Benchmark de Ferramentas para Sistemas de Recomendação: Comparação de Desempenho entre CPU e GPU em Grandes Volumes de Dados Utilizando PySpark e Rapids

Diego A. Fagundes[1], Maurício Pfleger[2]

Pontificia Universidade Católica do Paraná, Curitiba, PR, Brasil

{diego.fagundes,mauricio.pfleger}@pucpr.edu.br, paul.jean@pucpr.br

Abstract. *This study compares the performance of CPU and GPU processing for big data-based recommendation systems. Empirical tests were performed using the MLlib and Rapids libraries of Python to compare the parallel processing framework called Spark, using the MLlib library, with the GPU processing engine using the Rapids library. Results showed that GPU processing outperformed CPU processing, with a significant reduction in runtime and more efficient use of processing resources. The study concludes that GPU processing is a promising approach for big data-based recommendation systems, particularly for applications requiring high-volume calculations and parallel processing.*

Resumo. *Este estudo compara o desempenho do processamento da CPU com o processamento em GPU para sistemas de recomendação baseados em big data. Foram realizados testes empíricos usando as bibliotecas MLlib e Rapids do Python para comparar o desempenho da estrutura de processamento paralelo denominada Spark, através da biblioteca MLlib, com o mecanismo de processamento em GPU, utilizando a biblioteca Rapids. Os resultados mostraram que o processamento em GPU apresentou um desempenho superior em relação ao processamento em CPU, com uma redução significativa no tempo de execução e uma utilização mais eficiente dos recursos de processamento. O estudo conclui que o processamento em GPU é uma abordagem promissora para sistemas de recomendação baseados em big data, especialmente em aplicações que exigem um alto volume de cálculos e processamento paralelo.*

I. INTRODUÇÃO

Nosso trabalho é sobre o benchmark de ferramentas para sistemas de recomendação baseados em GPU e Big Data. Um sistema de recomendação é uma técnica que permite prever as preferências do usuário em relação a determinado item, com base em seus dados de histórico de navegação, compras e outras informações relevantes. Já o ALS (Alternating Least Squares) é um algoritmo amplamente utilizado em sistemas de recomendação para realizar a fatoração de matrizes esparsas, com o objetivo de encontrar relações entre usuários e itens, e gerar recomendações personalizadas.

O problema que nosso projeto visa atacar é comparar o desempenho da estrutura de processamento de dados paralelo em CPU, com o mecanismo de processamento de dados em GPU. Existem diversas ferramentas disponíveis que possibilitam seguir entre

ambos os caminhos, por meio delas, nosso objetivo é comparar o desempenho da estrutura de processamento paralelo denominado Spark, através da biblioteca MLlib, com o mecanismo de processamento em GPU, utilizando a biblioteca Rapids, sendo ambas bibliotecas do Python.

A proposta do nosso artigo é apresentar uma análise comparativa dessas ferramentas, com o objetivo de identificar a mais adequada para o processamento de grandes volumes de dados em sistemas de recomendação. Para isso, realizamos testes utilizando o ALS em duas implementações distintas: uma utilizando a biblioteca do Spark, implementada em Scala, e outra utilizando a biblioteca do Rapids, implementada em C.

A contribuição do nosso trabalho está em apresentar uma análise comparativa entre as duas ferramentas para sistemas de recomendação, baseadas em CPU e GPU, respectivamente. Além disso, utilizamos uma base de dados comum em nossos testes, que foi obtida a partir de um artigo de um colega da área.

O restante do nosso artigo está distribuído da seguinte forma: Na seção II, é feita a declaração do problema e o objetivo específico do estudo. A seção III apresenta os trabalhos relacionados e a literatura existente sobre o tema, abordando estudos anteriores sobre sistemas de recomendação e ferramentas de processamento paralelo. Na seção IV, descrevemos a metodologia que será empregada no estudo comparativo, detalhando as etapas e procedimentos que serão realizados para avaliar o desempenho das ferramentas. Na seção V, apresentaremos os procedimentos experimentais que serão executados para avaliar as ferramentas de processamento paralelo. Por fim, na seção VI "Análises e Conclusão", discutiremos os resultados obtidos nos experimentos comparativos, apresentando as análises, conclusões e insights obtidos a partir dos resultados.

II. DECLARAÇÃO DO PROBLEMA

Este projeto visa comparar o processamento de grandes volumes de dados em sistemas de recomendação baseados em ALS, fazendo uso da base de dados SMDI-500E, que possuí um número máximo de 500 iterações por usuário. Quando temos aumento na quantidade de dados, torna-se essencial a utilização de ferramentas de processamento paralelo para melhorar a eficiência do processamento.

Embora existam diversas ferramentas disponíveis, não está claro qual é a mais adequada para sistemas de recomendação. Portanto, o objetivo deste trabalho é realizar uma análise comparativa entre as ferramentas de processamento paralelo para sistemas de recomendação baseados em ALS, utilizando a biblioteca MLlib para Spark CPU e a biblioteca Rapids para GPU.

O comparativo pode ajudar a escolher a ferramenta adequada e melhorar significativamente a eficiência do processamento de grandes volumes de dados em sistemas de recomendação. Portanto, este trabalho pode contribuir para a otimização de sistemas de recomendação e pode ser útil para desenvolvedores e pesquisadores que trabalham nesta área.

III.TRABALHOS RELACIONADOS

Existem diversas pesquisas na área de sistemas de recomendação baseados em ALS e ferramentas de processamento paralelo. Um dos estudos mais relevantes é o Large-scale

Parallel Collaborative Filtering for the Netflix Prize (http://shiftleft.com/mirrors/www.hpl.hp.com/personal/Robert_Schreiber/papers/2008%20AAIM%20Netflix/netflix_aaim08(submitted).pdf), visando melhorar a eficiência computacional. O estudo descreve em detalhes o algoritmo paralelo, bem como sua implementação em sistemas distribuídos. O trabalho também apresenta resultados experimentais detalhados com o conjunto de dados do Netflix Prize, mostrando a eficácia da abordagem proposta.

Em relação às ferramentas de processamento paralelo, a biblioteca Spark MLlib é uma das mais utilizadas para o processamento de grandes volumes de dados em sistemas de recomendação. A biblioteca Spark tem sido amplamente utilizada em aplicações de Big Data, como sistemas de recomendação, processamento de dados e análise de dados em tempo real (Zaharia et al., 2016) (https://www.jmlr.org/papers/volume17/15-237/15-237.pdf).

Recentemente, a biblioteca Rapids tem sido proposta como uma alternativa para o processamento de dados em GPU. A biblioteca Rapids fornece um conjunto de ferramentas que podem ser usadas para acelerar o processamento de dados em GPU, incluindo o processamento de matrizes esparsas, que é a base para o ALS, não encontramos uma referência gratuita que demonstre a utilização.

Embora haja muitos trabalhos sobre sistemas de recomendação e ferramentas de processamento paralelo, há poucos trabalhos comparativos entre as ferramentas Spark MLlib e Rapids para sistemas de recomendação baseados em ALS. Portanto, este trabalho tem como objetivo preencher essa lacuna e fornecer uma análise comparativa entre as duas ferramentas.

IV. PROPOSTA

O estudo comparará as bibliotecas MLib e Rapids para o processamento paralelo de dados, utilizando os mesmos parâmetros e o algoritmo ALS para recomendação em dois cenários: um em que os dados serão processados em uma CPU com Spark, e outro em que os dados serão processados em uma GPU com Rapids. O Dataset foi dividido em treino e teste.

Utilizamos o Google Colab para aplicação do estudo e fizemos uso do Google Drive para armazenar e manipular os Datasets, utilizando dentro do mesmo a raiz Dataset (/content/drive/MyDrive/Dataset/).

Para facilitar a compreensão, esta sessão será dividida em duas partes: a primeira irá explicar o pseudo-código utilizando o Spark e a segunda utilizando o Merlin.

a. Spark

Os pseudocódigos apresentados neste tópico referem-se ao SparkALS[1].

No algoritmo 4 - Sessão Spark, estamos instanciando a sessão do spark.

Algoritmo 5 – Preliminaries, as variáveis COL_USER, COL_ITEM, COL_RATING, COL_TIMESTAMP e COL_PREDICTION são usadas para definir os nomes das colunas da tabela. A constante TOP_K é usada mais adiante no código para definir o número de itens recomendados para cada usuário.

No Spark, esse código está definindo o esquema (schema) da tabela fato que será usada na implementação do modelo de recomendação. O esquema define o formato das colunas da tabela e é necessário para carregar os dados em um DataFrame do Spark.

O esquema da tabela é definido como um objeto da classe StructType. Cada coluna da tabela é definida como um objeto da classe StructField. No exemplo apresentado, o esquema possui três colunas: COL_USER do tipo IntegerType, que representa o ID do usuário; COL_ITEM do tipo IntegerType, que representa o ID do item a ser recomendado; e COL_RATING do tipo IntegerType, que representa a avaliação dada pelo usuário para o item.

Por fim, o esquema é utilizado na seção seguinte para carregar os dados do conjunto de dados MovieLens em um DataFrame do Spark.

Algoritmo 6 – Dataset, carregamos um Dataframe como Dataset.

No Algoritmo 7 – Treinando (SparkALS²), o código cria uma instância do modelo ALS (Alternating Least Squares) do Spark ML e treina o modelo com os dados de treinamento.

As opções de configuração do modelo ALS são passadas como parâmetros para a classe ALS:

userCol: o nome da coluna que contém os IDs dos usuários nos dados de entrada;

itemCol: o nome da coluna que contém os IDs dos itens nos dados de entrada;

ratingCol: o nome da coluna que contém as avaliações dos usuários para os itens nos dados de entrada;

coldStartStrategy: como lidar com novos usuários e itens durante a previsão (no caso, "drop" significa que esses dados serão simplesmente ignorados);

regParam: o valor da regularização a ser aplicado durante o treinamento do modelo (para evitar overfitting);

maxIter: o número máximo de iterações permitido durante o treinamento do modelo;

rank: o número de fatores latentes a serem usados para a fatoração de matriz;

seed: a semente usada para a inicialização aleatória do modelo;

nonnegative: se deve ou não ser permitido que as previsões do modelo sejam negativas (no caso, True significa que serão apenas previsões positivas);

Em seguida, o método fit() é chamado na instância do modelo ALS, passando os dados de treinamento como parâmetro. O método fit() executa o treinamento do modelo ALS nos dados de entrada e retorna o modelo treinado.

O trecho de código também usa um objeto Timer() para medir o tempo de treinamento do modelo ALS. O tempo de treinamento é impresso na tela com a mensagem "Took {} seconds for training.".

No Algoritmo 8 – Avaliando, o código implementa um modelo de recomendação usando o algoritmo ALS do Apache Spark. Ele avalia o desempenho do modelo utilizando diferentes métricas de ranking, como MAP, NDCG, Precision@K e Recall@K, e as

imprime na tela. O modelo é treinado em um conjunto de treinamento e é usado para fazer previsões em um conjunto de teste. Em seguida, o conjunto de previsões do modelo é filtrado para manter apenas as previsões para itens que ainda não foram avaliados. O desempenho do modelo é avaliado com base nas previsões para esses itens não avaliados.

No Algoritmo 8.1 - Teste usuário 17, verificamos a recomendação de 10 itens para o usuário 17.

b. Rapids (Merlin)

Os pseudocódigos apresentados neste tópico referem-se ao MerlinALS[2].

Para este comparativo, foi necessário sobrescrever o método evaluate para a biblioteca Rapids, pois ela não apresentava a métrica recall@k, conforme Algoritmo 3 - Método com recall.

Algoritmo 5 – Preliminaries, As variáveis COL_USER, COL_ITEM, COL_RATING, COL_TIMESTAMP e COL_PREDICTION são usadas para definir os nomes das colunas da tabela. A constante TOP_K é usada mais adiante no código para definir o número de itens recomendados para cada usuário.

A variável, INPUT_DATA_DIR, é definida como o diretório onde se encontra o conjunto de dados MovieLens que será utilizado na implementação do modelo. Especificamente, este é o diretório no Google Drive onde o conjunto de dados é armazenado no exemplo apresentado.

Por fim, a função get_lib() é chamada para carregar uma biblioteca personalizada (lib.py) que contém algumas funções auxiliares para a implementação do modelo de recomendação

Algoritmo 6 – Dataset, realizamos a transformação de dados para um modelo de aprendizado de máquina.

No Algoritmo 7 – Treinando, cria-se uma instância do modelo AlternatingLeastSquaresWithRecall, que é uma implementação do algoritmo ALS (Alternating Least Squares) com uma variante que leva em conta a precisão de recall para o cálculo das recomendações. Os parâmetros passados para a criação do objeto são:

regularization: parâmetro de regularização para evitar overfitting;

iterations: número de iterações do algoritmo;

factors: número de fatores latentes para a decomposição da matriz;

use_gpu: indica se o algoritmo deve ser executado na GPU (caso haja uma disponível);

Por fim, treina o modelo utilizando o conjunto de treinamento fornecido.

No Algoritmo 8 – Avaliando, o código em questão implementa o algoritmo ALS para recomendação de itens. Realizando a avaliação do modelo ALS utilizando as métricas "precision@k", "map@k", "ndcg@k", "auc@k" e "recall@k" e geração de predições para o conjunto de teste. Desta forma o código mede a capacidade do modelo de recomendar itens com base nas interações passadas do usuário com esses itens,

enquanto as predições geram uma matriz esparsa com as pontuações de previsão para cada usuário-item, que podem ser usadas para recomendar itens com base nessas pontuações.

No algoritmo 8.1 - Teste usuário 17, verificamos a recomendação de 10 itens para o usuário 17.

SparkALS[1]
(https://github.com/MauricioPfleger/Recommendation/blob/main/SparkALS.ipynb)

MerlinALS[2]
(https://github.com/MauricioPfleger/Recommendation/blob/main/MelrinALS.ipynb)

V. PROTOCOLO EXPERIMENTAL

Na realização do experimento, foi utilizado o dataset SMDI-500E, que consiste em registros de compras da Himarket, uma empresa localizada em Curitiba, Paraná, Brasil. O conjunto de dados foi previamente balanceado e limitado a no máximo 500 iterações por usuário. Dados duplicados foram removidos e o conjunto de dados foi dividido em treino e teste com base na divisão temporal. Compras realizadas em agosto e setembro de 2019 foram utilizadas como conjunto de treinamento, enquanto as compras de outubro e novembro de 2019 foram utilizadas como conjunto de teste. O objetivo do experimento foi avaliar a capacidade dos modelos de recomendar itens que os usuários viriam a comprar.

O modelo ALS (Alternating Least Squares) foi utilizado para realizar a análise. O ALS é um algoritmo de fatoração de matriz que é comumente usado para prever classificações ausentes em sistemas de recomendação. O modelo foi aplicado ao conjunto de dados SMDI-500E da Himarket utilizando a implementação do PySpark e do Merlin.

Para lidar com novos usuários ou itens sem classificações durante a previsão, foi utilizada uma estratégia de remoção desses usuários ou itens. Foi usado um parâmetro de regularização de 0,01 para evitar overfitting, um número máximo de iterações para executar o algoritmo ALS de 15 e um número de fatores latentes no modelo de 128. Esses parâmetros foram escolhidos devido às limitações da implementação do Merlin.

A fim de comparar as recomendações geradas pelo PySpark e pelo Merlin, foram geradas recomendações para o mesmo usuário em ambas as aplicações.

Para avaliar os modelos, foram utilizadas as métricas MAP at k (Mean Average Precision at k), NDCG at k (Normalized Discounted Cumulative Gain at k), Precision at k e Recall at k. A métrica MAP at k mede a precisão média das recomendações até a posição k na lista, enquanto NDCG at k mede a qualidade das recomendações até a posição k na lista, levando em conta a ordem das recomendações.

VI. ANÁLISES

A comparação entre os modelos ALS implementados com PySpark e Rapids revelou diferenças significativas nos resultados das métricas de avaliação. Enquanto o modelo

treinado com PySpark apresentou MAP, NDCG, Precision@K e Recall@K muito baixos, o modelo implementado com Rapids apresentou resultados consideravelmente melhores.

Tabela 1. Métricas de cada modelo

Framework	PySpark	Rapids
Top K	10	10
MAP@K	0,01%	5,10%
NDCG@K	0,07%	11,30%
Precision@K	0,07%	12,10%
Recall@K	0,03%	3,30%
Tempo de Treinamento (segundos)	216	10
Tempo de Avaliação (minutos)	41	0,02

Como observado na tabela 1, o primeiro modelo foi treinado utilizando o PySpark e apresentou resultados insatisfatórios, com valores muito baixos para as métricas de avaliação MAP, NDCG, Precision e Recall. Além disso, o tempo de treino foi bastante elevado, levando 216 segundos para ser concluído, enquanto o tempo de avaliação foi ainda maior, levando 41 minutos.

Já o segundo modelo, treinado utilizando o framework Rapids, apresentou resultados significativamente melhores para todas as métricas de avaliação. Os valores para MAP, NDCG, Precision e Recall foram consideravelmente mais altos do que os obtidos pelo modelo treinado com PySpark. Além disso, o tempo de treino e avaliação foram bastante reduzidos, levando apenas 10 segundos para treinar o modelo e 5 segundos para avaliá-lo.

Para efeito de comparação, foi treinado um terceiro modelo utilizando a biblioteca Lenskit, sem técnicas de big data. Os resultados obtidos foram bastante próximos aos do modelo treinado com Rapids, apresentando valores razoáveis para as métricas de avaliação e tempos de treino e avaliação intermediários entre os modelos treinados com PySpark e Rapids.

Dessa forma, pode-se concluir que, para o problema de recomendação de itens utilizando o algoritmo ALS, o framework Rapids se mostrou bastante eficiente em relação ao PySpark, apresentando resultados significativamente melhores e tempos de treino e avaliação bastante reduzidos. Além disso, o modelo treinado com Lenskit apresentou resultados razoáveis, indicando que o algoritmo ALS pode ser aplicado com sucesso mesmo em frameworks mais simples, sem técnicas de big data.

VII.CONCLUSÃO

XXX

AGRADECIMENTOS

Agradeço ao professor Jean Paul por nos orientar na elaboração deste artigo e nos ensinar a escrever os códigos e revisá-los. Agradeço também ao nosso amigo Alcides Conte Neto, cientista de dados na Asia Shipping Transportes Internacionais, por nos ajudar na elaboração dos códigos. Este trabalho foi realizado como parte do TCC de conclusão da

pós-graduação em Ciência de Dados da Pontifícia Universidade Católica do Paraná. Além disso, agradeço a todos os órgãos de fomento e empresas que forneceram auxílio financeiro ou material para a realização deste trabalho.

REFERÊNCIAS

[1] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, DB Tsai, Manish Amde, Sean Owen, Doris Xin, Reynold Xin, Michael J. Franklin, Reza Zadeh, Matei Zaharia, Ameet Talwalkar, "MLlib: Machine Learning in Apache Spark", Journal of Machine Learning Research, vol. 17, (34):1−7, 2016.

[2] Yunhong Zhou, Dennis Wilkinson, Robert Schreiber and Rong Pan, "Large-scale Parallel Collaborative Filtering for the Netflix Prize," AAIM 2008, Springer-Verlag, LNCS Vol. 5034, pp. 337—348, 2008.