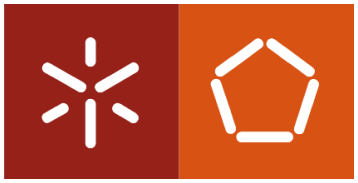

Universidade do Minho
Mestrado em Engenharia Informática
GGCD

Armazenamento e processamento de dados *Hadoop HDFS, Avro+Parquet e MapReduce*

Gonçalo Almeida - A84610
João Miguel Soares - A83581
Pedro Ribeiro - PG42848
Raimundo Barros - PG42814

José Orlando Roque Nascimento Pereira

Mestrado em Engenharia Informática
2021

Abstract

O presente projeto consiste no desenvolvimento de métodos para analisar dados armazenados em ficheiros provenientes do *dataset* público da plataforma IMDb e convertê-los num único ficheiro híbrido (binário e colunar) com um esquema apropriado utilizando AvroParquet. Para tal, foram utilizadas a linguagem de programação Java e a *framework* Hadoop, mais especificamente MapReduce, além de um ambiente de armazenamento distribuído HDFS. Com base nos dados armazenados no ficheiro foi possível calcular, para cada ano, o número total de filmes, o filme que recolheu mais votos e os 10 melhores filmes segundo a classificação. Numa fase de análise de resultados, observou-se que a solução implementada cumpriu todos os objetivos propostos.

Keywords: HDFS, AvroParquet, MapReduce.

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

O presente relatório é o resultado da resolução do primeiro trabalho prático da unidade curricular de Gestão de Grandes Conjuntos de Dados, do perfil de Ciência de Dados. O foco deste trabalho passou por implementar métodos capazes de converter dados armazenados em ficheiros tsv/zip em ficheiros híbridos colunar em formato binário com esquema apropriado. Para tal, teve-se por base a utilização das funcionalidades fornecidas pela linguagem de programação Java, bem como pela biblioteca Hadoop MapReduce e AvroParquet.

A linha de raciocínio para resolução dos problemas passou por utilizar algoritmos que forneciam uma forma eficiente de realizar a leitura dos *datasets* público disponíveis no site IMDb, o *title.basics.tsv.gz* (contém informações do filme, como por exemplo, título, ano de estreia, se é adulto, género e etc) e o *title.ratings.tsv.gz* (contém informações referentes a avaliação dos filmes), após a leitura dos ficheiros, estes posteriormente foram convertidos através de funções de MapReduce e AvroParquet em obtendo um único ficheiro.

O formato binário os ficheiros de textos são representados em binário (compacta e sem ambiguidade), este tipo de formato pode ser bastante eficiente nas operações de inserção, atualização e remoção, comumente este tipo de formato é utilizado em modelos transacionais relacionais. A utilizado do formato binário, em paralelo com o *layout* colunar híbrido permite organizar o ficheiro, dividindo este em segmentos sendo que cada segmento são compostos por linhas e organizando tais segmentos por colunas. Dessa forma permite armazenar e organizar o *layout* em memória até ter um tamanho suficiente, escreve este segmento para o disco. Este formato permite ler apenas os dados que é de interesse e facilita o processo de inserção, atualização e remoção de um único segmento ao invés de reescrever todo o ficheiro. O mesmo se aplica para a leitura dos segmentos.

Desse modo, com o ficheiro resultante do processo de MapReduce, foi possível realizar o cálculo para cada ano, o número total de filmes, o filme que recolheu mais votos e os 10 melhores filmes segundo a classificação.

No capítulos seguintes serão apresentados as etapas para implementação para a resolução das questões, além disso serão apresentados as configurações realizadas para rodar o programa em um ambiente distribuído, e as instruções que permitem executar o programa e fazer a leitura dos resultados.

Capítulo 2

Resolução das Tarefas

2.1 Carregamento dos Dados para AvroParquet

2.1.1 Descrição

A primeira tarefa consiste em carregar os dados dos ficheiros *title.basics.tsv.gz* e *title.ratings.tsv.gz* para um único ficheiro AvroParquet.

O desenvolvimento desta tarefa encontra-se na classe **ToParquet**.

2.1.2 Esquema

O esquema necessário para a realização desta tarefa deve conter, para cada filme, todas as informações fornecidas pelos dois ficheiros de dados. Assim sendo, foi utilizado o seguinte esquema.

Código 2.1: Esquema *movie_schema.parquet*

```
message Movie {
    required binary tconst (STRING);
    required binary titleType (STRING);
    required binary primaryTitle (STRING);
    required binary originalTitle (STRING);
    required binary isAdult (STRING);
    required binary startYear (STRING);
    required binary endYear (STRING);
    required binary runtimeMinutes (STRING);
    required group genres (LIST) {
        repeated binary genre (STRING);
    }
    required double averageRating;
    required int32 numVotes;
}
```

2.1.3 MapReduce

Para o processo de mapeamento da informação foram desenvolvidos dois *mapper's*, um para cada ficheiro. O primeiro *mapper* (BasicsMapper) toma como *input* o ficheiro de dados *title.basics.tsv.gz* e produz pares chave-valor, sendo o id do filme (tconst) a chave e uma *string* com os restantes dados o valor. Da mesma forma, o segundo *mapper* (RatingsMapper)

toma como *input* o ficheiro de dados *title.ratings.tsv.gz* e produz pares chave-valor com a mesma lógica. Como forma de distingir os *outputs* de cada *mapper*, no início de cada valor do par é anexada uma *substring* que identifica o *mapper*.

Foi desenvolvido um *reducer* (MyReducer) que, para cada chave, cria um registo correspondente ao esquema em 2.1 e nos seus campos são armazenadas as respetivas informações. Há que notar que este *reducer* consegue diferenciar as informações devido à distinção referida anteriormente. Por fim, utiliza o contexto do MapReduce para escrever cada registo.

2.2 Computação dos Dados por Ano

2.2.1 Descrição

A segunda tarefa consiste em utilizar o ficheiro AvroParquet resultante da tarefa anterior para calcular, para cada ano, o número total de filmes, o filme que recolheu mais votos e os 10 melhores filmes segundo a classificação. Estes resultados devem ser armazenados num ficheiro AvroParquet.

O desenvolvimento desta tarefa encontra-se na classe **ComputeYears**.

2.2.2 Esquemas

Sendo que não são necessárias todas as colunas que foram carregadas para o ficheiro AvroParquet da primeira tarefa, foi desenvolvido um esquema de projeção como forma de maximizar a performance, lendo apenas as colunas pretendidas.

Código 2.2: Esquema *projection_schema.parquet*

```
message Movie {
    required binary tconst (STRING);
    required binary titleType (STRING);
    required binary startYear (STRING);
    required double averageRating;
    required int32 numVotes;
}
```

Para armazenar os resultados pretendidos foi desenvolvido o seguinte esquema.

Código 2.3: Esquema *year_schema.parquet*

```
message Year {
    required int32 year;
    required int32 totalMovies;
    required binary mostVoted (STRING);
    required group topRated (LIST) {
        repeated binary tconst (STRING);
    }
}
```

2.2.3 MapReduce

Para esta tarefa, o *mapper* toma como input o ficheiro AvroParquet resultante da primeira tarefa, carregando os dados conforme o esquema em 2.2. Então, apenas para os filmes, produz pares chave-valor, sendo o ano de estreia a chave e uma *string* com os campos necessários para os cálculos pretendidos o valor.

O *reducer* guarda em memória, para cada chave (ano), o número total de filmes, o filme mais votado e uma lista com os 10 filmes melhor classificados que são atualizados para cada valor. Por fim, estas informações são armazenadas num registo correspondente ao esquema em 2.3 e é utilizado o contexto do MapReduce para o escrever.

2.2.4 Validações dos Resultados

Como forma de validar e confirmar os resultados obtidos foram desenvolvidas duas classes. A primeira, **FromParquet**, é semelhante à classe **ComputeYears** que foi explicada anteriormente, a diferença é que os resultados não são armazenados num ficheiro AvroParquet mas num ficheiro de texto para facilitar a leitura do mesmo. A segunda, **ValidateYears**, tem como objetivo verificar se os dados no ficheiro AvroParquet ficaram corretamente armazenados. Para isto, um *mapper* lê o próprio ficheiro e um *reducer* escreve a informação para um ficheiro de texto.

Deste modo, comparando os dois *outputs*, confirma-se se os resultados ficaram bem armazenados no formato AvroParquet.

Código 2.4: Resultados da classe **FromParquet** para o ano 2020

```
Number of movies: 14401

Movie with the most votes: tt6723592 (324184 votes)

Top 10 movies by average rating:

  > tt12372270 (average rating = 9.9)

  > tt12980094 (average rating = 9.8)

  > tt12593524 (average rating = 9.8)

  > tt13802560 (average rating = 9.8)

  > tt11933770 (average rating = 9.8)

  > tt11874886 (average rating = 9.8)

  > tt11782684 (average rating = 9.8)

  > tt13424150 (average rating = 9.7)

  > tt13545924 (average rating = 9.7)

  > tt13623860 (average rating = 9.7)
```

Código 2.5: Resultados da classe **ValidateYears** para o ano 2020

```
Number of movies: 14401

Movie with the most votes: tt6723592

Top 10 movies by average rating:

  > tt12372270

  > tt12980094

  > tt12593524

  > tt13802560

  > tt11933770

  > tt11874886

  > tt11782684

  > tt13424150

  > tt13545924

  > tt13623860
```

Capítulo 3

Hadoop HDFS

O Hadoop HFDS é utilizado para armazenar e analisar grandes quantidades de dados (estruturados e não estruturados). Num *cluster* Hadoop, os dados são armazenados e processados ao longo de diversos computadores de forma paralela.

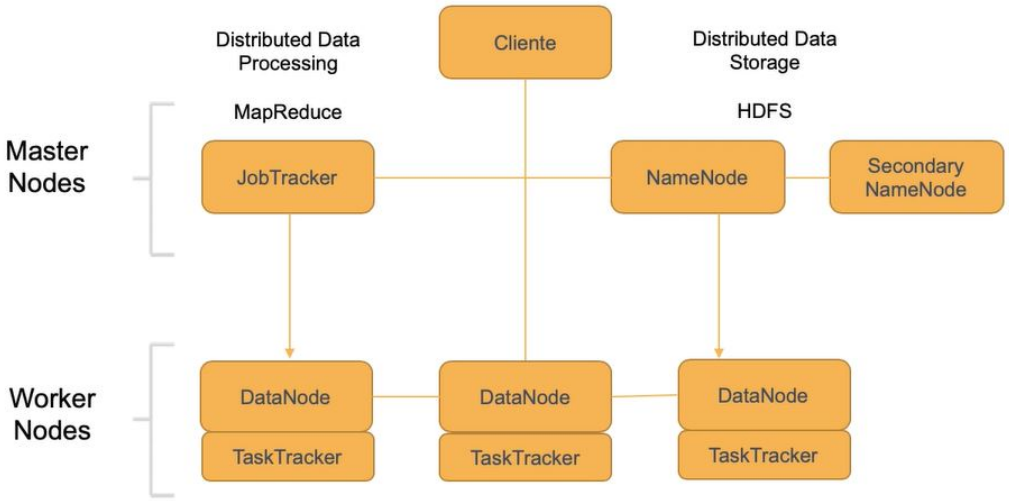

Figura 3.1: Arquitetura HDFS

Através do JAR é possível fazer a leitura e o armazenamento de dados, além de realizar o processamento de dados designados através dos *jobs*. O processamento distribuído de dados, conforme mencionado no capítulo anterior foi realizado através das funções de MapReduce, para o armazenamento distribuído utilizamos o ambiente HDFS, onde o *NameNode* é controlador principal (*master*) que mantém os metadados de todo o sistema de arquivos e permite nos conectar ao servidor. Os dados em geral serão armazenados no *DataNode*, no nosso projeto trabalhmos com duas arquiteturas, com 1 *datanode* por se tratar de uma única máquina, e 2 *datanodes* ao testar a solução na plataforma do Google Cloud (PaaS).

3.1 Docker-Hadoop

A criação do servidor HDFS foi realizado através de um *container* Docker.

Código 3.1: Repositório Git: Docker-Hadoop

```
$ git clone https://github.com/big-data-europe/docker-hadoop.git
```

O repositório acima traz consigo um servidor configurado. Ao executar o comando *docker-compose pull*, é descarregado todos os *nodes* e bibliotecas, e após o término executa-se o comando *docker-compose up* para dar início.

Para aceder ao *Namenode* e armazenar os ficheiros no Hadoop os seguintes comandos foram executados:

Código 3.2: Aceder ao namenode & Copiar ficheiros

```
$ docker exec -it namenode bash

$ docker run --env-file hadoop.env -v /home/aluno/IdeaProjects/App/:/data --network
    docker-hadoop_default -it bde2020/hadoop-base hdfs dfs -put
    data/movie_schema.parquet /
```

No software de desenvolvimento (Java IDE) um ficheiro Docker foi criado com os seguintes parâmetros:

Código 3.3: Dockerfile

```
FROM bde2020/hadoop-base
COPY target/app-1.0-SNAPSHOT.jar /
CMD [ "hadoop", "jar", "/app-1.0-SNAPSHOT.jar", "main" ]
```

O ficheiro JAR carrega consigo todas as bibliotecas, dependências, classes, métodos e funções necessários para que o *script* seja executado no ambiente HDFS, além de permitir fazer a leitura e o armazenamento dos ficheiros provenientes dos processos de MapReduce.

Além disso, para se conectar ao Hadoop HDFS, fez-se necessário adicionar os parâmetros de conexão do *docker-hadoop* através da IDE.

Código 3.4: Docker Run Options

```
Run options:
--env-file /home/aluno/docker-hadoop/hadoop.env -network docker-hadoop_default
```

Após realizar as configurações citadas acima, basta apenas executar o *Docker file* e todo o processo de MapReduce, AvroParquet e armazenamento de ficheiros se iniciam. Para tanto, é imprescindível configurar o ficheiro pom.xml (Maven), onde neste ficheiro se encontram todas as dependências e *plugins* necessários para que o JAR corra correctamente no ambiente Hadoop.

Como a versão do *docker-hadoop* possui uma versão antiga do Avro, se faz necessário realizar o *downgrade* das dependências do Parquet no pom.xml para a versão 1.7.0 (Código 3.5. Em adição, é importante nos ficheiros relacionados aos esquemas, alterar o formato definido de *STRING* para *UTF-8*.

Código 3.5: Pom.xml - Dependências para docker-hadoop

```
<dependencies>
    <dependency>
        <groupId>org.apache.hadoop</groupId>
        <artifactId>hadoop-client</artifactId>
        <version>3.2.1</version>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>org.apache.parquet</groupId>
        <artifactId>parquet-hadoop-bundle</artifactId>
        <version>1.7.0</version>
    </dependency>
    <dependency>
        <groupId>org.apache.parquet</groupId>
        <artifactId>parquet-avro</artifactId>
        <version>1.7.0</version>
    </dependency>
</dependencies>
```

Ao correr o ficheiro .jar através do *Dockerfile*, o *script* executa todas a etapas de *MapReduce* e conversões *AvroParquet* e o ficheiros de saída (*outputs*) são armazenados no servidor Hadoop (Fig. 3.2)

```
root@c0ce0b111a44:/# hdfs dfs -ls /
Found 12 items
drwxrwxrwt   - root root               0 2021-04-04 17:06 /app-logs
drwxr-xr-x   - root supergroup         0 2021-04-06 11:06 /compute_years_output
drwxr-xr-x   - root supergroup         0 2021-04-06 11:02 /from_parquet_output
-rw-r--r--   3 root supergroup       469 2021-04-06 10:54 /movie_schema.parquet
-rw-r--r--   3 root supergroup       192 2021-04-06 10:59 /projection_schema.parquet
drwxr-xr-x   - root supergroup         0 2021-04-01 18:39 /rmstate
-rw-r--r--   3 root supergroup     57704 2021-04-03 16:33 /title.basics.tsv
-rw-r--r--   3 root supergroup      1823 2021-04-03 16:34 /title.ratings.tsv
drwx------   - root supergroup         0 2021-04-04 14:16 /tmp
drwxr-xr-x   - root supergroup         0 2021-04-06 10:56 /to_parquet_output
drwxr-xr-x   - root supergroup         0 2021-04-06 11:07 /validate_years_output
-rw-r--r--   3 root supergroup       193 2021-04-06 10:59 /year_schema.parquet
```

Figura 3.2: Ficheiros outputs gerados após a execução do *script*

3.2 *Google Cloud - PaaS*

Experiências foram realizadas utilizando o *cluster* Hadoop no Google Cloud. O *cluster* é composto por um (1) *master* e dois (2) *workers* e o *cluster* Hadoop disponibilizado possui as últimas versões do Avro e Parquet.

Após a criação do *cluster* no ambiente *cloud*, é importante estabelecer uma conexão e para tanto essa conectividade se faz via protocolo de rede criptográfico (SSH), conforme apresentado no código abaixo.

Código 3.6: Aceder ao *Cluster* no Google Cloud

```
# SSH to connect into master node
gcloud compute ssh ggcd-m
```

Após aceder ao *master* (*namenode*) do *cluster* Hadoop, os ficheiros dos *datasets*, bem como os ficheiros relativos aos esquemas definido para a conversão dos ficheiros devem ser importados para o ambiente *cloud*.

Código 3.7: Import files

```
#Import datasets:
cat title.basics.tsv | gcloud compute ssh --zone=us-central1-f ggcd-m -- hdfs dfs
    -put - /title.basics.tsv

cat title.ratings.tsv | gcloud compute ssh --zone=us-central1-f ggcd-m -- hdfs dfs
    -put - /title.ratings.tsv

#import schemas:
cat movie_schema.parquet | gcloud compute ssh --zone=us-central1-f ggcd-m -- hdfs
    dfs -put - /movie_schema.parquet

cat projection_.parquet | gcloud compute ssh --zone=us-central1-f ggcd-m -- hdfs
    dfs -put - /projection_schema.parquet

cat year_schema.parquet | gcloud compute ssh --zone=us-central1-f ggcd-m -- hdfs
    dfs -put - /year_schema.parquet
```

Com os ficheiros necessários para rodar o programa armazenados na cloud, a próxima etapa é executar o job através do ficheiro .jar e informar a classe que será executada. Importante também informar o nome do cluster e a região onde o mesmo se encontra.

Código 3.8: Submit a Job

```
#Run .jar in hadoop environment on PaaS
gcloud dataproc jobs submit hadoop --jars=target/app-1.0-SNAPSHOT.jar --class=Main
    --cluster=ggcd --region=us-central1
```

O resultado do job é devolvido pelo job e pode-se observar através da imagem 3.3

```
}aluno@ggcd-m:~$hdfs dfs -ls /
Found 11 items
drwxr-xr-x   - root  hadoop          0 2021-04-08 22:49 /compute_years_output
drwxr-xr-x   - root  hadoop          0 2021-04-08 22:48 /from_parquet_output
-rw-r--r--   2 aluno hadoop        487 2021-04-08 22:42 /movie_schema.parquet
-rw-r--r--   2 aluno hadoop        198 2021-04-08 22:43 /projection_schema.parquet
-rw-r--r--   2 aluno hadoop  665312188 2021-04-08 22:15 /title.basics.tsv
-rw-r--r--   2 aluno hadoop   19432808 2021-04-08 22:17 /title.ratings.tsv
drwxrwxrwt   - hdfs  hadoop          0 2021-04-06 19:23 /tmp
drwxr-xr-x   - root  hadoop          0 2021-04-08 22:48 /to_parquet_output
drwxrwxrwt   - hdfs  hadoop          0 2021-04-06 19:23 /user
drwxr-xr-x   - root  hadoop          0 2021-04-08 22:50 /validate_years_output
-rw-r--r--   2 aluno hadoop        197 2021-04-08 22:44 /year_schema.parquet
```

Figura 3.3: Ficheiros outputs gerados após a execução do *script*

Capítulo 4

Conclusões

O objetivo deste trabalho prático consistiu no aumento da experiência dos alunos no âmbito do desenvolvimento de aplicações para trabalhar com grandes quantidades de dados utilizando diversos ficheiros. Além disso, o projeto visava a consolidação da utilização de processos de MapReduce juntamente com formato de ficheiro binário + híbrido colunar AvroParquet com esquema definido num ambiente distribuído (Hadoop HDFS), realçando a utilidade destas ferramentas para a resolução de problemas. O maior desafio encontrado foi, principalmente, na execução do ficheiro JAR no ambiente Hadoop, onde existiram algumas dificuldades na leitura dos esquemas e nas conversões destes para o formato binário (Avro). Como lições aprendidas e melhorias futuras, recomenda-se a prática e a execução dos scripts no ambiente HDFS pois, na máquina local, o *script* correu sem maiores problemas em resposta as perguntas da questão 2.