Fundação Getúlio Vargas
EMAp - Escola de Matemática Aplicada
Ciência de Dados e Inteligência Artificial
Computação Escalável

RPC

Guilherme Buss
Guilherme Carvalho
Gustavo Bianchi
João Gabriel
Vinícius Nascimento

Rio de Janeiro
2025

1 Solução Arquitetural e Decisões de Projeto

Neste projeto, atualizamos a arquitetura original do sistema ETL, que processava dados a partir de arquivos locais, para uma arquitetura distribuída baseada em RPC (Remote Procedure Call).

1.1 Arquitetura Original

Na solução inicial, a comunicação entre o simulador das fontes de dados e o pipeline ETL era feita por meio de arquivos intermediários, onde os dados eram gerados por scripts Python em arquivos CSV/JSON, posteriormente lidos e processados pelo ETL implementado em C++. Essa abordagem limitava a execução a uma única máquina e introduzia latências relacionadas à escrita e leitura de disco, que é bem mais lenta que apenas passar os dados diretamente após serem "criados".

1.2 Nova Arquitetura com RPC

Implementamos um mecanismo de comunicação via gRPC `https://grpc.io/`, onde múltiplos clientes Python simuladores enviam eventos diretamente para o servidor ETL em C++ via rede, eliminando a necessidade de arquivos intermediários. Os eventos são transmitidos no formato serializado definido por um arquivo `.proto`, garantindo interoperabilidade entre as linguagens.

1.3 Decisões de Projeto

- **Uso do gRPC:** Escolhemos gRPC pela sua performance, suporte a múltiplas linguagens e facilidade de definição dos contratos via Protobuf. Enquanto nossos clientes foram feitos em Python, nosso servidor é em C++.
- **Cliente em Python:** Responsável pela geração de dados de teste com bibliotecas como Faker e Pandas, além de permitir simular múltiplas instâncias concorrentes.
- **Servidor em C++:** Recebe os dados do cliente Python e processa-os no pipeline ETL.
- **Medida de latência:** Agora coletamos timestamps nos eventos para medir a latência de ponta a ponta, entre a geração do evento no cliente e seu processamento no servidor.
- **Eliminação de arquivos intermediários:** Substituindo o armazenamento em disco pela comunicação direta via rede, reduzimos a latência e aumentamos a escalabilidade.

1.4 Melhorias Implementadas

Antes, os dados eram gerados em massa, salvos em arquivos CSV e depois processados, o que introduzia gargalos e dificultava testes em ambientes distribuídos. Com o RPC:

- Os clientes Python geram eventos em tempo real, enviando-os diretamente ao servidor.
- O servidor recebe e processa os eventos imediatamente, mantendo a mesma estrutura paralela do pipeline.
- O sistema permite múltiplos clientes concorrentes enviando eventos simultaneamente, simulando cargas reais distribuídas em rede.
- A arquitetura facilita a expansão para o uso de múltiplas máquinas simultaneamente, o que o uso e criação de arquivos não permitia.

Essas mudanças representam uma evolução na arquitetura, permitindo maior escalabilidade, menor latência e maior fidelidade no teste de carga e análise de desempenho do sistema.

2 Configuração do Ambiente e Uso do WSL

Durante o desenvolvimento da integração entre o cliente Python e o servidor ETL em C++ utilizando gRPC, enfrentamos dificuldades para executar o gRPC diretamente em máquinas Windows. Para fazer funcionar, utilizamos o **WSL (Windows Subsystem for Linux)**, que oferece um ambiente Linux completo dentro do Windows, garantindo maior compatibilidade e estabilidade,.

2.1 Motivação para Uso do WSL

O gRPC em C++ depende de várias bibliotecas e ferramentas que possuem suporte limitado ou apresentam instabilidades no ambiente Windows tradicional. O WSL fornece um sistema Linux real, onde as ferramentas de desenvolvimento, bibliotecas e o compilador protoc do Protocol Buffers podem ser instalados e utilizados sem restrições, permitindo compilar e executar o servidor gRPC de forma confiável.

2.2 Configuração do Ambiente

Foi necessário instalar diversos pacotes essenciais para desenvolvimento, compilação e execução do projeto, incluindo compiladores, ferramentas de build e bibliotecas como SQLite (para o banco de dados SQL), gRPC, Protocol Buffers e suas dependências. Esses componentes permitiram:

- Compilar o servidor C++ com suporte a gRPC e SQLite.
- Gerar os arquivos fonte a partir do arquivo `.proto` para C++ e Python.
- Executar o cliente Python com as bibliotecas gRPC necessárias.

Para facilitar o processo, reutilizamos a estrutura baseada em `Makefile` já empregada no projeto final da A1. O `Makefile` gerencia a compilação dos arquivos protobuf, a ligação com as bibliotecas e a geração dos executáveis, tornando o build simples e reproduzível dentro do WSL.

2.3 Compilação e Execução

Os comandos básicos para geração dos arquivos protobuf foram:

```
# Para C++
protoc --grpc_out=. --plugin=protoc-gen-grpc=$(which grpc_cpp_plugin) event.proto
protoc --cpp_out=. event.proto

# Para Python (no ambiente Python/venv)
python -m grpc_tools.protoc -I. --python_out=. --grpc_python_out=. event.proto
```

Na pasta `src` do projeto, a compilação e execução do servidor são feitas via `make`, apenas escrevendo make nesse ambiente, o que roda o nosso *MakeFile*.
O servidor gRPC foi executado dentro do WSL, enquanto os clientes Python puderam rodar em terminais externos, tanto no WSL quanto no Windows, conectando-se ao servidor via rede local (localhost). Essa configuração facilitou testes e simulação de múltiplos clientes concorrentes, aproximando-se de um cenário real distribuído e possibilitando expansão para múltiplas máquinas.

3 Discussão sobre Alternativas de Comunicação

Além da abordagem adotada com gRPC, existem outras maneiras viáveis para implementar a comunicação entre o simulador das fontes de dados e o pipeline ETL, como:

- **Cliente e servidor ambos em Python:** Usar uma solução 100% Python com bibliotecas como Flask, FastAPI ou ZeroMQ.
- **Comunicação baseada em arquivos ou banco de dados compartilhado:** Manter o uso de arquivos CSV, JSON ou banco de dados como meio intermediário.
- **REST APIs:** Expor o ETL como um serviço RESTful consumido pelo simulador via HTTP..

3.1 Justificativa da Escolha pelo gRPC

Optamos pelo gRPC com cliente Python e servidor C++ devido a:

- **Alto desempenho:** Uso de HTTP/2 e Protobuf garante comunicação rápida e eficiente, essencial para testes de carga e processamento em tempo real.

- **Multilinguagem:** Geração automática dos stubs facilita a integração entre simulador Python e ETL em C++, evitando serialização manual.
- **Comunicação síncrona e escalável:** Permite comunicação direta entre máquinas, superando limitações de arquivos locais e melhorando escalabilidade horizontal.
- **Facilidade de manutenção:** Contrato de dados centralizado via `.proto` reduz erros de incompatibilidade.

4 Resultados

Para avaliar o desempenho da arquitetura RPC implementada entre o simulador e o pipeline ETL, foram realizados testes de carga variando o número de instâncias do simulador de 1 a 20. Em cada experimento, foi computado o tempo médio de resposta, definido como o intervalo entre o envio de um evento pelo simulador e a conclusão da sua análise pelo pipeline ETL.
No arquivo *response_times.csv* da pasta *gRPC* do nosso repositório, é possível ver nossos resultados, em que os testes foram feitos com 4 threads:

Nº de Clientes	**Tempo Médio de Resposta (s)**
1	0.9507
2	1.2076
3	1.3628
4	1.7969
5	1.8896
6	2.4543
7	2.9994
8	3.2603
8	3.8493
9	4.4664
10	4.9240
11	5.4684
12	6.1036
13	8.0591
14	7.7512
15	7.7095
16	10.8900
17	10.6146
18	9.5894
19	10.6811
20	13.6880

Tabela 1: Tempo médio de resposta em função do número de clientes simultâneos

Podemos ver que há um crescimento no tempo médio de resposta conforme o número de clientes aumenta, o que era esperado dado o maior volume de requisições concorrentes sendo processadas pelo servidor ETL. Até cerca de 8 a 12 clientes, o sistema manteve um crescimento (relativamente) linear e controlado no tempo de resposta. No entanto, a partir de 13 clientes, há um salto mais significativo nos tempos. Isso, provavelmente, é causado por uma possível saturação dos recursos computacionais da máquina servidora.
Ainda assim, a solução baseada em RPC demonstrou escalabilidade razoável, mantendo o funcionamento do sistema mesmo sob carga elevada, com até 20 clientes simultâneos.

5 Conclusão

Considerando o projeto como um todo e a necessidade de escalar para múltiplas máquinas, diversificar clientes e reduzir latência, o uso do gRPC com cliente Python e servidor C++ oferece o melhor equilíbrio entre desempenho, interoperabilidade e escalabilidade, alinhando-se perfeitamente aos objetivos da comunicação eficiente entre simulador e pipeline ETL distribuído.