Universidade do Minho
Departamento de Informática

Mestrado Integrado em Engenharia Informática

Gestão de Grandes Conjuntos de Dados

A82238 João Gomes

Braga
2021

Conteúdo

1 Introdução

Cada vez mais o tamanho dos conjuntos de dados utilizados em diversas aplicações aumenta e para acompanhar este crescimento, foram criadas diversas tecnologias que permitem realizar o tratamento e processamento destes enormes conjuntos de dados.

Este projeto, realizado no âmbito da Unidade Curriculad Gestão de Grandes Conjuntos de Dados, tem por objetivo fazer uso de tais tecnologias como **Hadoop**, **MapReduce**, **Avro Parquet** e **Spark** de forma a concretizar o processamento, tratamento e querying de grandes conjuntos de dados.

2 Objetivos

Para este projeto foram utilizados dois dos datasets públicos do IMDB, *title.basics* e *title.ratings*. Os objetivos deste projeto passaram por:

- Carregar os dados dos ficheiros para um formato **Avro Parquet**, utilizando MapReduce.
- Usando o ficheiro obtido na alínea anterior, calcular para cada ano:
  - O número total de filmes;
  - O filme que recolheu mais votos;
  - O género que tem mais filmes com classificação superior a 8.0.
- Utilizando **SparkRDDs**, calcular:
  - O género mais comum em cada década
  - O filme mais bem classificado em cada ano.

3 Tarefa 1

O primeiro passo para a realização desta tarefa foi a definição do esquema hierárquico para que possa fazer a leitura dos dados dos ficheiros para um ficheiro *AvroParquet* definido pelo esquema definido.

```
message Filmes {
    required binary tconst (STRING);
    required binary type (STRING);
    required binary primaryT (STRING);
    required binary originalT (STRING);
    required binary isAdult (STRING);
    required binary start (STRING);
    required binary end (STRING);
    required binary minutes (STRING);
    required binary rating (STRING);
    required binary numberVotes (STRING);
    required group generos (LIST)
    {
        repeated binary genero (STRING);
    }
}
```

Como se teve de carregar os dados de dois ficheiros diferentes, foram criados dois mappers diferentes, cada um responsável pelo tratamento de um dos ficheiros.

Do ficheiro *title.basics* foram retiradas as colunas: tconst, type, primaryTitle, originalTitle, isAdult, startYear, endYear, minutes e genres.. Estes dados foram todos guardados numa **String**, separados pelo delimitador _, sendo esta **String** devolvida pelo Mapper associada ao id do filme num par.

```
    @Override
        protected void map(LongWritable key, Text value, Context context) throws IOException,
        InterruptedException {
            if (key.get() == 0) return; // ignorar o cabeçalho

            String info [] = value.toString().split("\\t");

            if (info.length != 9) return;

            StringBuilder sb = new StringBuilder();
            sb.append(info[1] + "_"); // tipo
            sb.append(info[2] + "_"); // primary title
            sb.append(info[3] + "_"); // original title
            sb.append(info[4] + "_"); // isAdult
            sb.append(info[5] + "_"); // startYear
            sb.append(info[6] + "_"); // endYear
            sb.append(info[7] + "_"); // minutes

            boolean counter = false;
            for (String genre: info[8].split(","))
            {
                if (counter) sb.append(",");
                sb.append(genre);
                counter = true;
```

```
        }

        context.write(new Text(info[0]), new Text(sb.toString()));

    }
```

Do ficheiro title.ratings foram retiradas as colunas: rating e numberVotes. Tal como no primeiro ficheiro, estes dados foram guardados numa **String** separados pelo delimitador _sendo esta **String** devolvida pelo Mapper associada ao id do filme num par.

É de notar que neste mapper, os dados enviados pelo mapper são precedidos pela **String** *Ratings*, de forma que no reducer dê para distinguir de qual dos mappers vêm os dados.

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

        String [] info = value.toString().split("\\t");
        StringBuilder sb = new StringBuilder();

        sb.append("Ratings" + "_"); // indicar no reducer que este value vem de qual dos mappers
        sb.append(info[1] + "_"); // rating
        sb.append(info[2] + "_"); // votes

        context.write(new Text(info[0]), new Text(sb.toString()));
    }
```

No reducer, são agrupados os conteúdos de ambos os mappers de forma a serem processados e depois escritos no ficheiro destino *AvroParquet*. Nesta etapa de processamento podem ocorrer 3 cenários distintos: um dado titulo tem informação referente a ambos os ficheiros, só tem informação referente ao ficheiro basics ou só tem informação referentes ao ficheiro ratings. Apenas o primeiro caso é considerado, sendo um titulo descartado caso falte informação referente a qualquer um dos ficheiros.

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

    boolean ratings = false;
    boolean basics = false;

    GenericRecord record = new GenericData.Record(this.schema);
    record.put("tconst",key);

    for (Text value: values)
    {
        List <String> generos = new ArrayList<>();
        String [] info = value.toString().split("_");

        if (!info[0].equals("Ratings"))
        {
            record.put("type",info[0]);
            record.put("primaryT",info[1]);
```

```
            record.put("originalT",info[2]);
            record.put("isAdult",info[3]);
            record.put("start",info[4]);
            record.put("end",info[5]);
            record.put("minutes",info[6]);

            for (String genre: info[7].split(","))
                generos.add(genre);

            record.put("generos",generos);
            basics = true;
        }
        else
        {
            record.put("rating",info[1]);
            record.put("numberVotes", info[2]);
            ratings = true;
        }
    }

    if (basics && ratings)
        context.write(null, record);
    else return;

}
```

4 Tarefa 2

Nesta tarefa foram pedidas a implementação de algumas *Queries* referentes a cada ano.

4.1 Número total de filmes

No mapper da resolução desta *Query* começa-se por se verificar se o tipo do titulo a considerar é um filme. Se não for, descarta-se esta linha do ficheiro de input. De seguida, devolve-se um par cuja chave é o ano do filme e o valor é o número 1.

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

    String type = (String) value.get("type");

    if (!type.equals("movie")) return; // se não for do tipo movies,
                                       então nao interessam para esta alinea

    else
    {
        String year = (String) value.get("start");
        context.write(new Text(year), new LongWritable(1));
    }
}
```

No reducer, apenas se tem de somar o número de values associados a cada ano, resultando no número de filmes por cada ano.

```
@Override
protected void reduce(Text key, Iterable<LongWritable> values, Context context) throws IOExceptio
InterruptedException {
    long total = 0;
    for (LongWritable value: values)
        total += value.get();

    context.write(key,new Text(Long.toString(total)));
}
```

4.2 O filme que recolheu mais votos

No mapper desta resolução, efetuou-se novamente a verificação para apenas se considerarem títulos do tipo filme. De seguida, apenas se considera as colunas ano, numero de votos e o id do filme. De seguida constrói-se uma **String** que contém os dados dos votos e do id separados pelo delimitador _, que será devolvido como valor num par em que a chave é o ano.

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

    String type = (String) value.get("type");

    if (!type.equals("movie")) return; // se não for do tipo movies, então nao interessam para esta a
```

```
    else
    {
        String year = (String) value.get("start");
        String votes = (String) value.get("numberVotes");
        String id = (String) value.get("tconst");
        String nova = id + "_" + votes;

        context.write(new Text(year), new Text(nova));
    }
}
```

No reducer, primeiro efetua-se a verificação do número de votos, que pode vir como *"N"* e nesse caso, este valor do reducer não será tido em conta. Para calcular o filme com maior número de votos, sempre que se processa um valor verifica-se se o número de votos associado é maior do que o número de votos observado até agora, se sim substitui-se, se não não se considera. Desta forma, no fim de cada etapa de reduce, obter-se-á o filme com maior número de votos para cada ano.

```
@Override
protected void reduce(Text key, Iterable<Text> values, Context context) throws IOException,
InterruptedException {
    long maxVotes = 0;
    String maxVotesId = null;

    for (Text value: values)
    {
        String cValue = value.toString();
        String [] bocados = cValue.split("_");
        if (bocados[1].equals("\\N")) return;

        if (Long.parseLong(bocados[1]) > maxVotes)
        {
            maxVotes = Long.parseLong(bocados[1]);
            maxVotesId = bocados[0];
        }

    }
    context.write(key,new Text(maxVotesId));
}
```

4.3 Género com mais filmes superiores a 8.0 em classificação.

No mapper desta resolução efetuou-se novamente a verificação do tipo do título para filtrar apenas os titulos que sejam filmes. De seguida, filtrou-se todos os filmes cuja classicação seja inferior a 8.0. De seguida, para cada género que um dado filme tem, este mapper irá devolver o par (ano, género).

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

    String type = (String) value.get("type");
    float rating = Float.parseFloat((String) value.get("rating"));
```

```
    if (!type.equals("movie") || rating <= 8.0) return; // se não for do tipo movies ou rating for me

    else
    {
        String year = (String) value.get("start");

        List <String> lista = (List) value.get("generos");

        for (String genero: lista)
            context.write(new Text(year), new Text(genero));
    }
}
```

No reducer foi usado um **HashMap** para contar as ocorrências de cada gênero. No final, percorre-se este HashMap e vai-se guardando o gênero com mais ocorrências até ao momento.

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

    HashMap<String,Integer> contagem = new HashMap<>();
    int maxCount = 0;
    String generoMax = null;

    for (Text value: values)
    {
        if (!value.toString().equals("\\N"))
        {
            if (!contagem.containsKey(value.toString()))
                contagem.put(value.toString(),1);
            else
            {
                int count = contagem.get(value.toString());
                contagem.put(value.toString(),count+1);
            }
        }

    }

    for (Map.Entry<String, Integer> entry : contagem.entrySet())
    {
        if (entry.getValue() > maxCount)
        {
            maxCount = entry.getValue();
            generoMax = entry.getKey();
        }
    }
    String s = generoMax + " " + maxCount;
    context.write(key,new Text(s));
}
```

5 Tarefa 3

6 Análise de métricas

Para fazer um estudo do tempo de execução de cada uma das alíneas do segundo exercício, foi criado uma schema diferente para cada alínea onde apenas se guardou no ficheiro *AvroParquet* os dados necessários para realizar essa alínea, resultando na tabela seguinte com os tempos de execução para cada alínea com a schema geral apresentada na secção da Tarefa 1 e com a schema especializada para a alínea em questão.

	Alínea1	Alínea 2	Alínea 3
Schema geral	5502ms	5196ms	4803ms
Schema especifica	4347ms	4215ms	3778ms

7 Intruções para Utilização

7.0.1 Exercício 1 e 2

Para a realização deste trabalho, foi usado a ferramenta Docker e para tal é preciso primeiro realizar alguns passos de set up.

Primeiro, é necessário correr o container de docker, recorrendo ao comando

```
docker compose up
```

De seguida é necessário copiar os ficheiros de dados que se irão utilizar, bem como o ficheiro schema. Isto pode ser feito utilizando o comando

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

sendo PATH o caminho para a diretoria onde se encontram os ficheiros a copiar e FILE o nome do ficheiro que se deseja copiar.

Para a execução do código JAVA, deve ser criada uma imagem com Dockerfile e fazendo uso do IDE intellij executar o jar correspondente.