Futuros

Este trabalho apresentou o CARONTE, um mecanismo ciente de contexto para escalonamento de tarefas em ambientes de VEC. O CARONTE divide seu processo de escalonamento de tarefas em três etapas. A primeira etapa se refere à aplicação do BPP com a heurística FFD para seleção do conjunto de tarefas que melhor se comporta em uma determinada VC. A segunda etapa verifica a restrição de prazo da tarefa, considerando as informações de mobilidade futura dos veículos e seus recursos computacionais que compõem a VC. Por fim, a terceira etapa verifica o tempo de processamento das tarefas na VC selecionada para reduzir o atraso de computação e, consequentemente, o custo monetário pelo uso dos recursos computacionais. O CARONTE possui complexidade de tempo polinomial. Os resultados de simulação em cenário realístico mostraram que, quando comparado à outras soluções, o CARONTE pode escalonar uma porcentagem maior de tarefas, minimizar o custo monetário pelo uso dos recursos, minimizar o tempo de processamento de tarefas nas VCs e minimizar o tempo de espera das tarefas enfileiradas. Como trabalhos futuros, pretende-se realizar avaliações com diferentes técnicas de predição de mobilidade para verificar o comportamento do mecanismo com erros de predição e propor controle de falhas para mitigar possíveis degradações no sistema.

Agradecimentos

Os autores agradecem o apoio concedido pelo Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) e pela Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP) por meio dos processos de Nº #2018/16703-4 e #2015/24494-8.

Referências

Boukerche, A. and Soto, V. (2020). Computation offloading and retrieval for vehicular edge computing: Algorithms, models, and classification. *ACM Computing Surveys (CSUR)*, 53(4):1–35.

da Costa, J. B. D., Peixoto, M. L. M., Meneguette, R. I., Rosário, D. L., and Villas, L. A. (2020). Morfeu: Mecanismo baseado em otimização combinatória para alocação de tarefas em nuvens veiculares. In *Anais do XXXVIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, pages 505–518. SBC.

de Souza, A. M., Oliveira, H. F., Zhao, Z., Braun, T., Villas, L., and Loureiro, A. A. F. (2020). Enhancing sensing and decision-making of automated driving systems with multi-access edge computing and machine learning. *IEEE Intelligent Transportation Systems Magazine*, 14(1):44–56.

Guidoni, D. L., Maia, G., Souza, F. S., Villas, L. A., and Loureiro, A. A. (2020). Vehicular traffic management based on traffic engineering for vehicular ad hoc networks. *IEEE Access*, 8:45167–45183.

Hattab, G., Ucar, S., Higuchi, T., Altintas, O., Dressler, F., and Cabric, D. (2019). Optimized assignment of computational tasks in vehicular micro clouds. In *II International Workshop on Edge Systems and Networking (EdgeSys 2019)*, pages 1–6. ACM.

Li, C., Zhang, B., and Tian, X. (2021). Throughput-optimal dynamic broadcast for sinr-based multi-hop wireless networks with time-varying topology. *IEEE Transactions on Vehicular Technology*, 00(0):0–0.

Lieira, D. D., Quessada, M. S., da Costa, J. B., Cerqueira, E., Rosário, D., and Meneguette, R. I. (2021). Tovec: Task optimization mechanism for vehicular clouds using meta-heuristic technique. In *2021 International Wireless Communications and Mobile Computing (IWCMC)*, pages 358–363. IEEE.

Liu, Y., Yu, H., Xie, S., and Zhang, Y. (2019). Deep reinforcement learning for offloading and resource allocation in vehicle edge computing and networks. *IEEE Transactions on Vehicular Technology*, 68(11):11158–11168.

Luo, Q., Li, C., Luan, T., and Shi, W. (2021). Minimizing the delay and cost of computation offloading for vehicular edge computing. *IEEE Transactions on Services Computing*, 1374:1–12.

Meneguette, R., De Grande, R., Ueyama, J., Filho, G. P. R., and Madeira, E. (2021). Vehicular edge computing: Architecture, resource management, security, and challenges. *ACM Computing Surveys (CSUR)*, 55(1):1–46.

Sorkhoh, I., Ebrahimi, D., Atallah, R., and Assi, C. (2019). Workload scheduling in vehicular networks with edge cloud capabilities. *IEEE Transactions on Vehicular Technology*, 68(9):8472–8486.

Wang, X., Ning, Z., Guo, S., and Wang, L. (2020). Imitation learning enabled task scheduling for online vehicular edge computing. *IEEE Transactions on Mobile Computing*, pages 1–14.

Wu, X., Zhao, S., Zhang, R., and Yang, L. (2020). Mobility prediction-based joint task assignment and resource allocation in vehicular fog computing. In *IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–6. IEEE.