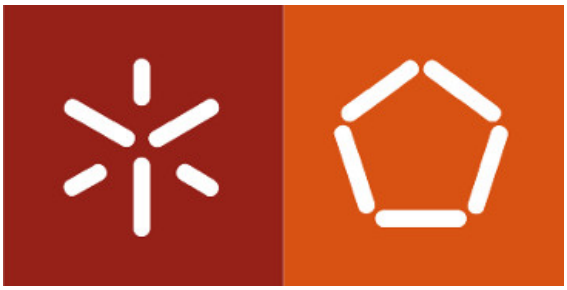

Universidade do Minho

Mestrado Integrado em Engenharia Informática

Ciências de Dados

Gestão de Grandes Conjuntos de Dados

IMDB

Bruno Veloso, A78352
Carolina Cunha, A80142
Diogo Tavares, PG42826
Hugo Nogueira, A81898
Luís Abreu, A82888

8 de abril de 2021

Conteúdo

Lista de Figuras

Lista de Excertos

Capítulo 1

Introdução

Com a constante evolução da tecnologia ocorre inevitavelmente o crescimento dos conjuntos de dados utilizados. Com este crescimento está associada a necessidade de tratamento e manipulação dos dados através de processos eficientes e fiáveis. Neste âmbito, surge a Unidade Curricular de Gestão de Grandes Conjuntos de Dados, cujo objetivo é estudar a pilha atual de gestão de dados, desde o armazenamento distribuído, passando pelos motores de processamento e incluindo as interfaces do utilizador com visualização interativa de resultados.

Capítulo 2

Descrição do Projeto

Este trabalho prático consistiu na concretização e avaliação experimental de tarefas de armazenamento e processamento de dados, através do desenvolvimento de métodos e classes que estendam as *interfaces* de *Map Reduce* e *Avro + Parquet* existentes na *framework Apache Hadoop*, permitindo responder às questões colocadas. De forma a responder a estas questões, foi utilizado o *dataset* público do IMDB [1].

2.1 Especificação dos Requisitos

De forma a tirar proveito do *Map Reduce*, *Hadoop HDFS* e *AvroParquet*, foram propostas as seguintes tarefas de processamento:

1. Devem ser carregados os dados dos ficheiros *title.basics.tsv.gz* e *title.ratings.tsv.gz* para um único ficheiro *AvroParquet*, com um esquema apropriado;
2. Considerando apenas filmes (*movies*), deve ser calculado, para cada ano, o número total de filmes; o filme que recolheu mais votos e os 10 melhores filmes segundo a classificação. Os resultados devem ser armazenados num único ficheiro *AvroParquet*.
3. Para cada filme, deve ser recomendado o filme do mesmo género que tenha a melhor classificação. Deve ser considerado apenas o primeiro género de cada filme.

[1] https://www.imdb.com/interfaces/

2.2 Conceção da Resolução

2.2.1 Tarefa 1

O primeiro passo consistiu na leitura dos dados dos ficheiros para um ficheiro *AvroParquet*. Para este efeito, definiu-se um esquema hierárquico para os dados pertencentes a ambos os ficheiros.

```
message Movies {
    required binary tconst (UTF8);
    required binary type (UTF8);
    required binary primaryTitle (UTF8);
    required binary originalTitle (UTF8);
    required binary isAdult (UTF8);
    required binary startYear (UTF8);
    required binary endYear (UTF8);
    required binary time (UTF8);
    required binary rating (UTF8);
    required binary votes (UTF8);
    required group genres (LIST) {
            repeated binary str (UTF8);
    }
}
```

Excerto 2.1: Esquema hierárquico

Neste esquema, foram assumidos todos os valores como *UTF8*, pelo que a sua omissão (ou o valor "\N") será substituída pelo valor nulo (*"null"*).

Posto isto, implementaram-se os métodos necessários para o armazenamento dos dados num ficheiro *AvroParquet*. O método *getSchema* é responsável pelo carregamento do ficheiro do esquema.

```
public static Schema getSchema() throws IOException {
    FileSystem fs = FileSystem.get(newConfiguration());
    FSDataInputStream s = fs.open(new Path("hdfs:///schema.movies"));
    byte[] buf = new byte[10000];

    s.read(buf);

    String ps = new String(buf);
    MessageType mt = MessageTypeParser.parseMessageType(ps);
    return new AvroSchemaConverter().convert(mt);
}
```

Excerto 2.2: Carregamento do esquema

Através da leitura do ficheiro *titles.basics.tsv.gz*, foram obtidos os dados de todos os títulos presentes no *dataset* público do IMDB. Neste ficheiro, observaram-se as colunas *tconst*, *titleType*, *primaryTitle*, *originalTitle*, *isAdult*, *startYear*, *endYear*, *runtimeMinutes* e *genres*. Estas encontram-se separadas por *tabs* (visto tratar-se de um ficheiro *tab-separated values*).

De forma a guardar estes dados num ficheiro, foi implementado um *LeftMapper*, onde são lidos os múltiplos campos e inseridos num *StringBuilder*. É de realçar a necessidade de ignorar a primeira linha do ficheiro (cabeçalho), identificado quando a chave é zero. Em caso de omissão dos dados relativos ao ano, duração e géneros, definiu-se o seu valor com a *string "null"*.

```
@Override
protected void map(LongWritable key, Text value, Context context) throws IOException,
    InterruptedException {
    //ignorar a primeira linha -> header
    if (key.get() == 0) return;

    String[] fields = value.toString().split("\t");
    StringBuilder values = new StringBuilder();
```

```
    //guardar titleType
    values.append(fields[1]);
    values.append("\t");

    //guardar primaryTitle
    values.append(fields[2]);
    values.append("\t");

    //guardar originalTitle
    values.append(fields[3]);
    values.append("\t");

    //guardar isAdult
    values.append(fields[4]);
    values.append("\t");

    //guardar startYear; se \N passa a "null"
    if(!fields[5].equals("\\N")) values.append(fields[5]);
    else values.append("null");
    values.append("\t");

    //guardar endYear; se \N passa a "null"
    if(!fields[6].equals("\\N")) values.append(fields[6]);
    else values.append("null");
    values.append("\t");

    //guardar runtimeMinutes; se \N passa a "null"
    if(!fields[7].equals("\\N")) values.append(fields[7]);
    else values.append("null");
    values.append("\t");

    //guardar genres; se \N passa a "null"
    int i = 0;

    for(String s : fields[8].split(",")){
        if(i!=0) values.append(",");
        if(!s.equals("\\N"))values.append(s);
        else values.append("null");
        i++;
    }
    values.append("\t");
    context.write(new Text(fields[0]), new Text(values.toString()));
}
```

Excerto 2.3: Left Mapper - map

No que diz respeito ao ficheiro *titles.ratings.tsv.gz*, a sua análise permite observar três colunas distintas: *tconst*, *averageRating* e *numVotes*. Similarmente ao ficheiro anteriormente analisado, foi necessário ignorar a primeira linha do ficheiro, correspondente ao cabeçalho.

Para o processamento destes dados, procedeu-se, inicialmente, à inserção do caractere *"R"* no início de cada linha, de forma a identificar os dados relativos a este ficheiro. Esta abordagem permite que o *Reducer* identifique e adicione estes dados unicamente após a *string (Text)* proveniente do *Left Mapper* (*ToParquetMapperLeft*).

```
@Override
protected void map(LongWritable key, Text value, Context context) throws IOException,
    InterruptedException {
    //ignorar a primeira linha -> header
    if (key.get() == 0) return;

    String[] fields = value.toString().split("\t");
    //primeira sub-string: rating || segunda sub-string: votes
```

```
    StringBuilder values = new StringBuilder();

    // Tag para os ratings
    values.append("R");
    values.append("\t");
    values.append(fields[1]);
    values.append("\t");
    values.append(fields[2]);

    context.write(new Text(fields[0]), new Text(values.toString()));
}
```

Excerto 2.4: Right Mapper - map

No excerto 2.5, verifica-se o *override* do método *setup*, permitindo o carregamento do ficheiro da estrutura hierárquica durante o processo de *reduce*.

```
@Override
protected void setup(Context context) throws IOException, InterruptedException {
    schema = getSchema();
}
```

Excerto 2.5: Join Reducer - setup

No *Reducer*, são agrupados os conteúdos de ambos os *Mappers* num só ficheiro *AvroParquet*. Para este efeito, caso uma linha contenha a entrada do primeiro ficheiro (*title.basics.tsv*), mas não do segundo (*title.ratings.tsv*), são criados os parâmetros *rating* e *votes* com valor nulo. Por outro lado, se só estiverem presentes dados relativos ao segundo ficheiro, a linha é descartada na sua totalidade, dada a impossibilidade de saber se se tratava de uma entrada de um filme (*movie*, crucial para dar resposta à segunda tarefa proposta). Este método resulta na inserção dos dados num *GenericRecord*, que é posteriormente escrito no ficheiro *AvroParquet*, seguindo a estrutura hierárquica fornecida.

```
@Override
protected void reduce(Text key, Iterable<Text> values, Context context) throws IOException,
    InterruptedException {

    boolean has_basics = false;
    boolean has_ratings = false;

    GenericRecord record = new GenericData.Record(schema);

    //criadas 2 strings para manter a ordem desejada
    String[] basics;
    String[] ratings;

    record.put("tconst",key);
    List<String> genres = new ArrayList<>();

    for(Text value : values){
        String[] aux = value.toString().split("\t");

        //se nao tiver R no inicio, entao estamos perante o value de basics
        if(!aux[0].equals("R")){
            basics = aux;
            record.put("type",basics[0]);
            record.put("primaryTitle",basics[1]);
            record.put("originalTitle",basics[2]);
            record.put("isAdult",basics[3]);
            record.put("startYear",basics[4]);
            record.put("endYear",basics[5]);
            record.put("time",basics[6]);

            String[] aux_gen = basics[7].split(",");
```

```
            for(String s : aux_gen) genres.add(s);
            record.put("genres",genres);

            has_basics = true;
        }
        else if(aux[0].equals("R")){
            ratings = aux;
            record.put("rating", ratings[0]);
            record.put("votes", ratings[1]);

            has_ratings = true;
        }
    }

    if(has_basics && has_ratings){
        context.write(null,record);
    }
    else if(has_basics && has_ratings == false){
            record.put("rating", "null");
            record.put("votes", "null");

            context.write(null,record);
    }
    else if(has_ratings && has_basics == false) return;
}
```

Excerto 2.6: Join Reducer - reduce

2.2.2 Tarefa 2

Numa segunda fase, foi requerido o cálculo, para cada ano, de algumas interrogações. Para esse efeito, foi utilizado o ficheiro *AvroParquet* previamente gerado. Foi aplicada, sobre este ficheiro, uma projeção, visto não ser necessário processar todos parâmetros dos dados para dar resposta a esta segunda tarefa.

```
job.setInputFormatClass(AvroParquetInputFormat.class);
AvroParquetInputFormat.addInputPath(job,new Path("hdfs:///resultado_parquet"));
Schema queries = getSchema("hdfs:///schema.queries");
Schema result = getSchema("hdfs:///schema.alinea2");
AvroParquetInputFormat.setRequestedProjection(job, queries);
```

Excerto 2.7: Excerto do método main - Tarefa 2

Esta projeção segue o esquema hierárquico que se apresenta em seguida.

```
message Queries {
    required binary tconst (UTF8);
    required binary type (UTF8);
    required binary originalTitle (UTF8);
    required binary startYear (UTF8);
    required binary rating (UTF8);
    required binary votes (UTF8);
}
```

Excerto 2.8: Esquema hierárquico da projeção - Tarefa 2

Posto isto, foi implementado um *Mapper*, que recebe como parâmetros um *GenericRecord*, e que verifica se a entrada do ficheiro diz respeito a um filme (*movie*). Caso se verifique, as entradas vão ser retornadas no formato

startYear, tconst	originalTitle	rating	numVotes	

em que *startYear* corresponde à chave do *map*. Para todas as entradas cujos valores de classificação e/ou número de votos sejam nulos, estes serão substituídos pelo valor -1.

```
@Override
    protected void map(Void key, GenericRecord value, Context context) throws IOException,
        InterruptedException {

        if (!value.get("type").equals("movie")) return;

        String tconst = value.get("tconst").toString();
        String votes = value.get("votes").toString();
        String rating = value.get("rating").toString();

        if(!rating.equals("null") && !votes.equals("null"))
            context.write(new Text(value.get("startYear").toString()),
                    new Text(tconst +"\t" + value.get("originalTitle").toString() + "\t" +
                        rating + "\t" +votes));
        else if(!rating.equals("null") && votes.equals("null"))
            context.write(new Text(value.get("startYear").toString()),
                    new Text(tconst +"\t" + value.get("originalTitle").toString() + "\t" +
                        rating + "\t" + "-1"));
        else if(rating.equals("null") && !votes.equals("null"))
            context.write(new Text(value.get("startYear").toString()),
                    new Text(tconst +"\t" + value.get("originalTitle").toString() + "\t" +
                        "-1" + "\t" + votes));
        else if(rating.equals("null") && votes.equals("null"))
            context.write(new Text(value.get("startYear").toString()),
                    new Text(tconst +"\t" + value.get("originalTitle").toString() + "\t" +
                        "-1" + "\t" + "-1"));
}
```

Excerto 2.9: Método map da classe *Mapper*- Tarefa 2

Finalmente, para a implementação do *Reducer*, foi sobreposto o método *setup*, onde é carregado o esquema hierárquico que se segue. Este esquema representa a estrutura do ficheiro *AvroParquet* com o resultado das interrogações.

```
message Alinea2 {
    required binary year (UTF8);
    required binary number_of_movies (UTF8);
    required binary tconst_most_votes (UTF8);
    required binary title_most_votes (UTF8);
    required binary number_of_votes (UTF8);
    required group top10 (LIST) {
                repeated binary str (UTF8);
    }
}
```

Excerto 2.10: Esquema hierárquico para o reduce - Tarefa 2

No método *reduce*, para cada chave, são iterados os valores correspondentes. Nesta iteração, é contabilizado o número total de filmes para cada ano; o filme que recolheu mais votos; e serão adicionadas a uma estrutura de dados (*top10*) todas as entradas iteradas.

```
long total_movies = 0;
long most_votes = -1;
int field_with_most_votes = -1;
String tconst_most_votes = "";
String title_most_votes = "";
GenericRecord record = new GenericData.Record(schema);

List<String> top10Year = new ArrayList<>();
for(Text s : values){
```

```
    top10Year.add(s.toString());
    // s = tconst + \t + title + \t + rating + \t + votes

    String[] fields = s.toString().split("\t");
    total_movies++;

    field_with_most_votes = Integer.parseInt(fields[3]);
    if (field_with_most_votes >= most_votes) {
        tconst_most_votes = fields[0];
        title_most_votes = fields[1];
        most_votes = field_with_most_votes;
    }
}
```

Excerto 2.11: Excerto do método reduce - Tarefa 2

Posteriormente, foi implementado um *Comparator*, ordenando por ordem decrescente, os elementos da lista *top10* por classificação e por número de votos, caso a classificação seja a mesma (figura 2.1).

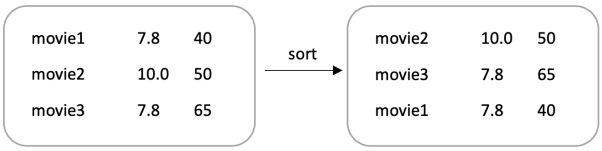

Figura 2.1: Método compare

De modo a obter a lista dos dez filmes com melhor classificação, os valores da lista são iterados. Uma nova lista é preenchida até que o seu tamanho seja igual a dez. De notar que são ignorados todos os filmes cuja classificação seja nula (com valor -1).

```
List<String> result = new ArrayList<>();

String ano = key.toString();
int count = 0;

for(String s : top10Year){
    //tconst title rating votes
    if(count == 10) break;

    String[] aux = s.split("\t");

    if(!aux[2].equals("-1")){
        result.add(aux[0] + "\t" + aux[1] + "\t" + aux[2] + "\t" + aux[3]);
        count++;
    }
}
```

Excerto 2.12: Excerto 2 do método reduce - Tarefa 2

2.2.3 Tarefa 3

Na terceira tarefa é pedido que, para cada filme, seja recomendado o filme com melhor classificação do mesmo género. Uma vez que se deve evitar carregar em memória simultaneamente todos os filmes do mesmo género, dividiu-se esta tarefa em duas fases distintas.

Na primeira fase, é escrito num ficheiro de texto os dois filmes com melhor classificação para cada género. Para este fim, foi utilizado o ficheiro *AvroParquet* gerado na primeira tarefa, sobre o qual foi aplicada a projeção do excerto 2.13.

```
message Alinea3 {
    required binary tconst (UTF8);
    required binary type (UTF8);
    required binary originalTitle (UTF8);
    required binary rating (UTF8);
    required binary votes (UTF8);
    required group genres (LIST) {
            repeated binary str (UTF8);
    }
}
```

Excerto 2.13: Esquema Hierárquico - Tarefa 3

Durante a etapa do *map* vai ser verificado, para cada entrada do ficheiro recebido, se se trata de um filme. Em seguida, vai ser guardado o primeiro género desse mesmo filme (se o género for *null*, significa que o filme não tem atribuído nenhum género e será descartado). Posteriormente, é criada uma *CompositeKeyWritableA3*, com a informação pretendida (Excerto 2.14). Se o número de votos for nulo, o seu valor é substituído por -1; caso a classificação seja nula, a entrada do filme é ignorada, uma vez que descarta a possibilidade de ser o filme com maior classificação do respetivo género.

```
public CompositeKeyWritableA3(String tconst, String originalTitle, String rating, String
    votes, String genero) {

    this.tconst = tconst;
    this.originalTitle = originalTitle;
    this.rating = rating;
    this.votes = votes;
    this.genero = genero;
}
```

Excerto 2.14: Construtor parameterizado da classe CompositeKeyWritableA3

Através do método *compareTo* definido na classe *CompositeKeyWritableA3*, as entradas foram ordenadas de acordo com a sua classificação e número de votos.

```
public int compareTo(CompositeKeyWritableA3 objKeyPair) {

    int result = genero.compareTo(objKeyPair.getGenero());
    if ( result == 0) {
        Double rate = Double.parseDouble(rating);
        Double rate2 = Double.parseDouble(objKeyPair.getRating());

        result = rate.compareTo(rate2);
        if(result == 0){
           Integer votes = Integer.parseInt(this.getVotes());
           Integer votes2 = Integer.parseInt(objKeyPair.getVotes());

           result = votes.compareTo(votes2);
        }
        return result*-1; // valor negativo para descendente, valor positivo para ascendente
            e valor 0 para igual
```

```
    }
    else return result; //se o genero nao for igual, quer-se ascendente


}
```

Excerto 2.15: Método compareTo - Tarefa 3

A classe *GroupingComparatorGenre* é responsável pela ordenação de *CompositeKeyWritableA3* por género. Se duas entradas pertencem ao mesmo género, serão agrupadas por género.

Quando no *Reducer*, são escritas no ficheiro as primeiras duas entradas de cada género, correspondentes ao Top 2 desse mesmo género.

A classe *PartitionerGenre* controla as chaves que cada *reducer* processa, garantindo que todos os dados com a mesma chave (género) são enviados para o mesmo *reducer*. Desta forma, garante-se uma ordenação fiável dos valores obtidos por cada *mapper*.

A segunda fase desta tarefa corresponde à escrita num ficheiro de texto da recomendação de um filme com maior classificação do mesmo género de cada entrada. Para este fim, é colocado em cache o ficheiro resultante da primeira fase desta tarefa. É, posteriormente, realizada uma projeção sobre o ficheiro resultante da primeira tarefa (Excerto 2.13).

Na etapa do *Mapper*, o método *setup* lê o ficheiro guardado em cache e, para cada género, as entradas são armazenadas num *HashMap* sob o formato

genre0, tconst	originalTitle	rating	votes
genre1, tconst	originalTitle	rating	votes

No método *map*, vai ser analisado o género de cada filme e comparado com as chaves existentes no *HashMap*. Se o filme recomendado é o mesmo que a entrada em análise, será recomendado o segundo filme com melhor classificação desse mesmo género.

Na classe do *Reducer*, os *values* serão escritos para um ficheiro de texto.

Capítulo 3

Análise de Métricas

De forma a observar em que medida os algoritmos implementados e os parâmetros de configuração estão otimizados, foram comparadas as medidas alternativas existentes nas diferentes tarefas realizadas. Para as diferentes tarefas, assume-se a utilização dos ficheiros em formato ***gzip***, **um *reducer*** e **com projeção**, salvo indicação contrária.

3.1 Tarefa 1

Na figura 3.1, observa-se que o carregamento dos dados é sensivelmente mais rápido quando os ficheiros dos dados não se encontram no formato *gzip*. Esta diferença deve-se à possibilidade de executar paralelamente vários passos dos *jobs*, reduzindo a carga de trabalho em cada *core*.

Alínea 1		
	***gzipped* (ms)**	***unzipped* (ms)**
Local	138642	112881
Docker-machine	178283	156976

Figura 3.1: Análise de Métricas - Tarefa 1

3.2 Tarefa 2

Para a segunda tarefa, foi importante analisar de que forma a projeção aplicada influencia o tempo de resposta da interrogação. Uma projeção (π) é um operador unário que produz a mesma relação apenas com as colunas especificadas (ou projetadas), na ordem especificada [1]. Assim sendo, é possível carregar apenas as colunas indispensáveis para a resolução desta tarefa. Por este motivo, é expectável que o tempo de execução desta interrogação seja consideravelmente menor quando se aplica a projeção, como se verifica na figura 3.2.

Alínea 2		
	Com projeção (ms)	**Sem projeção (ms)**
Local	30860	35541
Docker-machine	38519	42436

Figura 3.2: Análise de Métricas - Tarefa 2

3.3 Tarefa 3

Similarmente à tarefa anterior, averiguou-se o desempenho da interrogação quando aplicada uma projeção. Com base nos resultados obtidos (figura 3.3), verifica-se um aumento no tempo de execução da interrogação quando não é aplicada nenhuma projeção.

Alínea 3		
	Com projeção (ms)	**Sem projeção (ms)**
Local	55034	62787
Docker-machine	79275	81558

Figura 3.3: Análise de Métricas - Tarefa 3

Capítulo 4

Instruções para Utilização

Este projeto foi realizado num ambiente local, utilizando a ferramenta *Docker* e, posteriormente, remotamente, com recurso à *Google Cloud Platform* e à ferramenta *Docker-machine*. Por este motivo, encontram-se descritas as instruções de utilização dos programas implementados em ambos os ambientes. Ademais, foram descritas as diversas classes de verificação presentes em cada tarefa.

4.1 Máquina Local

O primeiro passo para a utilização das tarefas realizadas está na inicialização a configuração do *Docker* na pasta do *docker-hadoop*, através do comando

```
docker-compose up
```

Em seguida, é necessário carregar os ficheiros de dados diretamente para o *HDFS*, de forma a conseguir utilizá-los em tarefas *MapReduce*. Para este efeito, considere-se *PATH_ TO_ DATA* o *path* da pasta onde estão armazenados os dados e *FICHEIRO* o nome do ficheiro a colocar no *HDFS*.

```
docker run --env-file hadoop.env -v PATH_TO_DATA:/data --network docker-hadoop_default -it bde2020/hadoop-base hdfs dfs -put /data/title.basics.tsv.gz /

docker run --env-file hadoop.env -v PATH_TO_DATA:/data --network docker-hadoop_default -it bde2020/hadoop-base hdfs dfs -put /data/title.ratings.tsv.gz /
```

Uma vez que são utilizados ficheiros *AvroParquet*, é indispensável carregar também os ficheiros dos esquemas hierárquicos definidos.

```
docker run --env-file hadoop.env -v PATH_TO_DATA/:/data --network docker-hadoop_default -it bde2020/hadoop-base hdfs dfs -put /data/FILENAME /
```

Relativamente à execução do código *Java*, deve ser criada uma imagem para o *Dockerfile*. Devido à forma como foi desenvolvido o projeto, é necessário indicar a classe que se pretende executar e gerar o ficheiro *jar* correspondente, tirando proveito do *IDE Intellij*.

Para confirmar o resultado de cada interrogação, podem ser executados os seguintes comandos, onde RESULT_DIR corresponde à diretoria criada durante a tarefa.

```
hdfs dfs -ls / hdfs dfs -cat /RESULT_DIR/part-r-0000
```

4.2 *Docker machine*

De forma a executar as tarefas realizadas num ambiente remoto, foi criada uma máquina virtual, com o seguinte comando.

```
docker-machine create --driver google --google-project ggcd-308020 --google-zone europe-west1-b --google-machine-type n1-standard-4 --google-disk-size=50 project
```

Posteriormente, de modo a ter acesso a este ambiente, é necessário executar os comandos:

```
docker-machine env project

eval $(docker-machine env project)
```

A partir deste momento, quaisquer comandos *docker* realizados correm sobre a máquina remota, uma vez que a variável de configuração foi redefinida, apontando para essa máquina (*echo $DOCKER_HOST*).

De forma a utilizar o *docker* remoto a partir do IDE, é necessário alterar a configuração do *Dockerfile*, conectando ao *daemon* com o *Docker Machine*.

Finalmente, após inicialização da configuração do *Docker* (*docker-compose up*), é necessário transferir os ficheiros de dados para o HDFS.

```
curl https://datasets.imdbws.com/title.basics.tsv.gz | hdfs dfs -put - /title.basics.tsv.gz

curl https://datasets.imdbws.com/title.ratings.tsv.gz | hdfs dfs -put - /title.ratings.tsv.gz
```

Para o carregamento dos esquemas hierárquicos definidos, são executados os comandos:

```
cat PATH_TO_DATA | docker exec -i namenode bash -- hdfs dfs -put - /FILENAME
```

O processo de execução e verificação dos resultados são realizados de forma idêntica ao ambiente local.

4.3 Processo de Execução

4.3.1 Tarefa 1

Para a execução da primeira tarefa, é necessário carregar para o HDFS o esquema hierárquico **schema.movies**, indicando no *Dockerfile* a classe *GGCD_Alinea1.ToParquet*.

4.3.2 Tarefa 2

A execução da segunda tarefa requer a importação dos esquemas **schema.queries** e **schema.alinea2**. Para além disso, é indispensável que esteja presente o ficheiro *AvroParquet* gerado pela tarefa anterior (*resultado_parquet*). Deve ser indicada no *Dockerfile* a classe *GGCD_Alinea2_Parquet.FromParquetToParquetFile*.

4.3.3 Tarefa 3

Na terceira tarefa, é necessário carregar o esquema hierárquico **schema.alinea3** para o HDFS. Para além disso, é indispensável que esteja presente o ficheiro *AvroParquet* gerado pela primeira tarefa (*resultado_parquet*). Deve ser indicada no *Dockerfile* a classe *GGCD_Alinea3.FromParquetToTextAlinea3*.

4.4 Classes de Verificação

4.4.1 Tarefa 1

Uma vez que na primeira tarefa é criado um ficheiro *AvroParquet*, foram criadas duas classes que permitem verificar se o conteúdo deste ficheiro é o pretendido. Posto isto, tem-se:

- **GGCD_Alinea1.ToFileString**: Escreve num ficheiro de texto todas as informações de cada filme, incluindo a sua classificação e número de votos;
- **GGCD_Alinea1.ToFileStringAll**: Escreve num ficheiro de texto todas as informações de todas as entradas do ficheiro *title.basics.tv.gz*, incluindo a sua classificação e número de votos.

Na figura que se segue, observa-se um excerto do resultado da primeira tarefa.

```
tt0124346	movie	Dear Fanny	Dear Fanny	1	1985	null	84	Adult	4.9	13
tt0124347	movie	Dear Pam	Dear Pam	1	1976	null	84	Adult,Comedy	6.1	30
tt0124348	movie	Death Kiss	Eglima sto Kavouri	0	1974	null	87	Horror,Thriller	5.5	171
tt0124349	movie	Redline Deathline	0	1997	null	97	Action,Sci-Fi	4.9	1135
tt0124350	video	Debbie Class of '95	Debbie Class of '95	1	1995	null	82	Adult	7.6	5
tt0124351	movie	Debbie Does 'Em All	Debbie Does 'Em All	1	1985	null	77	Adult	6.5	53
tt0124352	movie	Debbie Does Dallas III: [The Final Chapter]	Debbie Does Dallas III: [The Final Chapter]	1	1985	null	73	Adult,Comedy	4.7	118
tt0124353	movie	Debbie Does Las Vegas	Debbie Does Las Vegas	1	1981	null	62	Adult	3.4	18
tt0124354	video	Debutante Training	Debutante Training	1	1995	null	58	Adult	null	null
tt0124355	video	Decadence	Decadence	1	1994	null	77	Adult	null	null
```

Figura 4.1: Resultado Alínea 1 (Excerto)

4.4.2 Tarefa 2

Na segunda tarefa, foi solicitado que as respostas às diferentes alíneas fossem apresentadas num único ficheiro *AvroParquet*. Por este motivo, foram realizadas duas abordagens distintas.

Abordagem 1

- **GGCD_Alinea2_Text.FromParquetToTextFile**: Nesta classe, a informação obtida em cada alínea é guardada em ficheiros de texto distintos, lendo a informação que se encontra armazenada no ficheiro *AvroParquet* gerado pela classe *GGCD_Alinea1.ToParquet*.

Nesta abordagem, é necessário carregar os esquemas schema.query1, schema.query2 e schema.query3 para o HDFS. No final, criados os ficheiros *resultado_from_parquet_query1_text* (contém o número total de filmes por ano), *resultado_from_parquet_query2_text* (com o filme que recolheu maior número de votos, por ano) e *resultado_from_parquet_query3_text* (contém os dez filmes com melhor classificação, por ano).

Abordagem 2

- **GGCD_Alinea2_Parquet.Verifica**: Escreve num ficheiro de texto a informação que está armazenada no ficheiro *AvroParquet* gerado na classe *GGCD_Alinea2_Parquet.FromParquetToParquetFile*.

O resultado desta tarefa é apresentado por ano, por ordem crescente, onde se encontram o número total de filmes produzidos no ano, o filme que recolheu mais votos e a lista dos dez filmes com melhor classificação (figura 4.2).

```
2001    5346    tt0120737       The Lord of the Rings: The Fellowship of the Ring       1679564
Top 10:
1- tt0274494    Faber   9.8     5
2- tt1845324    Wimbledon Official Film 2001    9.6     5
3- tt0305275    Battleship Texas: The Lone Star Ship    9.5     6
4- tt0298799    Der chinesische Markt   9.4     10
5- tt2884508    Soldiers United for Cash        9.4     10
6- tt0396609    Gahanna Bill    9.4     7
7- tt6081240    Especial: São Paulo 447 Anos    9.3     95
8- tt0404238    Miracle Boy and Nyquist 9.3     7
9- tt0322693    Gangstacity     9.3     6
10- tt0304949   Wonderboy       9.2     5
```

Figura 4.2: Resultado Alínea 2 (Excerto)

Capítulo 5

Conclusão

A realização deste trabalho prático permitiu cimentar conhecimentos e competências adquiridas no decorrer da Unidade Curricular de Gestão de Grandes Conjuntos de Dados. Neste contexto, foi abordado o processo de desenvolvimento e implementação de vários métodos que permitem responder às interrogações colocadas.

Desta forma, foi possível implementar os algoritmos sob uma perspetiva de otimização do mesmos. Neste sentido, foram aplicadas projeções sobre os ficheiros *AvroParquet*, reduzindo a quantidade de dados lida e manipulada nas diferentes tarefas; foi construída uma classe personalizada (*CompositeKeyWritableA3*) para armazenamento e ordenação dos dados. A implementação destas medidas permitiram o acréscimo da performance das tarefas propostas.

Numa perspetiva futura, a reestruturação da segunda tarefa, de forma a permitir a sua execução com múltiplas tarefas de *reduce*, contribuiria significativamente na otimização da mesma.

Em suma, a realização deste trabalho exigiu a aplicação de todos os temas abordados em contexto de aula, permitindo que o grupo cumprisse todos os objetivos, tanto ao nível do exercício proposto, como ao nível do desenvolvimento de competências.

Bibliografia

[1] Feliz Gouveia. *Fundamentos de Bases de Dados*. 1ª ed. FCA - Editora de Informática, LDA, 2014, p. 31 (ver p. 13).