Relatório da tarefa 3: RPC como mecanismo de comunicação

Computação Escalável

Dávila Meireles,
Dominique de Vargas de Azevedo
Lívia Verly
Pedro Thomaz Conzatti Martins
Thiago Franke Melchiors

Professor: Thiago Pinheiro de Araújo

Rio de Janeiro
2024

Conteúdo

1 Introdução

No contexto do desenvolvimento de sistemas de processamento de dados em larga escala, a eficiência e a escalabilidade são aspectos fundamentais para o sucesso de um projeto. No trabalho da A1, foi desenvolvido um micro-framework com o objetivo de facilitar a criação de pipelines de processamento de dados. Esse framework foi desenhado para garantir eficiência, balanceamento de carga e baixo acoplamento entre os componentes.

Neste trabalho, daremos continuidade ao projeto desenvolvido na A1, focando em evoluir a comunicação entre a fonte de dados CadêAnalytics e um simulador de processamento de eventos. Na A1, a comunicação entre a simulação de dados e o pipeline de processamento era limitada a uma única máquina, utilizando chamadas de método ou sistema de arquivos. Para superar essa limitação e permitir a operação em um ambiente distribuído, adotaremos o mecanismo de Remote Procedure Call (RPC) utilizando a biblioteca gRPC. Sabendo que o foco da tarefa é o RPC, o ETL não é algo essencial nessa tarefa em específico, então como não havíamos completado essa parte durante a A1, simplificaremos o problema, conforme indicado pelo professor.

A utilização de gRPC permitirá que instâncias de simulação, executadas em máquinas remotas, possam emitir eventos para o servidor através de uma rede, ampliando a escalabilidade e a flexibilidade do sistema. Para testar a eficácia dessa nova arquitetura, realizaremos um teste de carga, analisando o desempenho do servidor à medida que aumenta o número de clientes enviando eventos simultaneamente.

2 Arquitetura do sistema

2.1 Escolha da Tecnologia: Python

A escolha de Python foi recomendada pelo professor devido à facilidade de uso e à rápida prototipagem que a linguagem proporciona, e também pelo fato de que temos a limitação de o trabalho da A1 estar incompleto. Além disso, Python possui uma vasta quantidade de bibliotecas para análise de dados (como pandas) e é amplamente utilizado na comunidade acadêmica e na indústria.

Embora Python seja menos eficiente que linguagens de baixo nível como C++ (devido ao seu interpretador, gerenciamento de memória e tempo de execução), a facilidade de desenvolvimento e a legibilidade do código compensam essa desvantagem em muitos casos de uso.

2.2 Componentes do Sistema

- Simulador de Eventos (Client) - É responsável por gerar eventos simulados que representam interações de usuários com diversos componentes do sistema. Gera uma lista de eventos com atributos como data, ID do usuário, estímulo e alvo, e envia esses eventos para o servidor gRPC. É o arquivo `cade_analytics_client.py`.
- Servidor de Processamento de Eventos (Server) - Recebe os eventos enviados pelo simulador, processa-os transformando em DataFrames do Pandas e realiza análises simples calculando métricas como o número de eventos por usuário, por componente alvo e por data. É o arquivo `cade_analytics_server.py`.

2.3 Comunicação via gRPC

Nosso arquivo `.proto` define um serviço e duas mensagens:

- Mensagens:
  1. `Event`: Codifica um evento, contendo um inteiro representando a data no formato Unix timestamp, um inteiro representando o ID do usuário, um enum representando o estímulo realizado e um enum representando o componente alvo do estímulo.
  2. `EventList`: Uma lista de eventos.
- Serviço: Possui apenas um método `SendEvents` que recebe uma lista de eventos. Este método é usado para enviar os eventos do cliente para o servidor, onde são armazenados para serem posteriormente analisados.

A partir desse arquivo `.proto`, as ferramentas gRPC geraram os códigos de cliente e servidor em Python, facilitando a implementação da comunicação entre os componentes.

2.4 Integração entre os Componentes

- Geração de Eventos no Cliente: O cliente gera eventos simulados e cria uma instância de EventList e utiliza o stub gRPC para enviar a lista de eventos ao servidor.
- Recepção e Processamento de Eventos no Servidor: O servidor gRPC recebe a lista de eventos, converte os eventos em um DataFrame do Pandas, realiza análises simples e imprime os resultados.

2.5 Configuração da Rede

Para as configurações da rede o cliente pergunta quantas repetições executar com cada quantidade de threads, o endereço IP e a porta, e o servidor gRPC é configurado para executar esses inputs que o cliente dá.

3 Principais escolhas e justificativas no design do sistema

3.1 Protocolo de Comunicação - gRPC

O gRPC foi escolhido por sua eficiência e suporte a múltiplas linguagens de programação. Ele permite a definição clara de serviços e mensagens usando arquivos `.proto`. Além disso, ele também facilita a escalabilidade horizontal do sistema, permitindo que múltiplas instâncias do servidor sejam executadas em diferentes máquinas para balancear a carga. Além disso, a configuração de gRPC com Python é relativamente simples e bem documentada, facilitando a implementação.

3.2 Estratégias de Escalabilidade

- Multithreading: O servidor gRPC utiliza um pool de threads para lidar com múltiplos pedidos simultaneamente. Isso permite que ele processe eventos de vários clientes ao mesmo tempo.
- Testes de Carga: Realizamos testes de carga para avaliar o desempenho do sistema sob diferentes condições de uso, analisando como o tempo de processamento varia com o aumento da demanda.

3.3 Métodos de Sincronização de Dados

Utilizamos DataFrame do Pandas, os eventos recebidos são convertidos em DataFrames para facilitar a análise e a agregação dos dados.

Como dito anteriormente também optamos por análises simples que incluem contagem de eventos por usuário, por componente alvo e por data. Essas métricas são suficientes para avaliar a capacidade do servidor em lidar com a carga de trabalho.

4 Apresentação e discussão dos resultados do teste de carga

Apresentar os dados coletados durante os testes, como tempos de resposta e taxas de transferência. Discutir o que os resultados indicam sobre o desempenho e a eficiência do sistema, e como eles validam ou refutam as hipóteses iniciais.

Realizamos o teste de carga para avaliar o desempenho do servidor gRPC sob diferentes condições de carga. Utilizamos um computador para executar o stub do servidor e dois computadores adicionais como clientes. Em cada cliente, o teste variou o número de threads de 1 a 20, repetindo cada configuração 100 vezes para obter médias consistentes.

Os resultados do primeiro computador de cliente é apresentado a seguir. Note que cada entrada representa a média do tempo de resposta para o número especificado de threads:

```
Testing with 1 threads...
Average response time: 0.193884 seconds
Testing with 2 threads...
Average response time: 0.201270 seconds
Testing with 3 threads...
Average response time: 0.212098 seconds
Testing with 4 threads...
Average response time: 0.201452 seconds
Testing with 5 threads...
Average response time: 0.212609 seconds
Testing with 6 threads...
Average response time: 0.232301 seconds
Testing with 7 threads...
Average response time: 0.238953 seconds
Testing with 8 threads...
Average response time: 0.247744 seconds
Testing with 9 threads...
Average response time: 0.252368 seconds
Testing with 10 threads...
Average response time: 0.260050 seconds
Testing with 11 threads...
Average response time: 0.262806 seconds
Testing with 12 threads...
Average response time: 0.299685 seconds
Testing with 13 threads...
Average response time: 0.302657 seconds
Testing with 14 threads...
Average response time: 0.318317 seconds
Testing with 15 threads...
Average response time: 0.365308 seconds
Testing with 16 threads...
Average response time: 0.384774 seconds
```

```
Testing with 17 threads...
Average response time: 0.371179 seconds
Testing with 18 threads...
Average response time: 0.414800 seconds
Testing with 19 threads...
Average response time: 0.452461 seconds
Testing with 20 threads...
Average response time: 0.535000 seconds
```

Para a segunda máquina, obtivemos:

```
Testing with 1 threads...
Average response time: 0.120307 seconds
Testing with 2 threads...
Average response time: 0.122910 seconds
Testing with 3 threads...
Average response time: 0.145190 seconds
Testing with 4 threads...
Average response time: 0.166429 seconds
Testing with 5 threads...
Average response time: 0.146546 seconds
Testing with 6 threads...
Average response time: 0.300395 seconds
Testing with 7 threads...
Average response time: 0.161952 seconds
Testing with 8 threads...
Average response time: 0.174814 seconds
Testing with 9 threads...
Average response time: 0.213144 seconds
Testing with 10 threads...
Average response time: 0.220375 seconds
Testing with 11 threads...
Average response time: 0.249823 seconds
Testing with 12 threads...
Average response time: 0.456511 seconds
Testing with 13 threads...
Average response time: 0.357194 seconds
Testing with 14 threads...
Average response time: 0.443484 seconds
Testing with 15 threads...
Average response time: 0.527570 seconds
Testing with 16 threads...
Average response time: 0.594968 seconds
Testing with 17 threads...
Average response time: 0.705434 seconds
Testing with 18 threads...
Average response time: 0.932867 seconds
Testing with 19 threads...
Average response time: 1.138010 seconds
```

```
Testing with 20 threads...
Average response time: 1.255955 seconds
```

A análise dos dados do processamento da sugere que à medida que o número de threads (e, por extensão, o número de solicitações simultâneas ao servidor) aumenta, o tempo médio de resposta também aumenta. Isso indica que a capacidade do servidor para processar solicitações simultâneas tem um limite, após o qual ocorre uma degradação no desempenho. Especificamente, observamos um aumento substancial no tempo de resposta da segunda máquina quando o número de threads excede 12, sugerindo que essa pode ser a capacidade limite de processamento simultâneo do servidor sob a configuração atual. Este fenômeno é típico de sistemas sob carga pesada, onde a sobrecarga de gerenciamento de múltiplas threads e a contenção de recursos levam a um aumento no tempo de espera.