TEC 502 - MI - Concorrência e Conectividade - 2025.1
Problema 1: Recarga de Carros Inteligente

Diego Silva[1], **Felipe Pinto**[1], **Lucas Trindade**[1]

[1]Bacharelado em Engenharia de Computação
Universidade Estadual de Feira de Santana.

{diego.cs282, felipepsilva0206, lucssto03}@gmail.com

Abstract. *Este trabalho apresenta a criação de um sistema em arquitetura cliente-servidor para o gerenciamento inteligente de pontos de recarga de veículos elétricos. Com o crescente aumento da frota de carros elétricos e a baixa infraestrutura de carregamento, foi proposta uma solução baseada em comunicação cliente-servidor capaz de comunicar para o carro/cliente os pontos de recarga mais próximos e com menos espera, evitando uma espera desnecessária e otimizando a rotina de carregamento dos veículos. O sistema foi dividido em 3 partes - client, server, charger - cada um com sua função definida e executados em containers Docker.*
Palavras-chave: *veículos elétricos, recarga inteligente, cliente-servidor, docker.*

1. Introdução

A crescente adoção de veículos elétricos no Brasil, impulsionada por variados fatores econômicos e ambientais, traz consigo a desafios a questão da infraestrutura dos pontos de recarga. Apesar da frota crescente, os seus pontos de recarga ainda são escassos e mal distribuídos, dificultando o acessos dos donos desses veículos e sendo um empecilho para que outras pessoas possam adquiri-los.

Diante desse cenário, foi proposto aos alunos matriculados no MI - Concorrência e Conectividade o desenvolvimento de um sistema de recarga inteligente. Nesse sistema deve ser realizada uma comunicação eficiente entre carros, o servidor na nuvem e os pontos físicos de recarga. O sistema foi criado utilizando contêineres Docker para cada componente do sistema, em um ambiente distribuído e utilizando a linguagem Go. A aplicação permite que o usuário/motorista possa reservar um ponto de recarga a sua vontade e conforme uma necessidade urgente(carro com nível crítico de bateria) além de oferecer os melhores pontos disponíveis na localização do carro e permitindo o pagamento posterior da recarga.

Esse relatório está organizado de forma que, nas seções seguintes, estão descritos os conceitos utilizados, arquitetura da solução, a divisão das responsabilidades entre as partes do sistema, os desafios enfrentados durante a implementação, resultados obtidos com os testes feitos em laboratório e além disso as conclusões e reflexões acerca do problema no geral.

2. Fundamentação Teórica

Para contextualizar o presente trabalho, alguns conceitos chave devem ser abordados. Essa seção tem por objetivo apresentar essas definições.

2.1. Sistemas Distribuídos

Sistemas Distribuídos são um tipo de sistema que se baseia em troca de mensagens. Tanenbaum define um sistema distribuído como um sistema em que um conjunto de computadores independentes se comportam como um único computador em relação ao usuário [TANENBAUM 2007].

A utilização de sistemas distribuídos pode trazer uma série de vantagens. Uma das principais vantagens é a confiabilidade do sistema, considerando que quando um computador deixa de funcionar, outros ainda realizam a tarefa a ser executada. Além disso, ainda podem ser citados como vantagens o compartilhamento de recursos, velocidade do sistema e economia [LOPEZ FUENTES 2015].

2.2. Arquitetura Cliente-Servidor

Se tratando de aplicações web, é muito comum seguir uma arquitetura cliente-servidor. Esse tipo de arquitetura tem duas partes principais, sendo elas o cliente e o servidor. O cliente é o processo que inicia a comunicação e realiza solicitações. O servidor responde às solicitações do cliente, provendo serviços. A comunicação entre esses atores é realizada através de algum tipo de protocolo, que irá especificar as regras para essa troca de mensagens [PICOLI 2011].

3. Metodologia

O sistema é dividido em três partes que se comunicam por meio de troca de mensagens JSON, sendo elas, o *charger* que representa e gerencia o acesso físico dos veículos aos pontos de recarga, o *server* que atua como intermediário da comunicação entre veículos e os pontos de recarga, além do cliente que simula as necessidades do veículo referentes ao seu carregamento. O esquemático dessa comunicação está apresentado no diagrama da Figura 1. Neste tópico veremos detalhadamente como cada um deles se comporta e se comunica.

3.1. Charger.go

O ponto de recarga atua como um servidor TCP autônomo que escuta uma porta específica para receber comandos vindos do servidor central. Cada ponto gerencia uma fila de carros que desejam utilizar sua estação. Para isso, ele mantém uma estrutura de dados em memória (*slice*) e utiliza um *Mutex(sync.mutex*) para garantir a integridade da fila durante os acessos concorrentes, uma vez que múltiplas requisições podem ser recebidas simultaneamente.

Cada ponto guarda também seu identificador, sua localização e um número de porta definido, de onde virão as mensagens necessárias para seu funcionamento. O projeto foi feito de forma a permitir a escalabilidade do sistema, sendo possível a existência de vários pontos de recarga. O ponto deve responder a 4 mensagens específicas, e, para cada uma delas, deverá responder e executar uma ação correspondente.

Além disso, o uso de goroutines permite que o ponto de recarga processe diversas requisições paralelamente, promovendo um comportamento assíncrono e eficiente. Sendo um padrão comum em sistemas distribuídos de baixa latência, onde a responsividade é essencial.

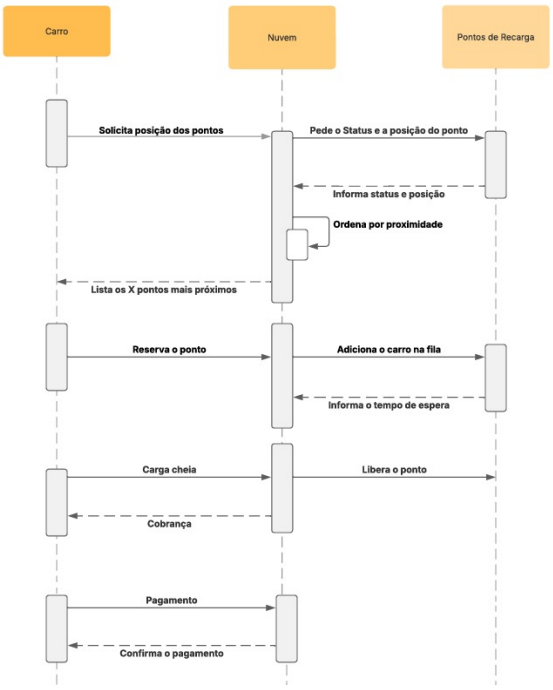

Figure 1. Diagrama de sequência

3.2. Server.go

O servidor atua como um intermediário na comunicação entre o cliente/carro e o ponto de recarga/charger, operando como um servidor TCP que escuta em uma porta definida e recebe requisições dos clientes, tratando-as de forma concorrente por meio de goroutines

A comunicação entre cliente e servidos é feita com sockets TCP/IP, utilizando o pacote *net* da linguagem Go. Cada mensagem recebida é um objeto JSON, estruturado com os campos *action*(tipo de operação) e *content* (dados relevantes). O servidor interpreta esse conteúdo para executar diferentes funções.

O servidor também simula uma consulta distribuída ao obter os dados dos pontos diretamente via socket. não armazenando essas informações localmente, garantindo que os dados estejam sempre atualizados.

3.3. Client.go

O módulo cliente simula o comportamento de um veículo elétrico inteligente, que interage com o sistema de recarga por meio de uma aplicação que se comunica via protocolo TCP. O programa funciona como uma aplicação interativa em linha de comando, oferecendo opções para que o ususario execute ações como buscar pontos de recarga, reserva de vaga, iniciar e finalizar o carregamento, além de pagamento posterior. Toda comunicação do cliente com o servidor é realizada por meio de sockets TCP, utilizando a biblioteca net da linguagem Go. As mensagens seguem um protocolo de comunicação simples baseado em objetos JSON contendo duas chaves principais:

action: identifica a operação solicitada.

content: transporta os dados necessários para a execução da ação.

O cliente também disponibiliza ao usuário um menu interativo para executar as ações necessárias. Essa interação é mediada por goroutines e canais, permitindo a execução concorrente de processos, c omo a simulação do consumo de bateria ao longo do tempo.

A bateria é monitorada em tempo real, e quando atinge um nível crítico, o sistema automaticamente envia uma solicitação de pontos de recarga ao servidor. Esse comportamento promove uma experiência autônoma e dinâmica, simulando um comportamento reativo do veículo.

4. Resultados e Discussões

A partir da implementação desse sistema, foi possível realizar o gerenciamento dos pontos de recarga inteligentes. Foram realizados testes manuais contemplando todos os fluxos principais requisitados no problema.

Alguns problemas relativos a concorrência não foram devidamente resolvidos no servidor, já que não foi implementado um *Mutex* nele. No entanto, nas condições de teste não ocorreram problemas de concorrência.

Como pontos de melhorias, foram elencados a possibilidade de realização de testes automatizados, através de *scripts*, e também o tratamento adequado das situações de concorrência.

5. Conclusão

A solução implementada demonstrou ser funcional e modular, permitindo que os veículos identifiquem os pontos de recarga mais próximos e menos ocupados, façam reservas de vagas e mais. A separação clara entre os componentes e o uso de concorrência e goroutines e mutexes garantiu integridade e acesso simultâneo, especialmente nas filas de espera e pontos de recarga.

Por fim, o trabalho cumpriu seu papel como exercício prático de redes, concorrência e arquitetura cliente-servidor, servindo como base sólida para projetos futuros, devido a possibilidade de escalabilidade do projeto

References

LOPEZ FUENTES, F. D. A. (2015). Sistemas distribuidos. *UAM*.

PICOLI, I. L. (2011). Arquitetura cliente-servidor em jogos multiplayer. B.S. thesis, Universidade Tecnológica Federal do Paraná.

TANENBAUM, A. S. (2007). Computer networks. tradução por vandenberg d. de souza. *Redes de Computadores. 4ªed. Rio de Janeiro: Campus/Elsevier*.