Computação Escalável

Comunicação RPC para Sistema de Reservas de Viagens

Gustavo Tironi, Kauan Mariani Ferreira, Matheus Fillype Ferreira de Carvalho,
Pedro Henrique Coterli, Sillas Rocha da Costa

25 de maio de 2025

Conteúdo

1 Introdução

Para um sistema de reserva de viagens, que precisa agregar informações de diversas plataformas como Decolar, Trivago, Booking ..., a capacidade de lidar com múltiplas requisições provenientes de diferentes máquinas é fundamental para o seu pleno funcionamento.

Imagine um cenário onde um sistema de reserva de viagens precisa consolidar ofertas de voos, hotéis e pacotes de diversas fontes, com cada uma dessas fontes gerando eventos (novas ofertas, atualizações de preços, disponibilidade) que precisam ser consumidos por um processo de ETL (Extract, Transform, Load) e disponibilizados aos usuários. É exatamente o caso que estamos simulando aqui, como pode ser visualizado no diagrama da Figura 1.

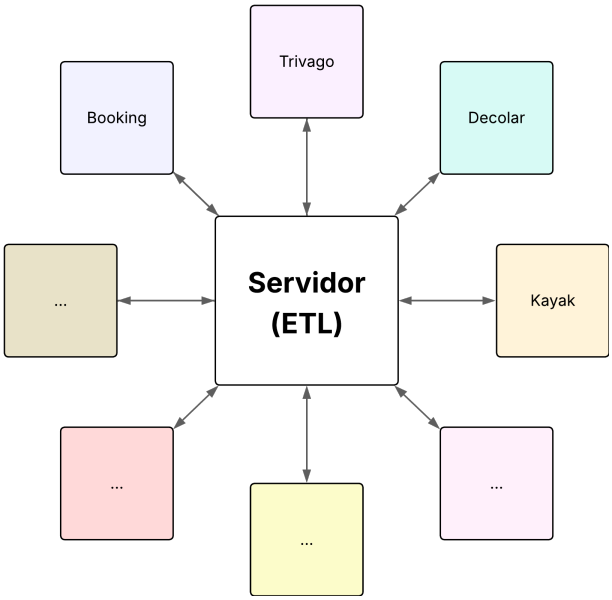

Figura 1: Diagrama ilustrativo

Nesse contexto, o mecanismo de Remote Procedure Call (RPC) emerge como uma solução poderosa. Ele possibilita que instâncias de simulação, em máquinas diferentes, possam emitir eventos para o ETL de forma eficiente, como se estivessem chamando uma função local. A utilização de RPC facilita a distribuição da carga de trabalho e a escalabilidade do sistema, garantindo que o pipeline de processamento de eventos possa lidar com o volume crescente de informações de maneira eficaz.

2 Decisões de Projeto

Escolhemos a abordagem de **gerar o stub cliente em Python e o stub do servidor em C++**. Nesta abordagem, a comunicação entre o simulador e o servidor foi feita por strings que representam dados em formato CSV. Além disso, consideramos que todos os dados de um evento seriam enviados de uma só vez pelas máquinas clientes, dado o contexto da aplicação, sendo que o processamento em lote (batch) seria feito pelo servidor ETL.

2.1 Escolha da abordagem

A escolha da abordagem de usar stubs em Python para o cliente e em C++ para o servidor foi impulsionada principalmente pela forma como nosso framework existente foi construído. Como o código do nosso servidor ETL já estava feito e funcional em C++, as vantagens dessa abordagem foram potencializadas em relação às desvantagens. Priorizamos o processamento de eventos em tempo real com alta eficiência. Ao comparar as abordagens, nossa escolha apresentou as seguintes características:

Vantagens:

- Eliminação da latência adicional introduzida por uma camada intermediária (como uma interface em Python e um banco de dados).
- O uso de stubs em C++ no servidor permite aproveitar o alto desempenho da linguagem no processamento de eventos.
- Dispensa a necessidade de gerenciar um banco de dados, reduzindo a complexidade e os custos de manutenção da infraestrutura.

Desvantagens:

- Maior complexidade na configuração inicial do sistema.
- Aumento do acoplamento entre cliente e servidor, dificultando a flexibilidade e a modularidade da aplicação.
- Menor tolerância a falhas, uma vez que, na ausência de persistência em banco de dados, eventuais falhas podem resultar na perda definitiva dos dados processados. Com o uso de um banco, é mais fácil retomar a operação a partir do ponto de falha, pois os dados permanecem armazenados.

2.2 Comunicação de Eventos via String

Para a comunicação entre o simulador (cliente RPC) e o ETL (servidor RPC), definimos que os eventos seriam transmitidos como uma string que representa um CSV (Comma Separated Values). Esta decisão foi tomada por sua simplicidade e facilidade de implementação. Nosso pipeline inicial já conseguia lidar com o processamento desse tipo de string, facilitando a implementação e garantindo a eficiência do processo.

Embora existam outras opções, como a definição explícita de campos no Protocol Buffers, a escolha do CSV via string para esta fase do projeto equilibrou a necessidade de um formato de dados estruturado com a simplicidade de implementação e a agilidade no desenvolvimento, considerando a natureza do exercício.

2.3 Envio de Dados

A decisão de enviar todos os dados de uma só vez pelas máquinas clientes, ou seja, as três tabelas (voos, hotéis e reservas) de uma vez, com o processamento em batch sendo realizado pelo servidor, foi baseada no contexto de negócio ao qual nosso exemplo está inserido e nas características de desempenho.

Em nosso cenário, onde o ETL processa dados de diferentes provedores (como Decolar e Trivago), não faria sentido se os dados fossem enviados em batch ou de forma intercalada entre diferentes fontes para uma mesma análise. Para processar uma requisição da Decolar, por exemplo, é necessário que todos os dados relacionados a essa requisição sejam recebidos para que a análise seja completa e consistente. Receber dados pouco a pouco de diferentes fontes para uma mesma análise poderia bloquear o processo desnecessariamente, devido a latências de comunicação ou demoras no envio por parte do cliente.

Além disso, como nosso servidor tem alta capacidade de processamento, faz sentido que o sistema seja limitado principalmente pela capacidade de processamento do servidor e não pela capacidade de comunicação de eventos fragmentados, mesmo que a comunicação em si ainda possa ser mais rápida.

Pensando também na integração, essa abordagem simplifica a lógica do cliente e concentra a complexidade no servidor. Isso é benéfico, uma vez que o objetivo é poder adaptar vários clientes diferentes com maior facilidade.

3 Resultados Experimentais

Nesta seção, apresentamos os resultados obtidos a partir da simulação do sistema com diferentes quantidades de clientes concorrentes realizando requisições ao servidor.

A contagem do tempo de resposta foi realizada considerando o instante em que o cliente envia os dados (ou seja, inclui o tempo de envio até o servidor) até o momento em que o mesmo cliente recebe a resposta processada. Isso significa que o tempo total inclui não apenas o processamento no servidor, mas também o tempo de transmissão de dados pela rede.

Após enviar uma requisição e receber a resposta correspondente, cada cliente aguarda um intervalo aleatório entre 3 e 10 segundos antes de realizar uma nova requisição. Essa estratégia visa simular de forma mais realista um cenário de uso em produção, com múltiplos clientes interagindo com o sistema de forma assíncrona e não contínua.

Para cada teste, variamos o número de clientes concorrentes entre os seguintes valores: 1, 2, 3, 4, 5, 7, 9, 11, 14, 17 e 20. Cada experimento foi executado por pelo menos dois minutos, permitindo a coleta de um número significativo de amostras para o cálculo da média do tempo de resposta. Além disso, quanto maior o número de clientes no teste, mais requisições foram processadas (mais tempo), garantindo uma representatividade estatística consistente para a média obtida.

O servidor responsável pelo processamento foi executado em uma máquina com processador Intel i5-11300H, 24GB de memória RAM e 8 núcleos. As comunicações foram realizadas através da rede Wi-Fi da FGV, que apresenta certa instabilidade e variação de latência, um fator relevante para interpretar os resultados.

O primeiro resultado analisado é a distribuição dos tempos de resposta registrados durante um dos testes com maior carga: 20 clientes concorrentes. Como esse cenário representa o ponto de maior concorrência e potencial saturação do servidor, ele oferece uma visão clara sobre o comportamento do sistema sob estresse. É possível observar que, embora a maioria das respostas tenha tempos relativamente baixos, existe uma dispersão significativa nos valores. Essa assimetria indica que, em momentos de menor concorrência, as requisições são processadas rapidamente, enquanto que, em picos de carga, o tempo de espera aumenta para alguns clientes.

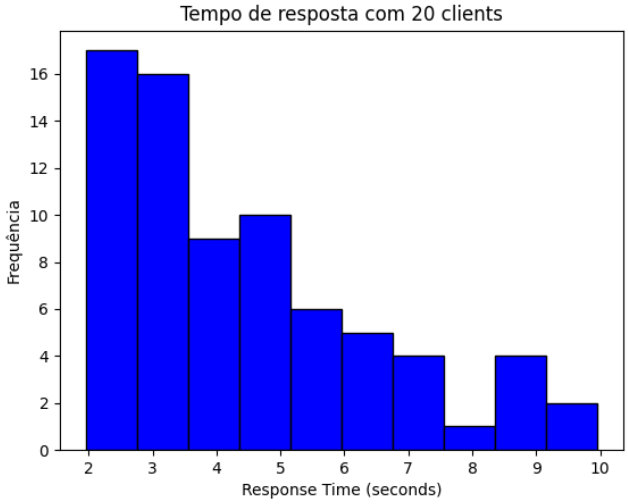

Figura 2: Histograma dos tempos de resposta no teste com 20 clientes

Essa variação se deve ao fato de que, aleatoriamente, diferentes máquinas podem realizar requisições em simultâneo, provocando filas no servidor. Por outro lado, quando poucas máquinas requisitam ao mesmo tempo, as respostas tendem a ser mais rápidas. O histograma apresentado na Figura 2 mostra uma distribuição enviesada à esquerda, o que indica uma maior concentração de tempos baixos, um comportamento esperado.

A seguir, avaliamos o tempo médio de resposta em função do número de clientes (Figura 3).

Figura 3: Tempo médio de resposta em função do número de clientes concorrentes

Como esperado, o tempo médio aumenta à medida que mais clientes são adicionados,

devido ao aumento da carga computacional e da concorrência de rede. No entanto, o crescimento não segue uma linha perfeitamente linear, apresentando um padrão de crescimento irregular.

Essa variação pode ocorrer por diversos fatores, mas algo que vale ser mencionado é que entre os testes com 7 e 9 clientes, por exemplo, houve uma percepção do grupo de que a qualidade da rede Wi-Fi da FGV se deteriorou, o que pode ter impactado negativamente os tempos de resposta nesse intervalo.