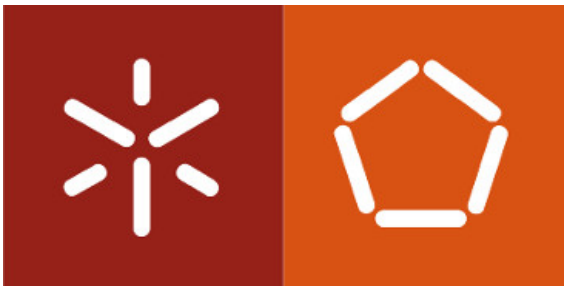

Universidade do Minho

Mestrado Integrado em Engenharia Informática

Ciências de Dados

Gestão de Grandes Conjuntos de Dados

IMDB - Spark

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

29 de junho de 2021

Conteúdo

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, recorrendo à biblioteca *Spark* e ao serviço *Hive Metastore*. De forma a responder a estas questões, foi utilizado o *dataset* público do IMDB [1].

2.1 Especificação dos Requisitos

De modo a tirar proveito das ferramentas apresentadas, foram propostas as seguintes tarefas de processamento:

1. Devem ser carregados os ficheiros do *dataset* para formatos mais eficientes e adequados às interrogações a implementar;
2. Recorrendo a *Resilient Distributed Datasets* (RDDs) do *Spark*, devem ser respondidas as seguintes interrogações:
   - **Top Genres** - Obter o género mais comum em cada década;
   - **Season Hits** - Para cada ano, recolher o título mais bem classificado;
   - **Top 10** - Calcular o *Top* 10 de atores que participaram em mais títulos diferentes;
3. Através de *Spark RDDs* ou *SQL*, devem ser desenvolvidas operações que materializem, num único ficheiro, os dados para **páginas de ator**, com a seguinte informação:
   - **Base** - Nome, idade, número de títulos em que participou, intervalo de anos de atividade e classificação média dos títulos em que participa;
   - **Hits** - *Top* 10 dos títulos mais bem classificados em que participou;
   - **Generation** - Nomes do *Top* 10 de atores da mesma geração (i.e., que nasceram na mesma década);
   - **Friends** - Conjunto de colaboradores de cada ator (i.e., outros atores que participaram nos mesmos títulos).

Para a conceção deste projeto, deve recorrer-se ao *Google Cloud*, de modo a utilizar diversos processos *worker* e armazenar todos os ficheiros *HDFS*.

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

2.2 Conceção da Resolução

2.2.1 Tarefa 1

A primeira tarefa consistiu no carregamento dos ficheiros em formatos eficientes e adequados às operações efetuadas. Para este efeito, os ficheiros dos *datasets* foram inicialmente guardados em pastas distintas que foram, por sua vez, adicionadas ao *HDFS* do *cluster*.

Para o ficheiro *title_ basics.tsv.gz*, foi criada uma tabela externa, composta pela totalidade dos dados existentes no ficheiro. As configurações definidas na tabela permitem que cada registo seja separado por linhas e cada género separado por vírgulas. A tabela é, posteriormente, armazenada na pasta *basics* do *HDFS* em formato de texto.

```
create external table title_basics (
        tconst string,
        titleType string,
        primaryTitle string,
        originalTitle string,
        isAdult boolean,
        startYear integer,
        endYear integer,
        runtimeMinutes integer,
        genres array<string>)
    row format delimited
    fields terminated by '\t'
    collection items terminated by ','
    lines terminated by '\n'
    stored as textfile
    location 'hdfs:///basics'
    tblproperties ("skip.header.line.count"="1");
```

Excerto 2.1: Tabela externa title_basics

De modo a ler os dados com eficiência, foi criada uma nova tabela, com os mesmos campos, armazenada em formato *parquet*. Esta tabela foi povoada com os dados da tabela anterior através do comando *SQL insert overwrite*.

```
create table title_basics_parquet (
        tconst string,
        primaryTitle string,
        originalTitle string,
        isAdult boolean,
        startYear integer,
        endYear integer,
        runtimeMinutes integer,
        genres array<string>)
    stored as parquet;

insert overwrite table title_basics_parquet select * from title_basics;
```

Excerto 2.2: Tabela parquet title_basics

Para os ficheiros *title.ratings.tsv.gz*, *title.principals.tsv.gz* e *name.basics.tsv.gz*, foi aplicado o mesmo processo de criação de tabelas. As tabelas *parquet* destas provenientes foram denominadas com o mesmo nome, acrescido de *parquet* (e.g., *title_ ratings_ parquet*).

2.2.2 Tarefa 2

Top Genres

Nesta interrogação, foi requerida a obtenção do género mais comum em cada década. Para esse efeito, foi utilizada a tabela *title_basics_parquet* previamente criada. Sobre esta tabela foram filtradas todas as entradas cujos géneros e ano sejam diferentes de nulo. Em cada entrada, foram criados, para cada género, pares ((género, década), 1), sendo estes posteriormente agrupados pela chave (género, década) e somados os valores. Estes dados foram mapeados num par (década, (género, número de ocorrências)), propiciando o agrupamento por década e consequente ordenação, por ordem decrescente. Subsequentemente, realizou-se a junção das entradas por década, ficando no formato (década, Iterable<(género, ocorrências)>). Finalmente, foi recolhido, para cada par, a primeira posição do *array*, obtendo-se, assim, o género com maior número de ocorrências de cada década. O resultado final é ordenado, permitindo a obtenção das décadas por ordem decrescente.

```
JavaPairRDD<String, Tuple2<Object,Integer>> query1 =
    spark.table("title_basics_parquet").toJavaRDD()
            .filter(l -> !l.isNullAt(8) && !l.isNullAt(5))
            .flatMapToPair(l -> {
                List<String> f = l.getList(8);
                return Arrays.stream(f.toArray())
                        .map(g -> new Tuple2<>(new Tuple2<>(g,get_decade(l.getInt(5))),
                            1)).iterator();
            })
            .reduceByKey((i, j) -> i + j)
            .mapToPair( l -> new Tuple2<>(l._1._2, new Tuple2<>(l._1._1,l._2)))
            .groupByKey()
            .mapValues( v -> {
                List<Tuple2<Object, Integer>> result = new ArrayList<Tuple2<Object,
                    Integer>>();
                v.forEach(result::add);
                result.sort(new ComparadorTuplosConta());
                return result.get(0);
            })
            .sortByKey(false);
```

Excerto 2.3: Top Genres

O cálculo da década em que se encontra cada entrada foi realizado através do método *get_decade*.

```
    public static String get_decade(int n){

        int dec = -1;
        dec = n - (n%10);
        return String.valueOf(dec);
    }
```

Excerto 2.4: Método para obtenção da década de uma entrada

No excerto que se segue, observa-se um excerto do resultado da primeira interrogação.

```
(2020,(Drama,86879))
(2010,(Drama,777658))
(2000,(Drama,404754))
(1990,(Drama,195351))
(1980,(Drama,120149))
(1970,(Drama,101454))
(1960,(Drama,62458))
(1950,(Drama,41136))
(1940,(Short,7735))
(1930,(Short,9304))
```

Excerto 2.5: Resultado Top Genres (Excerto)

Season Hits

Para esta interrogação era pedido o título mais bem classificado em cada ano. A concretização desta interrogação exigiu a leitura de duas tabelas *parquet* (*title_ basics_ parquet* e *title_ ratings_ parquet*).

No que diz respeito à primeira tabela, foram filtradas todas as entradas cujos géneros e ano sejam diferentes de nulo e criados pares no formato (tconst, (startYear, primaryTitle)). Na segunda tabela filtraram-se todas as entradas cujas classificações sejam nulas e foram criados pares na forma (tconst, averageRating).

Sobre estes RDDs foi realizada uma junção, dando origem a entradas no formato (tconst, ((startYear, primaryTitle), averageRating)).

As entradas foram, então, mapeadas, originando novos pares no formato (startYear, (primaryTitle, averageRating)). Subsequentemente, estes novos pares são agrupados por chave e é obtido o título com maior classificação do respetivo ano. Por fim, é realizada uma ordenação decrescente por ano (Excerto 2.7).

```
JavaPairRDD<String, Tuple2<Integer,String>> query2_basics =
    spark.table("title_basics_parquet").toJavaRDD()
            .filter(l -> !l.isNullAt(2) && !l.isNullAt(5))
            .mapToPair(l -> new Tuple2<>(l.getString(0), new
                Tuple2<>(l.getInt(5),l.getString(2))));

JavaPairRDD<String, BigDecimal> query2_ratings =
    spark.table("title_ratings_parquet").toJavaRDD()
            .filter(l -> !l.isNullAt(1))
            .mapToPair(l -> new Tuple2<>(l.getString(0), l.getDecimal(1)));

JavaPairRDD<Integer, Tuple2<String,BigDecimal>> query2_results =
    query2_basics.join(query2_ratings) //id,((ano, titulo), rating)
            .mapToPair(l -> new Tuple2<>(l._2._1._1, new Tuple2<>(l._2._1._2, l._2._2)))
                //ano (titulo,rating)
            .reduceByKey((Function2<Tuple2<String, BigDecimal>, Tuple2<String, BigDecimal>,
                Tuple2<String, BigDecimal>>)
                       (i1, i2) -> i1._2.floatValue() > i2._2.floatValue() ? i1 : i2)
            .sortByKey(false);
```

Excerto 2.6: Season Hits

```
(1983,(Episode #1.1,10.0))
(1982,(Ustrijelite Kastora,10.0))
(1981,(Arturo Benedetti Michelangeli,10.0))
(1980,(Julie Andrews' Invitation to the Dance with Rudolf Nureyev,10.0))
(1979,(The Unbroken Circle: A Tribute to Mother Maybelle Carter,10.0))
(1978,(Homo homini,9.9))
(1977,(Episode #1.1,10.0))
(1976,(Tri jablana,10.0))
(1975,(Kompanjoni,9.8))
(1974,(Lov,10.0))
(1973,(Krhka igracka,10.0))
(1972,(Episode #11.100,9.7))
(1971,(Autodafe moga oca,10.0))
(1970,(Episode #2.200,10.0))
(1969,(Burza rada,10.0))
(1968,(Paviljon broj VI,10.0))
(1967,(Vrijeme rakova,10.0))
```

Excerto 2.7: Resultado Season Hits (Excerto)

Top 10

A interrogação para obter o top 10 dos atores que participaram em mais títulos diferentes requer apenas da tabela *title_principals_parquet*. Assim sendo, após leitura da tabela, foram filtradas todas as entradas cuja categoria seja diferente de nulo e seja ator, atriz ou *self*. Os dados filtrados foram posteriormente colocados sobre a forma de par (nconst, 1), em que *nconst* representa o identificador de cada ator/atriz. Em seguida, os valores foram agrupados, de forma a obter o número de filmes em que cada ator participou. Os pares dos dados resultantes sofreram uma inversão da chave pelo valor, de forma a serem ordenados por número de títulos. Finalmente, foi realizada uma nova inversão e retirados os primeiros dez elementos do par resultante (Excerto 2.9).

```
List<Tuple2<String,Integer>> query3 = spark.table("title_principals_parquet").toJavaRDD()
            .filter(l -> !l.isNullAt(3) && (l.getString(3).equals("actor") ||
                l.getString(3).equals("actress") || l.getString(3).equals("self")))
            .mapToPair(l -> new Tuple2<>(l.getString(2), 1))
            .reduceByKey((i, j) -> i + j)
            .mapToPair(l-> l.swap()).sortByKey(false)
            .mapToPair(l-> l.swap())
            .take(10);
```

Excerto 2.8: Top 10

```
(nm0573012,10163)
(nm0001468,10091)
(nm0741299,10067)
(nm10120013,9518)
(nm0318114,9076)
(nm0005092,8983)
(nm0001992,8519)
(nm0871618,8124)
(nm0163863,7675)
(nm0756929,7388)
```

Excerto 2.9: Resultado Top 10

A execução da tarefa 2 teve uma duração de **1 minuto e 44 segundos**, incluindo o tempo de impressão dos resultados no terminal.

2.2.3 Tarefa 3

Para a última tarefa, é pedido o desenvolvimento de operações com Spark RDDs ou SQL que materializem num único ficheiro as informações obtidas das interrogações requeridas.

Base

Nesta primeira interrogação, pretende-se obter, para cada ator, o nome, idade, número de títulos em que participou, intervalo de anos de atividade e classificação média dos títulos em que participa. Por este motivo, esta interrogação foi dividida em sub-grupos.

A obtenção do nome e idade de cada ator foi conseguida através da leitura da tabela *name_basics_parquet*. Para o cálculo da idade do ator, foram verificados os campos *birthYear* e *deathYear*.

```
JavaPairRDD<String, Tuple2<String, Integer>> actors_info =
    spark.table("name_basics_parquet").toJavaRDD()
            .mapToPair(l -> new Tuple2<>(l.getString(0), new Tuple2<>(l.getString(1),
                l.isNullAt(2) ? 0 : (l.isNullAt(3) ? 2021 - l.getInt(2) : l.getInt(3) -
                l.getInt(2)))))
            .cache();
```

Excerto 2.10: Base - actors_info

Para o número de títulos em que participou, foram utilizados os dados da tabela *title_principals_parquet*, onde se filtraram todas as entradas cuja categoria seja diferente de nulo e seja ator, atriz ou *self*. Posto isto, as entradas foram mapeadas, de forma a obter pares (nconst, 1), em que *nconst* corresponde ao identificador do ator. Estes pares foram, posteriormente, agrupados, somando os seus valores.

```
JavaPairRDD<String, Integer> number_of_titles=
    spark.table("title_principals_parquet").toJavaRDD()
            .filter(l -> l.getString(3).equals("actor")|| l.getString(3).equals("actress") ||
                l.getString(3).equals("self"))
            .mapToPair(l -> new Tuple2<>(l.getString(2), 1))
            .foldByKey(0, Integer::sum)
            .mapToPair(p -> new Tuple2<>(p._1, p._2))
            .cache();
```

Excerto 2.11: Base - number_of_titles

De forma a conhecer o intervalo de anos de atividade de um ator, foi necessário ler as tabelas *title_principals_parquet* e *title_basics_parquet*. A filtragem e mapeamento realizados sobre esta tabela deram origem a entradas compostas pelo identificador do ator e o identificador do título em que participou (tconst, nconst). Por outro lado, as transformações aplicadas na segunda tabela permitiram a obtenção de cada título e o seu período de produção (tconst, (inicio, fim)).

A junção dos dois RDDs permitiu agregar o ator com a duração de produção de cada título. Subsequentemente, foi aplicado um mapeamento, onde se removeu o título, e realizada uma redução por chave, de forma a calcular o maior intervalo de anos de atividade para o respetivo ator.

```
JavaPairRDD<String, String> title_by_actor =
    spark.table("title_principals_parquet").toJavaRDD()
            .filter(l -> l.getString(3).equals("actor")|| l.getString(3).equals("actress") ||
                l.getString(3).equals("self"))
            .mapToPair( l -> new Tuple2<>(l.getString(0),l.getString(2)))
            .cache();

JavaPairRDD<String, Tuple2<Integer,Integer>> title_year =
    spark.table("title_basics_parquet").toJavaRDD()
            .mapToPair( l -> new Tuple2<>(l.getString(0),
                                new Tuple2<>( l.isNullAt(5) ? 2021 :
                                    l.getInt(5),l.isNullAt(6) ? 2021 : l.getInt(6))))
            .cache();
```

```
JavaPairRDD<String, Tuple2<Integer,Integer>> act_years = title_by_actor.join(title_year)
            .mapToPair( l -> new Tuple2<>(l._2._1,l._2._2))
            .reduceByKey(
                        (Function2<Tuple2<Integer, Integer>, Tuple2<Integer, Integer>,
                            Tuple2<Integer, Integer>>)
                                (i1, i2) -> new Tuple2<>(Math.min(i1._1, i2._1),
                                    Math.max(i1._2 , i2._2)))
            .cache();
```

Excerto 2.12: Base - Anos de atividade

Por fim, a classificação média dos títulos em que um ator participou requereu a leitura da tabela *title_ratings_parquet*, cujo mapeamento converteu as entradas em pares (tconst, averageRating).

A junção deste RDD com o *title_by_actor* (ver 2.12) originou entradas sob a forma de (tconst, (nconst, averageRating)). Sobre estas entradas foram aplicadas transformações que realizaram o cálculo da média das classificações dos títulos em que um ator participou.

```
JavaPairRDD<String, BigDecimal> title_rating =
    spark.table("title_ratings_parquet").toJavaRDD()
            .mapToPair( l -> new Tuple2<>( l.getString(0), l.getDecimal(1)))
            .cache();

JavaPairRDD<String, Float> actor_class = title_by_actor.join(title_rating)
        .mapToPair( l -> new Tuple2<>(l._2._1, new Tuple2<>(l._2._2,1)))
        .reduceByKey(
                    (Function2<Tuple2<BigDecimal, Integer>, Tuple2<BigDecimal, Integer>,
                        Tuple2<BigDecimal, Integer>>)
                    (i1, i2) -> new Tuple2<>(new BigDecimal(i1._1.floatValue() +
                        i2._1.floatValue()), i1._2+ i2._2))
        .mapToPair(v -> new Tuple2<>(v._1, v._2._1.floatValue() / v._2._2))
        .cache();
```

Excerto 2.13: Base - Classificação média

Hits

Nesta interrogação, é pedido o top 10 dos títulos mais bem classificados em que um ator participou. Para este efeito, foi realizada uma junção entre os RDDs *title_by_actor* e *title_rating*, dando origem a entradas no formato (tconst, (nconst, averageRating)). O mapeamento e consequente agrupamento dos valores permite obter entradas por ator (nconst, Iterable<(tconst, averageRating)>), possibilitando um segundo mapeamento, desta vez por valores, onde são recolhidos, através de um *Comparator* 2.15, os (no máximo) 10 títulos mais bem classificados em que um ator participou.

```
JavaPairRDD<String, String> hits = title_by_actor.join(title_rating)
            .mapToPair(l -> new Tuple2<>(l._2._1, new Tuple2<>(l._1,l._2._2)))
            .groupByKey()
            .mapValues( v -> {
                List<Tuple2<String, BigDecimal>> result = new ArrayList<Tuple2<String,
                    BigDecimal>>();
                v.forEach(result::add);
                result.sort(new ComparadorTuplos());
                result = result.subList(0,Integer.min(result.size(),10));
                StringBuilder sb = new StringBuilder();
                sb.append("Top 10 Titles: ");
                for (Tuple2<String,BigDecimal> aux : result)
                    sb.append("("+ aux._1 +","+aux._2+")");
                return sb.toString();
            })
            .cache();
```

Excerto 2.14: Base - Classificação média

```
class ComparadorTuplos implements Comparator<Tuple2<String, BigDecimal>>, Serializable {
    final static ComparadorTuplos INSTANCE = new ComparadorTuplos();
    public int compare(Tuple2<String, BigDecimal> t1, Tuple2<String, BigDecimal> t2) {
        return -t1._2.compareTo(t2._2);
    }
}
```

Excerto 2.15: ComparadorTuplos

Generation

Para esta interrogação, pretende-se conhecer os nomes do top 10 de atores da mesma geração, isto é, que nasceram na mesma década. Com base nas interrogações anteriormente respondidas, considera-se o número de títulos em que participaram como fator determinante para pertencerem a este *ranking*. Assim sendo, a resolução desta interrogação passa pela leitura da tabela *name_basics_parquet* e determinação da década a que um título pertence (nconst, (nome, década)).

Agrupando este RDD ao *number_of_titles* (ver 2.11), foi possível obter o número de títulos em que cada ator participou. O mapeamento dos valores, após algumas transformações, permitiu obter uma lista com, no máximo, dez atores para cada geração, com base no número de títulos em que participou. Esta obtenção foi conseguida recorrendo ao *Comparator* 2.17.

Finalmente, o agrupamento dos dois RDDs implementados possibilitou a obtenção de entradas no formato (década, ((nconst, nome), List<(nconst, número de títulos)>)). Desta forma, um mapeamento destas entradas originou pares com o identificador do título (*tconst*) e uma lista composta pelo identificador dos atores da mesma geração (nconst) e o número de títulos em que estes participaram.

```
JavaPairRDD<String, Tuple2<String, Integer>> actors_info_decade =
    spark.table("name_basics_parquet").toJavaRDD()
                .mapToPair(l -> new Tuple2<>(l.getString(0),
                        new Tuple2<>(l.getString(1), l.isNullAt(2) ? 190 : (int)
                            (l.getInt(2)/10))))
                .cache();

JavaPairRDD<Integer, String> c = actors_info_decade.join(number_of_titles)
                .mapToPair(l -> new Tuple2<>(l._2._1._2, new Tuple2<>(l._2._1._1, l._2._2)))
                .groupByKey()
                .mapValues( v -> {
                    List<Tuple2<String, Integer>> result = new ArrayList<Tuple2<String,
                        Integer>>();
                    v.forEach(result::add);
                    result.sort(new ComparadorTuplosTitles());
                    result = result.subList(0,Integer.min(result.size(),10));
                    StringBuilder sb = new StringBuilder();
                    sb.append("Top 10 Decade: ");
                    for (Tuple2<String,Integer> aux : result)
                        sb.append("("+ aux._1 +","+aux._2+")");
                    return sb.toString();
                })
                .cache();

JavaPairRDD<String, String> query3 = actors_info_decade
                .mapToPair(l -> new Tuple2<>(l._2._2, new Tuple2<>(l._1,l._2._1)))
                .join(c)
                .mapToPair(l -> new Tuple2<>(l._2._1._1, l._2._2))
                .cache();
```

Excerto 2.16: Generation

```
class ComparadorTuplosTitles implements Comparator<Tuple2<String, Integer>>, Serializable {
    final static ComparadorTuplosTitles INSTANCE = new ComparadorTuplosTitles();
    public int compare(Tuple2<String, Integer> t1, Tuple2<String, Integer> t2) {
        return -t1._2.compareTo(t2._2);
    }
}
```

Excerto 2.17: ComparadorTuplosTitles

Friends

Nesta última interrogação, é pretendido o conjunto de colaboradores de cada ator. Para este efeito, foi utilizado o resultado, armazenado em cache, das entradas que se encontram em formato (tconst, nconst) (ver 2.12). Em seguida, foi realizado um agrupamento do RDD com o próprio, permitindo que o valor do par seja, por sua vez, um par de dois atores. Uma vez que esta abordagem implica que um ator esteja relacionado consigo mesmo, tornou-se indispensável aplicar um filtro que removesse as entradas cujos elementos dos pares fossem iguais. Posto isto, as entradas foram mapeadas de forma a ignorar o identificador do título em questão, originando um par (nconst, nconst).

```
JavaPairRDD<String, Iterable<String>> friends = title_by_actor.join(title_by_actor)
            .filter(p -> !p._2._1.equals(p._2._2))
            .mapToPair(p -> p._2)
            .groupByKey()
            .cache();
```

Excerto 2.18: Friends

Um excerto do resultado da tarefa três pode ser verificado em seguida.

```
(nm0451858,(((((((Armen Khostikyan,74),5),(1955,2021)),6.5799994),
Top 10 Titles: (tt0415984,7.5) (tt0361100,7.2) (tt0406990,7.1) (tt0361966,6.3)
(tt0261192,4.8)), Top 10 Decade: (Ed McMahon,10163) (Johnny Gilbert,9076) (Johnny
    Carson,8519) (Dick Clark,7675) (Bob Barker,7148) (Gene Wood,5305) (Doc
    Severinsen,5185) (Merv Griffin,4614) (Mike Douglas,4447) (Hugh Downs,3744)),
[nm1212954, nm1347010, nm1351444, nm1372258, nm1045752, nm1537963, nm1356768,
    nm0594796, nm0594798, nm0592369, nm0594796, nm0016121, nm0341921, nm0765489,
    nm1372258, nm1045752, nm1617464]))
```

Excerto 2.19: Resultado Tarefa 3 (Excerto)

A execução e armazenamento em cache da tarefa 3 ocuparam **22 segundos**. No entanto, o armazenamento das quatro interrogações num único ficheiro de texto teve uma duração de **8 minutos e 32 segundos**.

Capítulo 3

Instruções para Utilização

Este projeto foi realizado remotamente, com recurso à *Google Cloud Platform*.

De forma a executar as tarefas realizadas num ambiente remoto, foi criado um *cluster* (sparktp2), composto por uma máquina *master* e duas máquinas *worker*. Este *cluster* foi configurado na região *europe-west1* e na zona *europe-west1-b*. Por sua vez, as máquinas que o constituem são *n1-standard-4* (**4 vCPUs, 15 GB de memória**), com discos de 500GB.

Numa primeira instância, é necessário transferir os ficheiros de dados para o HDFS. Uma vez que o *Hive metastore* requer que os ficheiros se encontrem em pastas dedicadas, os ficheiros dos *datasets* foram importados para o *HDFS* do seguinte modo:

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

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

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

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

Posteriormente, de modo a ter acesso à linha de comandos do *Hive*, é necessário conectar a um dos servidores do *Hive*, através do comando:

```
gcloud compute ssh sparktp2-m -- beeline -u "jdbc:hive2://localhost:10000"
```

Nesta linha de comando, é possível consultar e alterar os metadados existentes. Assim sendo, é nesta fase que são criados os metadados para os ficheiros armazenados (*create external table*). Os metadados de cada tabela *NAME_ TABLE* são observáveis através do comando

```
describe NAME_ TABLE;
```

A linha de comando permite ainda a conversão de ficheiros de formatos ineficientes para formatos mais eficientes (por exemplo, em formato *parquet*). O carregamento do conteúdo da tabela *NAME_ TABLE* em formato de texto para formato *parquet* é realizado da forma

```
insert overwrite table TABLE_ PARQUET select * from NAME_ TABLE;
```

De forma a confirmar a criação das tabelas *parquet*, pode ser executado o seguinte comando.

```
hdfs dfs -ls /user/hive/warehouse
```

Relativamente à execução do código *Java*, deve ser gerado o ficheiro *jar* correspondente, tirando proveito do *IDE Intellij*.

A execução das tarefas 2 e 3 é realizada através dos comandos

```
gcloud dataproc jobs submit spark --jars target/GGCD_Spark-1.0-SNAPSHOT.jar --class parte2.Queries --cluster sparktp2 --region europe-west1

gcloud dataproc jobs submit spark --jars target/GGCD_Spark-1.0-SNAPSHOT.jar --class parte3.Queries --cluster sparktp2 --region europe-west1
```

Capítulo 4

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, desde a conversão dos dados referentes a filmes e atores em informação, como o tratamento dos mesmos, tirando proveito da ferramenta *Spark*.

O desenvolvimento deste projeto permitiu a implementação dos algoritmos numa perspetiva de otimização, possibilitando uma melhor compreensão das vantagens da computação distribuída para tratamento de grandes conjuntos de dados.

Em suma, a realização deste projeto exigiu a aplicação de diversos 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.