Fundação Getulio Vargas
Escola de Matemática Aplicada
Curso de Ciência de Dados e Inteligência Artificial

Bruno Kauan Lunardon
Fabrício Dalvi Venturim
Luís Felipe de Almeida Marques
Otávio Augusto Matos Alves
Vinícius Antunes de Sousa

A1

Computação Escalável

Professor: Thiago Pinheiro de Araújo

Rio de Janeiro
2024

Modelagem

DataFrame

O `DataFrame` consiste em uma classe que guarda um hashmap que liga os nomes (strings) das colunas do dataframe às colunas, que consistem em
`std::vectors<std::variant<int, float, std::string, std::tm>>`,
de forma que as colunas podem assumir qualquer um desses tipos. A classe também guarda atributos auxiliares para sabermos a ordem da coluna, e a os tipos das colunas "reais das colunas", possibilitando sabermos a ordem em que linhas novas devem ser inseridas e para qual tipo podemos fazer a conversão das colunas.
A classe `DataFrame` contém vários métodos para manipulação dos dados, como adicionar colunas e linhas e também removê-las.

ConsumerProducerQueue

A classe `ConsumerProducerQueue` é uma implementação de uma fila "thread-safe"usando o padrão produtor consumidor para controlar o acesso a fila. Os elementos da fila são ponteiros para a abstração de DataFrame.

Handlers

Em nossa implementação, os `Handler` são represssentados por uma classe em que cada subclasse é um tratador específico. Exemplos dessas incluem `Filter`, `GroupBy` e `Sort`.

Cada `Handler` recebe uma `Queue` de entrada e uma `Queue` para saída.

Trigger

Para garantir o funcionamento automático da pipeline o usuário tem a sua disposição 3 triggers, um trigger que executa a pipeline a cada x segundos, um trigger que observa um diretório escolhido pelo usuário e verifica se tem algum arquivo `txt` ou `csv` novo no diretório, o último trigger verifica um diretório dado e verifica se algum arquivo foi alterado, esse trigger então manda/aplica um killswitch no pipeline ativo e cria um novo pipeline idêntico ao anterior, agora porém com os dados do `csv`, por exemplo, atualizado.

Concorrência

Pipeline

A pipeline é modelada como um Directed Acyclic Graph (DAG) onde cada vértice é um `Handler` e cada aresta é uma `ConsumerProducerQueue`. Todo vértice tem grau de entrada 1 e grau de saída 1 ou 0. Inicialmente tentamos adicionar a opção de que um `Handler` enviasse para mais de uma fila de saída, porém encontramos o seguinte problema, como os elementos das filas são ponteiros para `DataFrames` e os `Handlers` mudam esses `DataFrames` ao enviar um ponteiro para n filas diferentes o primeiro dos n futuros `Handlers` que acessar o dado contido em um ponteiro vai mudar esse dado para os demais `Handlers`, poderíamos contornar isso fazendo com que os `Handlers` trabalhassem com cópias dos dados mas isso seria ineficiente, consumindo tanto memória quanto adicionando

um overhead de execução ao programa. Decidimos limitar o número de filas de saída um para não encontrar esse problema. Uma possível expansão do trabalho seria modelar uma solução para esse problema.

Nossas pipelines foram modeladas para estarem sempre em execução e usamos um `Event Trigger` para adicionar novos dados quando eles estiverem disponíveis no momento que uma parcela dos dados alcance um vértice com grau de saída 0, ou seja, algum ponto final na pipeline ela é adiciona ao repositório em sua respectiva tabela.

Em cada etapa da pipeline, são alocados números fixos de "threads", sendo a leitura de um arquivo realizada por uma única "thread"enquanto os tratadores geralmente vão ser executados por múltiplas "threads". Toda a concorrência de acesso é resolvida pela classe das `Queues`

Arquivos

Para controlar o acesso simultâneo de arquivos, foi utilizado a função `flock`, de forma que várias "threads"podem ler o mesmo arquivo, mas para a escrita nos arquivos o escritor precisa de acesso exclusivo do arquivo. Em nossa implementação, o `flock` só é liberado quando o leitor ou escritor de arquivo termina de ler ou escrever o arquivo.

Para aumentar a eficiência e permitir um processamento paralelo mais eficaz nas etapas subsequentes da pipeline, além de não ficar preso em uma leitura infinita, implementamos a leitura e criação de blocos de DataFrame. No processo de leitura de arquivos, os dados são divididos em blocos de tamanho fixo ou conforme necessário, dependendo do contexto. Esses blocos de dados são então inseridos em uma fila de DataFrames, onde podem ser processados por tratadores em paralelo.

Essa abordagem de leitura e criação de blocos permite que múltiplos tratadores processem os dados simultaneamente, mesmo que a leitura da base de dados não esteja completa.

Mock Data

Para simular os arquivos de dados do serviço de compras utilizamos as dicas do enunciado, para os arquivos de `log` fizemos 4 tipos de arquivos de diferentes como especificado, porém para o *User Behavior* interpretamos o click do usuário em um botão que corresponde a um produto como uma visualização. Com a Conta Verde apenas implementamos o especificado, é de se notar o problema de consistência quando estamos executando o pipeline, se a tabela de estoque é mudada durante a execução não podemos só sobrescrever a anterior e seguir o processamento, por esse problema decidimos pela parada da pipeline e criação de outra idêntica que continua do mesmo lugar.