COMPUTAÇÃO ESCALÁVEL
Relatório Trabalho RPC

Autores: Ana Carolina Erthal, Bernardo Vargas, Cristiano Larréa, Felipe Lamarca, Guilherme de Melo e Paloma Borges.
Docente: Thiago Pinheiro de Araújo.

Escola de Matemática Aplicada
FGV EMAp

Rio de Janeiro
2023.1

1 Introdução

Este projeto consiste no aprimoramento do sistema de monitoramento de rodovias criado na A1, a fim de utilizar RPC como mecanismo de comunicação entre o simulador e o ETL. Com isso, torna-se possível que máquinas remotas conectadas através de uma rede enviem eventos para o ETL.

1.1 Solução Arquitetural

Para alcançar tal objetivo, desenvolvemos a seguinte solução: nosso arquivo `mock.py` passou a atuar como cliente do RPC, e ao invés de construir arquivos, constrói strings em formato `JSON`, linguagem nativa do banco de dados que escolhemos, o MongoDB. Essas mensagens são enviadas através do RPC (escrito em gRPC), e recebidas pelo `server.py`, que insere essas mensagens no banco.

Realizamos, também, as mudanças necessárias no ETL para lidar com a leitura dos dados, passando a utilizar o driver MongoDB C++.

1.2 Decisões de Projeto

Optamos por gerar os *stubs* cliente e servidor em *Python* armazenando os eventos em um banco de dados não relacional (MongoDB), que será utilizado como fonte de dados pelo extrator do ETL. A vantagem de se utilizar um banco de dados em detrimento de um *stub* de C++ é justamente o fato de que a implementação é bastante simplificada. Apesar de perder-se tempo no envio de mensagens devido a escrita e leitura no banco de dados, essa perda não justifica a escolha da outra abordagem.

A escolha do modelo de banco de dados utilizado envolveu uma comparação entre duas modelagens — uma utilizando NoSQL e outra utilizando SQL, tipos de bancos que estudamos em disciplinas anteriores. Nos interessamos por utilizar um banco de dados não relacional, em particular o MongoDB, pela facilidade em manter a estrutura de arquivo que já tínhamos, sem que houvesse necessidade de modelagens mais complexas.

Após pesquisarmos mais profundamente, percebemos que havia vantagens adicionais de utilização do MongoDB, principalmente a alta capacidade de escalabilidade da estrutura. Além disso, a adaptação do código da primeira entrega fica simplificado na medida em que conseguimos guardar todas as informações (nome da rodovia e os carros com placa e posição) em um dicionário e inserir como um `.json` no banco de dados.

1.2.1 Alterações no código

- Paralelização do simulador, utilizando a biblioteca `multiprocessing` para criar um determinado número de instâncias em paralelo, sem a necessidade de executar o comando em múltiplos terminais.

- O tempo de geração dos dados agora é passado como uma das propriedades do `.json` (antes era extraído do nome dos arquivos), e resolvemos um problema adicional de cálculo simples na A1.
- Um dos erros identificados na nossa entrega da A1 foi não fazermos a leitura dos arquivos em paralelo, mas a biblioteca de acesso ao MongoDB nos retornou um erro constante no cursor ao tentar solucioná-lo, então não abordamos esse problema.
- Correção da arquitetura para um ETL, visto que na A1 analisávamos o risco de colisão durante a extração dos dados. Agora temos uma divisão mais clara das etapas de Extract, Transform e Load.
- O simulador passou a ser executado com 5 lanes, para ultrapassar o número cores.

2 Resultados

Para os testes, utilizamos uma máquina executando o servidor e o ETL, e outras 3 máquinas executando o cliente (simulador). Para estabelecer a conexão localmente, criamos um *hotspot* na máquina do servidor e conectamos cada máquina com o servidor através de seu ip. Obtivemos o seguinte resultado de tempo médio de análises por número de rodovias:

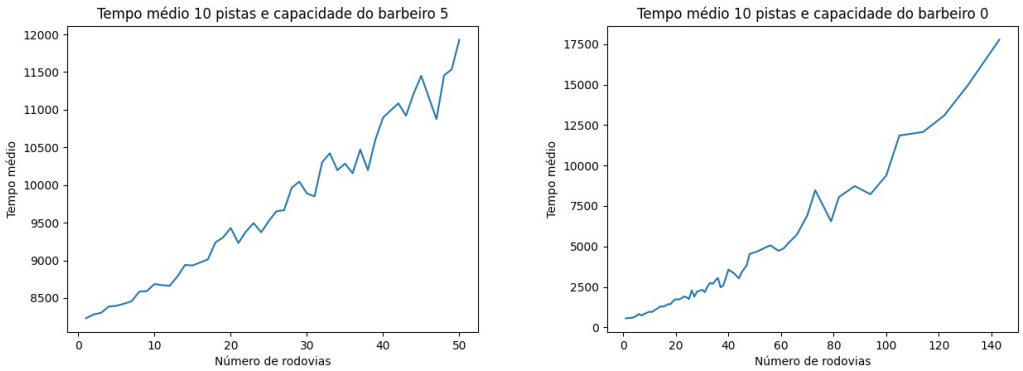

Figura 1 – Gráficos de tempo médio de análise para 50 e 150 rodovias e capacidade 5 no barbeiro

No primeiro gráfico fizemos um experimento utilizando apenas uma máquina externa levantando 50 instâncias da simulação, e uma fila com capacidade 5 no barbeiro, e no segundo repetimos o experimento com 150 instâncias (50 em cada máquina externa). Observamos em ambos um comportamento linear, e cogitamos a possibilidade de a causa ser a alta constante de tempo adicionada pela espera pelo barbeiro, e decidimos realizar testes zerando a capacidade da fila:

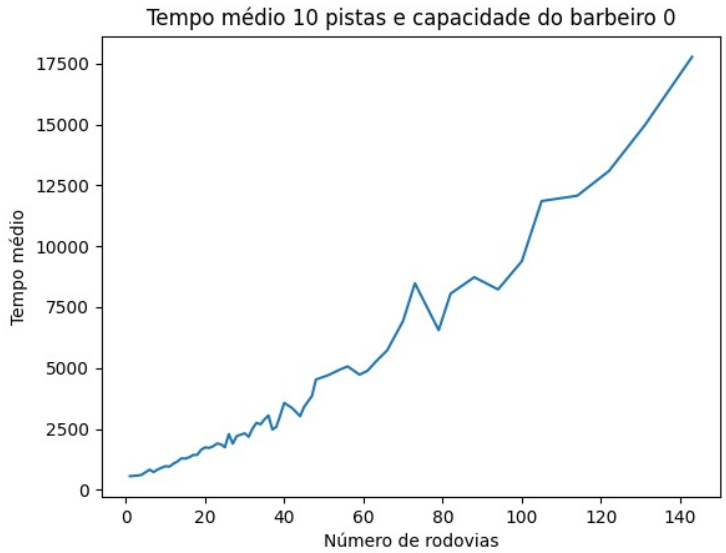

Figura 2 – Gráfico de tempo médio de análise para 150 rodovias e capacidade 0 no barbeiro

Não observamos, no entanto, nenhuma mudança na linearidade. Analisando melhor nossa arquitetura, concluímos que esse comportamento é esperado porque chegamos a um número de threads superior aos cores da CPU logo na primeira rodovia, uma vez que criamos *threads* por cada pista. Assim, um comportamento de constância a princípio (com aumento no tempo quando atinge-se o *bottom neck*) não seria razoável para nossa modelagem, dado que esse gargalo é atingido na primeira instância do *mock*.