Amanda de Mendonça Perez - 211708002
Eduardo Adame - 211708006
Juan Belieni - 211708016
Kayo Yokoyama - 211708028
Marcelo Amaral - 211708022

# TRABALHO DA A1: PIPELINE ETL PARA O MONITORAMENTO DE RODOVIAS

Brasil
Abril de 2023

Amanda de Mendonça Perez - 211708002
Eduardo Adame - 211708006
Juan Belieni - 211708016
Kayo Yokoyama - 211708028
Marcelo Amaral - 211708022

# TRABALHO DA A1: PIPELINE ETL PARA O MONITORAMENTO DE RODOVIAS

Trabalho elaborado para a disciplina de Computação Escalável, do 5° período do curso de Ciência de Dados e Inteligência Artificial, com o objetivo de implementar um pipeline ETL para um sistema de monitoramento de rodovias.

Fundação Getulio Vargas - RJ
Escola de Matemática Aplicada

Professor: Thiago Pinheiro de Araújo

Brasil
Abril de 2023

# Sumário

**Sumário** . . . . . i

**1 MODELAGEM GERAL DO PROBLEMA** . . . . . 1

**1.1 Simulador** . . . . . 1

**1.2 ETL** . . . . . 2

**1.3 Dashboard** . . . . . 5

**2 PROBLEMAS DE CONCORRÊNCIA E PARALELISMO** . . . . . 6

**3 RESULTADOS E DISCUSSÃO** . . . . . 7

# 1 Modelagem geral do problema

O principal objetivo deste projeto é implementar, utilizando a linguagem C++, um pipeline de ETL (extração, transformação e carregamento - *load*) para um sistema de monitoramento de rodovias, aplicando conceitos estudados em aula para lidar com os problemas de concorrência envolvidos. Além da implementação do ETL em si, buscou-se também construir uma simulação de rodovia em Python, a fim de gerar os dados necessários ao sistema.

Tendo isso em vista, foi fundamental primeiramente modelar de maneira adequada o problema, visando a identificar e tratar precocemente possíveis obstáculos envolvidos nesse processo. A modelagem do problema, de modo geral, apesar de ter sido pensada a partir de uma interação envolvendo todas as partes envolvidas no projeto, pode ser dividida em três principais: primeiramente, no processo de geração dos dados (ou seja, o *mock*), em seguida em como lidar com esses dados no ETL, e por fim, em como gerar o dashboard a partir deles. Cada uma dessas partes está melhor detalhada a seguir.

## 1.1 Simulador

Para a implementação do *mock*, responsável por simular as rodovias, optou-se por usar uma modelagem discreta do espaço e do tempo. Desse modo, o passar do tempo ocorre em ciclos e, a cada ciclo, o programa recalcula a posição do veículo a partir de sua posição atual e velocidade, e de informações como aceleração e movimentação de outros carros. Ao final de cada ciclo, a posição dos veículos é armazenada em disco, visando a ser usada posteriormente no ETL.

Para representar esses dados, cada rodovia foi tratada como dois *grids* (cada um representando um sentido de deslocamento na pista) com $n$ linhas cada, onde cada linha representa uma faixa e cada coluna representa um trecho (de tamanho fixo) da estrada. Além disso, os carros presentes nas entradas dessa "grade" possuem, cada um, uma velocidade $v$ e uma aceleração $a$, sendo $v \in \mathbb{N} \cup \{0\}$ e $a \in \mathbb{Z}$; para facilitar o desenvolvimento do simulador, optou-se por utilizar unidades de medida genéricas para ambas, isto é, tanto aceleração quanto velocidade são medidas em termos de espaço no *grid* e ciclo decorrido.

Já para descrever o deslocamento dos veículos, além das informações determinísticas de velocidade e aceleração, foram adicionados fatores aleatórios, a dizer: probabilidade de troca de pista, probabilidade de um novo veículo surgir em cada pista e probabilidade de ocorrer uma colisão. Em especial, a probabilidade de ocorrer uma colisão, diferentemente das outras, foi construída como sendo não um valor fixo, mas sim um valor crescente em função da velocidade do automóvel (ou seja, quanto mais alta a velocidade, maior a chance de colisão). Essa escolha se relaciona diretamente com as decisões de projeto para o ETL, onde o cálculo para estimar a probabilidade de colisão considera que veículos com maior velocidade têm mais chance de sofrer um acidente.

Nesse contexto, primeiramente a probabilidade de troca de faixa atua definindo, logo de início, qual seria uma faixa "desejável" ao motorista e verifica se essa troca causaria colisões: caso não, o veículo

tentaria a troca; em caso positivo, ele mudaria de faixa, porém diminuindo a velocidade, de modo a ficar atrás do outro automóvel. Já a probabilidade de colisão lidaria com situações nas quais foi verificado esse risco, ou seja, se a próxima posição calculada para o carro causaria uma colisão, essa probabilidade definiria se o motorista consegue ou não realizar uma manobra para evitar o acidente.

A representação desses acidentes no modelo considera que há uma colisão quando existe mais de um veículo ocupando uma mesma posição no *grid*. Assim, cada espaço do *grid* pode conter de $0$ ou mais carros, dependendo da quantidade de automóveis envolvidos em uma colisão. Após alguns ciclos, os veículos acidentados são removidos da rodovia.

Por fim, para permitir ao ETL o acesso aos dados gerados, o simulador escreve em disco as informações que o sistema receberia em contexto real. Mais especificamente, os dados são registrados em um arquivo de texto, onde cada linha contém o id (placa) de um veículo (sequência de 7 caracteres alfa-numéricos) e sua posição de GPS naquele momento. Por simplicidade, a posição retornada não é a posição de GPS em termos de latitude e longitude, mas sim a posição do veículo na rodovia. Foram utilizados três valores para codificar essa informação:

- o primeiro valor representa o sentido da pista na qual o veículo está: 0 para o sentido da direita para a esquerda e 1 para o sentido da esquerda para a direita;
- o segundo valor representa a faixa na qual o veículo se encontra; e
- o terceiro valor, por fim, é a distância, ou seja, o trecho da pista onde o carro está.

Além dos dados descritos acima, também são fornecidas ao ETL informações refentes ao ciclo (por exemplo, o *timestamp* dos dados, que é necessário para algumas das análises solicitadas) e à rodovia (nome, velocidade máxima permitida, número de faixas e tamanho). Esses metadados foram entregues ao ETL por meio do cabeçalho do csv onde foram registrados os dados de placa e GPS.

## 1.2 ETL

As principais decisões de modelagem envolvidas no desenvolvimento do ETL estão diretamente relacionadas com problemas de concorrência e paralelismo. Por esse motivo, a maior parte do processo de implementação desse pipeline está melhor descrita na próxima seção. Contudo, ainda existiram algumas outras decisões de projeto relevantes de já serem mencionadas.

Uma dessas decisões diz respeito ao cálculo de probabilidade de colisão, solicitado no enunciado. Para tratar desse problema, foi modelada a seguinte fórmula:

$$\mathbb{P}(\text{colisão}) = \sigma\left(k\frac{v + |a|v}{v_{\max}} + p\right), \tag{1.2.1}$$

onde $\sigma(x) = (1 + e^{-x})^{-1}$ é a função sigmoide, que retorna sempre valores entre 0 e 1.

A ideia abstraída na equação acima é que, quanto maior a velocidade de um carro em comparação com a velocidade máxima permitida naquele trecho da rodovia, maior o risco de acidente. Além disso, em vez de considerar apenas a velocidade atual do carro, levou-se em consideração também sua aceleração, de modo que a razão $\frac{v+|a|v}{v_{max}}$ corresponda à "próxima velocidade" esperada para o veículo (dada velocidade atual e sua aceleração) sobre o limite de velocidade. Como $v_{max}$ é fixa em um determinado trecho, quanto maior a aceleração e/ou a velocidade, maior fica o valor dessa razão.

Note que em (1.2.1), a aceleração está em módulo, pois o modelo está assumindo que mudanças de velocidade aumentam a chance de colisão, tanto para acelerações negativas quanto positivas (por exemplo, se um motorista freia muito bruscamente, apesar de estar diminuindo sua velocidade, seu risco de colisão aumenta, pois podem haver carros atrás dele que não conseguiriam frear a tempo). Outra observação relevante é a presença dos valores $k$ e $p$ multiplicando e sendo adicionados à razão de velocidades; tratam-se de hiperparâmetros que atuam como termos de correção, cuja escolha se deu a partir de testes com diferentes valores, até que fosse encontrada uma função que atendesse ao comportamento esperado para definir uma probabilidade de colisão. Os gráficos a seguir, presentes nas figuras 1 e 2, mostram a cara dessa função quando aplicada a valores fixos de aceleração e velocidade, respectivamente, e consideram $k = 1$ e $p = -5$. Entretanto, os valores dos hiperparâmetros utilizados no ETL em si não foram exatamente esses, mas sim $k = 3$ e $p = -5$, que foram os valores que melhor se ajustaram ao problema, conforme foi observado após a realização de diversos testes.

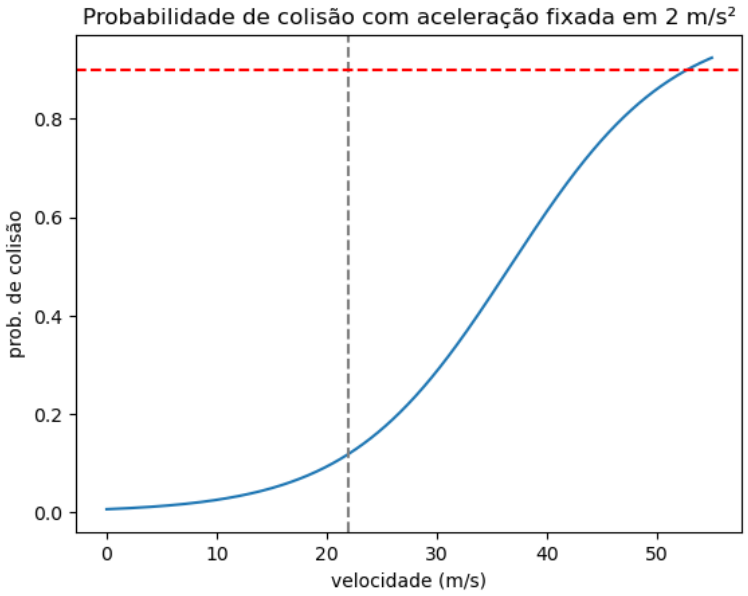


Figura 1 – Gráfico do risco de colisão com aceleração fixa considerando o limite de velocidade como 22 m/s.

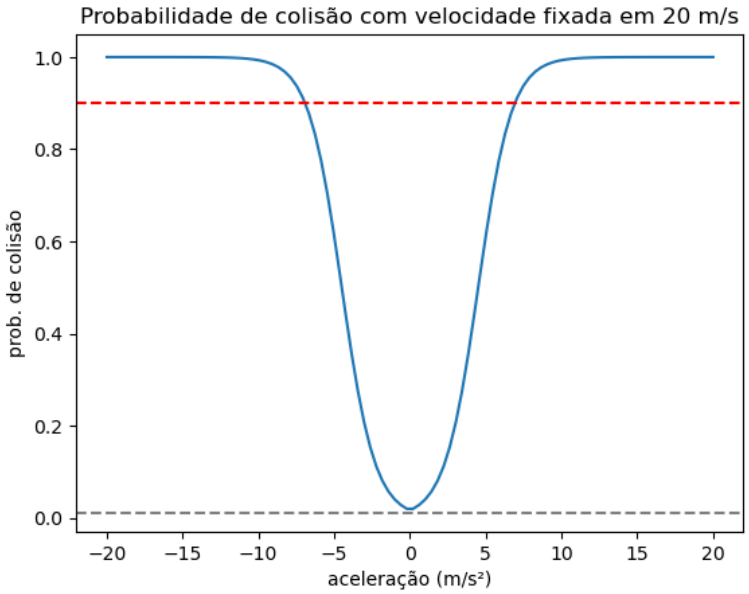


Figura 2 – Gráfico do risco de colisão com velocidade fixa considerando o limite de velocidade como 22 m/s.

Em ambos os gráficos, a velocidade limite estava fixada em 22 m/s (cerca de 80 km/h). Note que o risco de colisão é bem baixo enquanto a velocidade está dentro dos limites e, ao ultrapassá-lo, passa a crescer rapidamente, e esse mesmo risco é bem baixo quando a aceleração está próxima de zero, mas começa a crescer rapidamente ao se afastar desse valor.

Cabe, contudo, ressaltar que o modelo apresentado acima para estimar a probabilidade de colisão não se propõe a fornecer resultados matematicamente acurados, dado que esse não é o escopo do trabalho. Na realidade, seu objetivo é apenas fornecer valores que façam sentido (i.e., números reais entre 0 e 1) e que possuam uma interpretação razoável dentro do contexto do problema modelado.

Outra decisão relevante diz respeito à implementação de um serviço externo. Conforme solicitado, o serviço externo deveria ser modelado como sendo um sistema que fornece dados adicionais necessários ao pipeline ETL ou ao dashboard, mas extremamente ineficiente, capaz de lidar com apenas uma requisição por vez. Nesse sentido, ele foi modelado como um *mock* capaz de fornecer informações como nome do motorista, e modelo e ano do carro.

Para simular a incapacidade de lidar com mais de uma requisição, foram utilizados dois mutex: o primeiro bloqueia apenas a fila, evitando que duas threads aumentem a fila simultaneamente; já o outro, bloqueia o processamento da requisição. Desse modo, é possível que a fila aumente enquanto uma requisição é processada, porém nunca há mais de uma requisição sendo processada ao mesmo tempo. Ao ser chamada, essa função retorna um booleano indicando se ela processou o pedido ou não; assim, o ETL é capaz de verificar se foi ignorado ou não. Por fim, para simular a lentidão do sistema, foi usado um sleep.

Finalmente, cabe ainda ressaltar a maneira como foram tratadas as diferenças de prioridade estabelecidas pelo proposta do projeto. Segundo o enunciado do problema, o pipeline de ETL deveria ser capaz de lidar com diferenças em prioridade para diferentes tarefas, atribuindo uma preferência maior

para o cálculo de risco de colisão. Nesse sentido, buscou-se calcular, na etapa de transformação dos dados, primeiro a velocidade e a aceleração de cada veículo, uma vez que esses valores seriam usados para estimar o risco de acidente. Após esses dados serem calculados, o *transform* continua executando e processa os dados do serviço externo, modificando um vetor usado durante o processo de carregamento (*load*), então os dados são atualizados em tempo real (desde que o usuário pressione para mudar de veículo, uma vez que o dashboard permite que informações de um único veículo sejam visualizadas por vez[1]).

## 1.3 Dashboard

O principal desafio identificado na construção do dashboard foi a necessidade de minimizar o tempo de espera entre as transformações e a exibição das informações. Objetivando lidar com isso, inicialmente foram realizadas diversas tentativas para criar a visualização partindo diretamente do C++ e, consequentemente, dos dados armazenados em memória. Dentre as opções testadas, optou-se por utilizar a biblioteca *ncurses* do C++, muito recorrente na criação de TUIs - *Terminal User Interfaces*. Contudo, vale destacar as demais ferramentas avaliadas e o motivo de não terem sido mantidas no projeto:

- **FTXUI:** biblioteca para componentes em C++ inspirada no ReactJS. Não foi utilizada por não permitir a exibição de mais de um componente na tela, quando utilizando tabelas.
- **GTK:** usando a biblioteca oficial para C++ (gtkmm). Não foi usada pois utiliza uma estrutura não muito adequada para criar tabelas, e abastecê-las e atualizá-las não era compatível com o que vinha sendo feito no restante do projeto em termos operacionais nem com as lógicas implementadas. Além disso, há a dificuldade maior de se criar um GUI, ao invés de TUI.
- **ExpressJS + Chart.js/Tabulator:** foi implementado um servidor http em javascript com ExpressJS, possuindo uma url para GET e uma para POST do dataset. Utilizamos Curl e Json no C++ para realizar as requisições nesses dois endereços. A partir daí, restavam apenas decisões sobre como exibir os dados, entretanto havia o problema de ter que realizar algumas requisições HTTP para enfim o dado chegar a ser exibido; além disso, trata-se de uma implementação externa ao C++, o que não era de nosso agrado. Todavia, os códigos implementados para essa tentativa, apesar não estarem mais em uso, foram mantidos no repositório do projeto, na pasta `dashboard-js`, como código legado.

Nesse contexto, utilizando o Ncurses, foi necessário implementar tabelas capazes de receber entradas de usuário, como para selecionar filtros e para navegar pelos itens. Além disso, assim como no desenvolvimento do ETL, o dashboard também exigiu a utilização de técnicas de threading, uma vez que gostaríamos que cada construção gráfica fosse concluída antes de iniciar outra. Os problemas de concorrência envolvidos nisso estão melhor discutidos na próxima seção.

[1] Em outras palavras, o sistema atualiza em tempo real assim que todos os carros terminam de ser atualizados, mas a visualização desses dados depende de que o usuário "solicite" visualizar um novo veículo - só então o dashboard é atualizado.

# 2 Problemas de concorrência e paralelismo

As situações que envolveram lidar com problemas de concorrência ou a necessidade de paralelismo ocorreram durante a implementação do ETL e do dashboard. Para o caso do ETL, foram aplicadas técnicas de threading em diversas circunstâncias, como para lidar com a atualização de dados no terminal, a leitura dos dados, a interação com o serviço externo e o carregamento para o dashboard. As soluções apresentadas para esses casos e o processo como um todo para o desenvolvimento do ETL lidando com técnicas de threads, estão melhor descritos a seguir.

O programa inicialmente separa uma thread para lidar com a atualização dos dados exibidos no terminal e para lidar com input do usuário. A partir daí, começa um loop verificando os arquivos em cada pasta especificada nos argumentos da linha de comando. Os arquivos são verificados por índice, voltando ao 0 quando o maior índice especificado como uma constante da classe é atingido. Sabe-se que um arquivo está pronto para leitura quando um arquivo temporário de mesmo nome é criado, então o programa o exclui para evitar repetições.

A etapa de extração consiste em usar múltiplas threads para ler os dados do arquivo carregados para a memória RAM, e então colocar todas as threads, fora a principal, em espera até que a estrutura de dados usada (`unordered_map`) tenha reservado a quantidade de memória requerida, se necessário. Em seguida, as threads retornam a fazer a extração dos dados de modo balanceado, mas o único caso de concorrência tratado é caso uma nova placa seja inserida, pois consultar elementos existentes não altera o contêiner. Cada thread também tem um vetor atribuído a ela, no qual são armazenados os dados modificados para que ela os processe na etapa seguinte.

A partir daí, as threads são recriadas para a etapa de transformação dos dados. Informações como velocidade e aceleração são extraídas com prioridade (pois são usadas para o cálculo do risco de colisão) a partir do histórico de posições de um veículo. A unidade de tempo considerada foi número de ciclos, que é igual a 1, então é o cálculo leva em conta principalmente os dados brutos. Ele ocorre se houver pontos suficientes, o que possibilita o cálculo do risco de colisão. Caso contrário, um valor negativo é colocado, simbolizando para a leitura na próxima etapa que os dados são ausentes. Após o término do cálculo de risco, as threads esperam por uma variável de condição que sinaliza que a etapa de carregamento já recebeu os novos dados e eles estão distribuídos em contêiners que permitem a atualização dos valores enquanto o *dashboard* é executado. Então, as threads retomam a transformação por meio da comunicação com o serviço externo e a thread principal aguarda até que todas tenham encerrado a execução, o que pode atrasar o início da análise do próximo ciclo a depender do parâmetro que controla (até certo ponto) a ineficiência do serviço externo.

Na etapa de carregamento, os dados são copiados em um vetor para a thread que lida com input e exibição dos dados, pois optamos por exibir apenas um carro por vez. Os dados podem ser filtrados usando comandos exibidos na tela ou o programa pode ser encerrado. A biblioteca usada para facilitar a exibição foi `ncurses`, conforme já discutido na seção 1.3.

# 3 Resultados e discussão

Primeiramente, após exaustivas pesquisas não foi possível encontrar uma solução/fórmula simples que fosse muito funcional para o cálculo da probabilidade de colisão. Contudo, conforme já discutido na seção de modelagem do problema, optamos por utilizar a função da maneira como já foi apresentado, uma vez que essa parte da modelagem foge do escopo do problema. Além disso, como os valores retornados estavam, de fato, entre 0 e 1, consideramos que esses valores estavam bons o suficiente para seus propósitos. Em uma possível continuação deste projeto, poderia ser razoável tentar implementar outros modelos para o cálculo dessas probabilidades.

Paralelo a estas análises, realizando os testes com os programas implementados verificou-se também que os tempos entre as informações geradas e sua chegada na análise do *dashboard* podem ser um pouco inconsistentes, pois o processamento de um arquivo só tem início quando a comunicação com o servidor externo terminou para todas as placas processadas no ciclo anterior. Caso poucas novas placas entrem na simulação, os tempos se aproximam de 0. Ainda assim, em uma testagem geral os tempos obtidos foram satisfatórios e o sistema pareceu lidar bem com os problemas de concorrência relacionados ao serviço externo.

A figura a seguir mostra um exemplo de resultado apresentado no dashboard.

```
Dashboard

Número de rodovias: 1
Número de veículos: 317
Número de veículos acima do limite de velocidade: 20
Tempo entre simulação e análise: 0.001070 segundos;

< Veículos com risco de colisão (1/26) >

Placa: prdeomF
        Posição: (6, 464)
        Velocidade: 18.00
        Aceleração: 5.00
        Risco de colisão: 1.00
        Proprietário: Felipe Lima
        Modelo: March
        Ano de fabricação: 2014

Comandos:
        <: anterior
        >: próximo
        q: sair
        t: todos os veículos
        r: veículos com risco de colisão
        v: veículos acima do limite de velocidade
```

Figura 3 – Resultados apresentados no dashboard.