

Universidade do Minho

Gestão de Grandes Conjuntos de Dados Trabalho Prático nº 1

**Diogo Dias PG42825, Guilherme Palumbo PG42832, João Silva PG42834,
Nelson Costa PG44587**

MESTRADO DE ENGENHARIA INFORMÁTICA

2021

Índice

Lista de Figuras	ii
1 Introdução	1
1.1 Objectivos	1
2 Concretização e Avaliação Experimental	3
2.1 Preparação dos Dados	5
2.1.1 Maven	5
2.2 Processamento dos Dados	7
2.2.1 Processo 1 - Ficheiro AvroParquet	7
2.2.2 Processo 2 - Filmes	10
2.2.3 Processo 3 - Sistema de Recomendação	16
2.3 Resultados	19
3 Conclusão	22
Bibliografia	23
Apêndice A Informações Relevantes adicionais	24
A.1 Razão pela qual não conseguimos obter resultados	24

Índice de Figuras

2.1	Processo MapReduce	4
2.2	Dependência hadoop-client	5
2.3	Dependência parquet-hadoop-bundle	6
2.4	Dependência parquet-avro	6
2.5	Dependência slf4j-log4j12	6
2.6	Plugin Maven-compiler-plugin	6
2.7	Esquema AvroParquet	7
2.8	Funções readFile e getSchema	8
2.9	Classe Mapper Parquet [1]	9
2.10	Classe Mapper Parquet [2]	9
2.11	Classe main Parquet	10
2.12	Classe Mapper MovieCount	11
2.13	Classe Reducer MovieCount	11
2.14	Classe main MovieCount	12
2.15	Schema AvroParquetResult	13
2.16	Classe Mapper MovieVotedMovie	14
2.17	Classe Reducer MostVotedMovie	14
2.18	Classe Mapper Top10Movie	15
2.19	Classe Mapper Top10Movie - Cleanup	15
2.20	Classe Reducer Top10Movie	16
2.21	Classe Mapper RecommendationMovie	17
2.22	Classe Mapper RecommendationMovie - Cleanup	17
2.23	Classe Reducer RecommendationMovie	18
2.24	Classe Main RecommendationMovie	18
2.25	Classe Mapper MovieGenre	19
A.1	Cluster Criado	24
A.2	Instâncias geradas pelo cluster	24
A.3	utilização do cluster e instancia main	24
A.4	Jar gerado pelo package	26
A.5	Erro gcloud	26
A.6	Erro local	26

Capítulo 1

Introdução

No âmbito da unidade curricular “Gestão de Grandes Conjuntos de Dados” do perfil “Ciência de Dados”, foi nos proposto a concretização e avaliação experimental de tarefas de armazenamento e processamento de dados utilizando Hadoop HDFS, Avro+Parquet e MapReduce. Para desenvolvermos as tarefas utilizamos o dataset público do IMDB. Posto isto, o principal objectivo deste projeto é analisar e explorar o dataset com o auxílio das ferramentas utilizadas e apresentar um ficheiro AvroParquet com os resultados obtidos. No final deste trabalho prático, deve ser entregue um relatório onde é descrito todo o trabalho desenvolvido, assim como todos os documentos provenientes do mesmo.

1.1 Objectivos

Este trabalho prático é constituído por 3 processos a serem executados, sendo estes:

1. Carregar os dados dos ficheiros *title.basics.tsv.gz* e *title.ratings.tsv.gz* para um único ficheiro AvroParquet com um esquema apropriado.
2. Usar o ficheiro resultante da alinea anterior e considerando apenas filmes (movie), calcular para cada ano:

- o número total de filmes;
- o filme que recolheu mais votos;
- os 10 melhores filmes segundo a classificação.

Estes resultados devem ser armazenados num ficheiro AvroParquet com um esquema apropriado.

3. Para cada filme, recomendar outro do mesmo género que tenha a melhor classificação. Considerar apenas o primeiro género de cada filme. Estes resultados devem ser apresentados em ficheiros de texto e deve evitar carregar em memória simultaneamente todos os filmes do mesmo género.

Capítulo 2

Concretização e Avaliação Experimental

MapReduce é uma técnica de processamento. O algoritmo de MapReduce contém duas tarefas importantes: Mapear e Reduzir. O Map pega num conjunto de dados, onde os elementos individuais são divididos em tuplas (pares chave/valor). Em segundo lugar, reduz a tarefa, no qual seleciona a saída de um Map como entrada e combina essas tuplas de dados num conjunto menor de tuplas. Como a sequência do nome indica, MapReduce tem como tarefa a redução que é sempre executada após o mapeamento. A principal vantagem do MapReduce é a facilidade de dimensionar o processamento de dados em vários nós da computação. No modelo Map Reduce, existem os Mappers e Reducers. Depois de se escrever uma aplicação no formato MapReduce, escalar a aplicação para correr em centenas, milhares ou mesmo dezenas de milhares de máquinas num cluster é somente uma mudança de configuração.

1. O MapReduce é executado em 3 fases: fase map, fase shuffle e fase reduce.

- Fase Map: O trabalho do Map é processar os dados de entrada. Geralmente, os dados de entrada são na forma de arquivo ou diretório e são armazenados no sistema de arquivos Hadoop (HDFS). O arquivo de entrada é passado para a linha de função mapper por linha. O Mapper processa os dados e cria vários pequenos pedaços de dados.

- Fase Reduce: Esta fase é a combinação da fase shuffle e da fase Reducer. O trabalho do Redutor é processar os dados que vêm do Mapper. Após o processamento, produz um novo conjunto de saída, que será armazenado no HDFS.

2. Durante um job MapReduce, o Hadoop envia um Mapa e Reduz as tarefas para os servidores apropriados no cluster.
3. A estrutura gere todos os detalhes de passagem dos dados, como emissão de tarefas, verificação da conclusão da tarefa e cópia de dados ao redor do cluster entre os nós.
4. A maior parte da computação ocorre em nós com dados em discos locais que reduzem o tráfego da rede.
5. Após a conclusão das tarefas dadas, o cluster coleta e reduz os dados para formar um resultado apropriado e envia-os de volta para o servidor Hadoop.

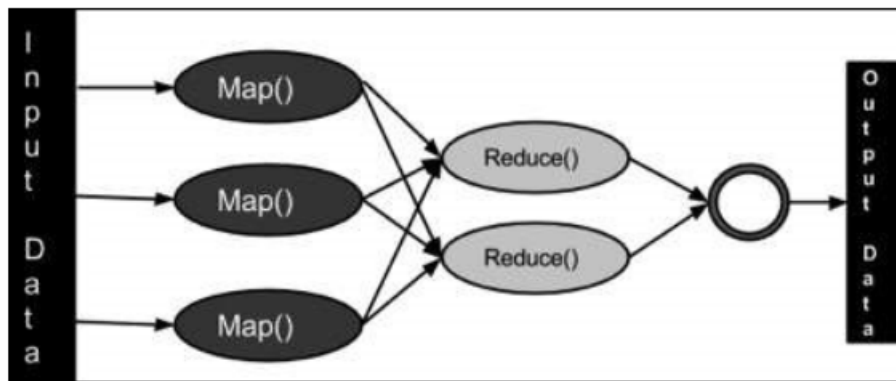


Figura 2.1: Processo MapReduce

2.1 Preparação dos Dados

2.1.1 Maven

Para iniciar a nossa preparação dos dados, começamos por adicionar algumas configurações ao `pom.xml`, ou seja, configurações do Maven. Estas configurações, resultaram no acréscimo de algumas dependências (`dependencies`) e alguns plugins, que serão explicadas ao longo desta secção.

Dependências

Neste trabalho foi usada a dependência `hadoop-client`, a dependência `parquet-hadoop-bundle`, a dependência `parquet-avro`, a dependência `slf4j-log4j12` e o plugin `maven-compiler-plugin`.

O Hadoop Client refere-se às bibliotecas clientes usadas para se comunicar com os componentes comuns de Hadoop (HDFS, MapReduce, YARN) [Figura 2.2].

```
<dependency>
  <groupId>org.apache.hadoop</groupId>
  <artifactId>hadoop-client</artifactId>
  <version>3.3.0</version>
</dependency>
```

Figura 2.2: Dependência `hadoop-client`

A dependência `parquet-hadoop-bundle` contém todas as classes que requerem o uso do Parquet dentro de um *environment* [Figura 2.3].


```
<dependency>
  <groupId>org.apache.parquet</groupId>
  <artifactId>parquet-hadoop-bundle</artifactId>
  <version>1.11.1</version>
</dependency>
```

Figura 2.3: Dependência parquet-hadoop-bundle

Dependência para usar os módulos de *Parquet-Avro* [Figura 2.4].

```
<dependency>
  <groupId>org.apache.parquet</groupId>
  <artifactId>parquet-avro</artifactId>
  <version>1.11.1</version>
</dependency>
```

Figura 2.4: Dependência parquet-avro

Para usar o *binding* Log4j 1.2 para o Slf4j é usada a dependência usada na figura seguinte [Figura 2.5].

```
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-log4j12</artifactId>
  <version>1.7.25</version>
</dependency>
```

Figura 2.5: Dependência slf4j-log4j12

O plugin Maven-compiler-plugin é usado para compilar as *sources* do projeto [Figura 2.6].

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-compiler-plugin</artifactId>
  <version>3.8.1</version>
  <configuration>
    <source>8</source>
    <target>8</target>
  </configuration>
</plugin>
```

Figura 2.6: Plugin Maven-compiler-plugin

2.2 Processamento dos Dados

Nesta secção, apresentamos as nossas soluções para os objetivos propostos anteriormente, utilizando tarefas MapReduce.

2.2.1 Processo 1 - Ficheiro AvroParquet

Neste processo, tínhamos de carregar os dados dos ficheiros *title.basics.tsv* e *title.ratings.tsv* para um único ficheiro, sendo este com a estrutura AvroParquet. Para isso, começamos por criar um esquema AvroParquet [Figura 2.7] para dar como entrada ao nosso modelo.

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

Figura 2.7: Esquema AvroParquet

No esquema AvroParquet decidimos manter o mesmo nome para cada uma das colunas, alterando apenas o formato do tipo de dados, ou seja, para os dados numéricos inteiros e decimais selecionamos, respetivamente, as opções *int32* e *float*, para as listas selecionamos *LIST*, para os dados binários (bool) selecionamos *boolean* e para os restantes dados selecionamos como *STRING*.

Depois de definir o esquema começamos por desenvolver as nossas tarefas de MapReduce. Neste caso, só desenvolvemos uma tarefa uma vez que o nosso ficheiro de entrada é um ficheiro de texto. Este mapper desenvolvido cria um registo e preenche-o com os dados relativos a cada entrada. Para este job não é necessário nenhum reducer.

Inicialmente começamos por criar duas funções, uma função de *readFile* e uma função *getSchema*. A função *readFile* permite ler ficheiros, enquanto que a função *getSchema* permite receber e seleccionar Schemas criados para serem utilizados. [Figura 2.8]

```
public static String readFile(String filePath) throws IOException {
    InputStream is = new FileInputStream(filePath);
    String data = new String( IOUtils.toByteArray(is));
    is.close();

    return data;
}

public static Schema getSchema(String schemaPath) throws IOException {
    MessageType mt = MessageTypeParser.parseMessageType(readFile(schemaPath));
    return new AvroSchemaConverter().convert(mt);
}
```

Figura 2.8: Funções readFile e getSchema

Uma vez criada as duas funções que servirão de apoio, desenvolvemos a função de Mapper. Os dados em que a função de Map vai operar são nomeadamente um LongWritable correspondente à chave e um Text correspondente ao valor do parâmetro de entrada (visto que estamos a tratar de strings). A função de Mapper irá por sua vez produzir pares, onde a chave será um Void e o valor de output será um GenericRecord (Uma instância genérica de um schema de registo).

Uma vez definido os valores de entrada e saída criamos uma variável String onde iremos ler um dos ficheiros de dados (*title.ratings.tsv.gz*.) Posteriormente criamos um ciclo onde, através dos valores armazenados nos arrays, criamos um novo HashMap (onde o primeiro argumento é a chave e o segundo argumento o

valor mapeado). A função `setup` permite obter o Schema criado do `AvroParquet` e aplicá-lo à função `fillRating()`. [Figura 2.9]

```
public class ToParquetMapper extends Mapper<LongWritable, Text, Void, GenericRecord>{
    private Schema schema;
    HashMap<String, Rating> ratings = new HashMap<>();

    private void fillRatings() throws IOException {
        String ratingsData = readFile( filePath "src/title.ratings.tsv.gz");
        boolean headerGone = false;
        // Fazer split as colunas
        String[] values = ratingsData.split( regex: "\\n");

        for(String row: values) {
            String[] fields = row.split( regex: "\\t");

            try{
                ratings.put(fields[0], new Rating(Double.parseDouble(fields[1]), Integer.parseInt(fields[2])));
            }
            catch (NumberFormatException e){
                if(!headerGone){
                    headerGone = true;
                }
                else {
                    System.out.println(e.getMessage());
                }
            }
        }
    }

    @Override
    protected void setup(Context context) throws IOException {
        schema = getSchema( schemaPath: "src/main/schemas/ToParquet.parquet" );
        fillRatings();
    }
}
```

Figura 2.9: Classe Mapper Parquet [1]

A função de Map propriamente dita, encontrada na Figura 2.10 começa por criar um novo `GenericRecord` e com o schema pretendido irá, para cada valor presente no array, criar um novo registo.

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

    //A chave tem de ser diferente de zero para ignorar-mos o cabeçalho senão não funciona
    if (key.get() != 0) {
        GenericRecord record = new GenericData.Record(schema);

        String[] values = value.toString().split( regex: "\\t");

        record.put( key: "tconst", values[0]);
        record.put( key: "titleType", values[1]);
        record.put( key: "primaryTitle", values[2]);
        record.put( key: "originalTitle", values[3]);
        record.put( key: "isAdult", ( Integer.parseInt(values[4]) == 1);
        record.put( key: "startYear", values[5].equals("\\N") ? null : Integer.parseInt(values[5]));
        record.put( key: "endYear", values[6].equals("\\N") ? null : Integer.parseInt(values[6]));
        record.put( key: "runtimeMinutes", values[7].equals("\\N") ? null : Integer.parseInt(values[7]));
        record.put( key: "genres", values[8].equals("\\N") ? null : Arrays.asList(values[8].split( regex: "\\|")));
        record.put( key: "averageRating", ratings.containsKey(values[0]) ? ratings.get(values[0]).getAvgRating() : null);
        record.put( key: "numVotes", ratings.containsKey(values[0]) ? ratings.get(values[0]).getNumVotes() : null);

        context.write( keyout: null, record);
    }
}
```

Figura 2.10: Classe Mapper Parquet [2]

Por último, a classe Main do Parquet é constituída por vários jobs onde ini-

cialmente é definido a qual a classe principal para a geração do ficheiro Jar e definido qual a classe Mapper a qual o job de Mapper estará atribuído. O output pretendido será um Void e estará no formato de AvroParquet. O input está no formato de Texto. O output portanto estará em AvroParquet com o respetivo schema desenvolvido. O ficheiro de input está no pathString apresentado na figura e o Output será localizado para o Path "outputParquet". Este output será o ficheiro AvroParquet com o resultado pretendido do exercício 1. [Figura 2.11]

```
public static void main(String[] args) throws IOException, ClassNotFoundException, InterruptedException {
    FileSystem filesystem = FileSystem.get(new Configuration());
    filesystem.delete(new Path( pathString: "outputParquet"),  b: true);

    Job job = Job.getInstance(new Configuration(),  jobName: "ToParquetJob");
    job.setJarByClass( ToParquet.class);
    job.setMapperClass(ToParquetMapper.class);

    job.setNumReduceTasks(0);

    job.setOutputKeyClass(Void.class);
    job.setOutputValueClass(AvroParquetOutputFormat.class);

    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(AvroParquetOutputFormat.class);

    AvroParquetOutputFormat.setSchema(job, getSchema(  schemaPath: "src/main/schemas/ToParquet.parquet"));
    FileInputFormat.addInputPath(job, new Path( pathString: "src/title.basics.tsv.gz"));
    FileOutputFormat.setOutputPath(job, new Path( pathString: "outputParquet"));

    job.waitForCompletion( verbose: true);
}
```

Figura 2.11: Classe main Parquet

2.2.2 Processo 2 - Filmes

Neste processo, para cada ano, tinhamos de apresentar várias características dos filmes (*movies*), nomeadamente qual o **número total de filmes**, o **filme que recebeu mais votos** e o **Top 10 de filmes segundo a classificação**.

Número total de filmes

Ao contrário do que aconteceu no Processo 1, em que desenvolvemos apenas uma tarefa de MapReduce, no Processo 2 desenvolvemos várias tarefas de MapReduce.

Com recurso à classe `MovieCountMapper` é realizada a divisão dos dados e é verificado se o registo do dataset contém um filme, em caso afirmativo regista na variável `year` com o ano e devolve `(year,um)` [Figura 2.12]

```
public static class MovieCountMapper extends Mapper<Object, Text, IntWritable, IntWritable> {
    private final static IntWritable UM = new IntWritable( value: 1);
    private final static IntWritable YEAR = new IntWritable();
    private static final String TYPE_MOVIE = "movie";

    @Override
    protected void map(Object key, Text value, Context context) throws IOException, InterruptedException {
        String[] tokens = value.toString().split( regex: "\\t");

        if(tokens.length >= 9 && tokens[1].equals(TYPE_MOVIE)) {
            int year = Integer.parseInt( tokens[5]);
            YEAR.set(year);
            context.write(YEAR, UM);
        }
    }
}
```

Figura 2.12: Classe Mapper MovieCount

Através da classe `MovieCountReducer` são somados todos os valores com a mesma chave (*movieCountPorAno*), apresentando a contagem de filmes por cada ano [Figura 2.13].

```
public static class MovieCountReducer extends Reducer<IntWritable, IntWritable, IntWritable, IntWritable> {
    private IntWritable result = new IntWritable();

    @Override
    protected void reduce(IntWritable key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {
        int movieCountPorAno = 0;
        for (IntWritable count: values) {
            movieCountPorAno += count.get();
        }
        result.set(movieCountPorAno);
        context.write(key, result);
    }
}
```

Figura 2.13: Classe Reducer MovieCount

De seguida, apresenta-se a classe main `MovieCount`, que contém os jobs necessários para a execução de todas as tarefas. Denota-se que o resultado es-

perado neste exercício seria um ficheiro AvroParquet, portanto, apesar de não termos conseguido obter nenhum resultado, assumimos que o Output seria em Avro Parquet.

O input desta função main, juntamente com o input dos dois exercícios posteriores, seria um input AvroParquet com o nome de *outputParquet* (ou seja, o resultado do processo 1). Seria então aplicado um novo schema (*AvroParquetResult*)[Figura 2.15], obtendo então um novo ficheiro. [Figura 2.14]

```
public static void main(String[] args) throws IOException, ClassNotFoundException, InterruptedException {

    Job job = Job.getInstance(new Configuration(), jobName: "MovieCountJob");
    job.setJarByClass(MovieCount.class);
    job.setMapperClass(MovieCountMapper.class);
    job.setReducerClass(MovieCountReducer.class);

    job.setMapOutputKeyClass(IntWritable.class);
    job.setMapOutputValueClass(IntWritable.class);

    /*
    job.setOutputKeyClass(IntWritable.class);
    job.setOutputValueClass(IntWritable.class);
    */
    job.setInputFormatClass(AvroParquetInputFormat.class);
    job.setOutputFormatClass(AvroParquetOutputFormat.class);

    /*
    FileInputFormat.addInputPath(job, new Path("src/title.basics.tsv.gz"));
    FileOutputFormat.setOutputPath(job, new Path("outputMovie"));
    */

    AvroParquetInputFormat.addInputPath(job, new Path(pathString: "outputParquet"));
    AvroParquetOutputFormat.setSchema(job, ToParquet.getSchema(schemaPath: "src/main/schemas/AvroParquetResult.parquet"));
    FileOutputFormat.setOutputPath(job, new Path(pathString: "resultsMovieCount"));

    job.waitForCompletion(verbose: true);
}
```

Figura 2.14: Classe main MovieCount

O schema *AvroParquetResult* [Figura 2.15]. seria (hipoteticamente) o resultado esperado dos 3 exercícios conjuntamente. Mas, como explicado, visto que não conseguimos gerar qualquer resultado, apenas supusemos que seria este o schema apropriado visto que os ficheiros iriam devolver o ano (*start year*); o número total de filmes desse ano (*totalMovies*); um grupo onde estaria o filme mais votado (*mostVoteMovie*). Esse grupo seria constituído pelo id do filme (*tconst*), pelo nome do filme (*primaryTitle*) e pelo número de votos do filme (*numVotes*). Este schema também seria constituído por um MAP (*Top10Movies*) que possuiria vários grupos dentro dele (*movieMap*) visto que é suposto gerar um Top 10 para cada ano. O *movieMap* seria constituído por um rank e por um outro grupo onde estariam incluídos o id novamente, o título e por último a classificação média (*avgRating*) por ordem decrescente.


```
message AvroParquetResult {
  required int32 startYear;
  required int32 totalMovies;
  optional group mostVoteMovie {
    required binary tconst (STRING);
    required binary primaryTitle (STRING);
    required int32 numVotes;
  }
  required group Top10Movies (MAP) {
    repeated group moviesMap {
      required binary rank (UTF8);
      optional group movieRatingInfo {
        required binary tconst (STRING);
        required binary primaryTitle (STRING);
        required float avgRating;
      }
    }
  }
}
```

Figura 2.15: Schema AvroParquetResult

Filme que recolheu mais votos

Para o cálculo do filme que recebeu mais votos por ano criamos inicialmente uma classe Mapper *MostVotedMovie*. Nesta classe decidimos limitar as entradas apenas aos filmes com mais de 1000 votos para facilitar o processo e poupar recursos. De seguida, e à semelhança do que aconteceu no exercício anterior, apenas os filmes eram selecionados. Após essa seleção é gerada uma chave de saída pelo *startyear* na qual devolve um Text com o *numVotes* o *averageRating* e o título do filme (*originalTitle*) [Figura 2.16].

Visto que não conseguimos correr o Avro Parquet, assumimos como ficheiro de entrada um (LongWritable, Text) referente ao ficheiro de dados local, isto porque posteriormente tentamos resolver o exercício localmente, utilizando o próprio ficheiro de texto, porém novamente sem sucesso. Contudo, se o nosso input fosse um ficheiro AvroParquet (resultado do processo 1) iríamos definir como dados de entrada (Void, GenericRecord). [Ver Anexo A]


```
static class MostVoteMovieMapper extends Mapper<LongWritable, Text, Text, Text> {

    private static final String TYPE_MOVIE = "movie";

    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {

        String tokens = value.toString();
        // Suposto formato [tconst] [titleType] [primaryTitle] [originalTitle] [isAdult] [startYear]
        // [endYear] [runtimeMinutes] [averageRating] [numVotes] [genres[genre]]
        String[] tokens_values = tokens.split(regex "\\t");

        // Apenas filmes com mais de 1000 votos para facilitar o processo
        if (Integer.parseInt(tokens_values[9]) > 1000 && tokens_values[1].equals(TYPE_MOVIE))
            context.write(new Text(tokens_values[5]), new Text(tokens_values[9] + "\\t" + tokens_values[8] + "\\t" + tokens_values[2]));
        // Gera (startYear), (numVotes + averageRating + originalTitle) para cada linha
    }
}
```

Figura 2.16: Classe Mapper MovieVotedMovie

Na classe MostVotedMovieReducer são realizadas validações de forma a selecionar qual o filme que recebeu mais votos. Decidimos que em caso de empate, o filme que possui maior classificação seria considerado o filme com mais votos. Ou seja, caso dois filmes possuam o mesmo número de votos, entre os dois será selecionado o filme com maior classificação. O resultado devolve como chave o *startyear* e como conteúdo o *avgRating* e *numVotes*. [Figura 2.17]

```
static class MostVoteMovieReducer extends Reducer<Text, Text, Text, Text> {

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

        float max_rating = Float.MIN_VALUE;
        int max_votes = Integer.MIN_VALUE;
        String[] tokens_values;
        String output = "";
        int vote;
        float rating;

        for (Text value: values) {
            // (numVotes + averageRating + originalTitle)
            tokens_values = value.toString().split(regex "\\t");

            // Valores numericos de rating e vote
            rating = Float.parseFloat(tokens_values[1]);
            vote = Integer.parseInt(tokens_values[0]);

            // Ver se os votes é maior que o max_vote
            if (vote > max_votes) {
                max_votes = vote;
                max_rating = rating;

                output = tokens_values[2] + "\\t" + tokens_values[0];
            }
            else if (vote == max_votes) {
                // Decidir pelo rating, caso os votos sejam igual ao max_vote.
                if (rating > max_rating) {
                    max_rating = rating;
                    max_votes = vote;

                    output = tokens_values[2] + "\\t" + tokens_values[0];
                }
            }
        }
        context.write(key, new Text(output));
    }
}
```

Figura 2.17: Classe Reducer MostVotedMovie

10 melhores filmes segundo a classificação

Através da classe Mapper *Top10MoviesMapper* inicialmente, caso ainda não exista, é gerado um treemap [Figura 2.18]. Posteriormente, o treemap é populado por até 10 filmes.

```
public static class Top10MoviesMapper extends Mapper<LongWritable, Text, Text, Text> {
    //Para armazenar os dados do map
    private Map<Integer, TreeMap<Float,Text>> mapa = new HashMap<>();
    private TreeMap<Float, Text> moviesMap = new TreeMap<>();
    private final static IntWritable YEAR = new IntWritable();
    private static final String TYPE_MOVIE = "movie";

    // Suposto formato [tconst] [titleType] [primaryTitle] [originalTitle] [isAdult] [startYear]
    // [endYear] [runtimeMinutes] [averageRating] [numVotes] [genres[genre]]

    @Override
    protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
        String[] tokens = value.toString().split( regex: "\\t");

        //
        if(tokens.length >= 9 && tokens[1].equals(TYPE_MOVIE)) {
            int year = Integer.parseInt(tokens[5]);
            YEAR.set(year);
            //se nao existir a chave cria um treeMap (para facilitar trabalho)
            TreeMap<Float, Text> t = mapa.putIfAbsent(year, new TreeMap<>());
            try
            {
                //Limitar o tamanho do treeMap para 10
                t.put(Float.parseFloat(tokens[0]), new Text(value));
                if(t.size() > 10)
                    t.remove(t.firstKey());
            }
            catch(NumberFormatException ex){}
        }
    }
}
```

Figura 2.18: Classe Mapper Top10Movie

Decidimos ainda na classe Mapper desenvolver uma função de Cleanup que irá transferir os Top10 filmes para cada map, sendo executado após todos os dados serem processados nesta partição [Figura 2.19]. Isto é o que nos permite gerar o resultado do mapper.

```
//Quando acabar todas as funcoes de map.
//No cleanup vou enviar o Top10 para cada valor do mapa. Ele executa depois de todos os dados serem processados desta partição.
@Override
protected void cleanup(Mapper<LongWritable, Text, Text, Text>.Context context) throws IOException, InterruptedException {
    for(Map.Entry<Integer, TreeMap<Float, Text>> t : mapa.entrySet()) {
        for(Map.Entry<Float, Text> v : t.getValue().entrySet()) {
            context.write(new Text(t.getKey().toString()), new Text( String: v.getKey() + "\t" + v.getValue()));
        }
    }
}
//Isto manda o Top10 para cada ano
```

Figura 2.19: Classe Mapper Top10Movie - Cleanup

Com recurso à classe Reducer (*Top10MoviesReducer*) é devolvido o resultado

com os 10 melhores filmes por ano por ordem decrescente, utilizando a classificação como medida. [Figura 2.20]

```
public static class Top10MoviesReducer extends Reducer<Text, Text, Text, Text> {
    @Override
    protected void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {
        float max_rating = Float.MIN_VALUE;
        int max_votes = Integer.MIN_VALUE;
        String[] LINE_values;
        String output = "";
        int vote = 0;
        float rating;
        TreeMap<Float, Text> moviesMap = new TreeMap<>();

        for (Text value: values) {
            LINE_values = value.toString().split( regex "\\t");

            try
            {
                //Limitar o tamanho do treeMap para 10
                moviesMap.put(Float.parseFloat(LINE_values[0]), new Text(LINE_values[1]));
                if(moviesMap.size() > 10)
                    moviesMap.remove(moviesMap.firstKey());
            }
            catch(NumberFormatException ex){}
        }

        //Para colocar por ordem decrescente
        for(Map.Entry<Float, Text> t : moviesMap.descendingMap().entrySet()) {
            context.write(new Text(t.getKey().toString()), t.getValue());
        }
    }
}
```

Figura 2.20: Classe Reducer Top10Movie

2.2.3 Processo 3 - Sistema de Recomendação

Ao desenvolvermos o sistema de recomendação tínhamos como objetivo recomendar para cada filme, um filme do mesmo género que tenha a melhor classificação, considerando apenas o primeiro género de cada filme.

Na classe RecommendationMovieMapper agrupamos os dois melhores filmes para cada género, limitando o tamanho do treeMap para 2. Tomamos esta decisão para caso o melhor filme seja seleccionado, será recomendado o segundo melhor filme daquele género. [Figura 2.21].

```
public static class RecommendationMovieMapper extends Mapper<LongWritable, Text, Text, Text> {
    //Para armazenar os dados do map
    private Map<String, TreeMap<Float,Text>> mapa = new HashMap<>();
    private TreeMap<Float, Text> moviesMap = new TreeMap<>();
    private final static Text GENRE = new Text();
    private static final String TYPE_MOVIE = "movie";

    // Suposto formato [tconst] [titleType] [primaryTitle] [originalTitle] [isAdult] [startYear]
    // [endYear] [runtimeMinutes] [averageRating] [numVotes] [genres[genre]]

    @Override
    protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
        String[] tokens = value.toString().split( regex: "\\t");

        //
        if(tokens.length >= 9 && tokens[1].equals(TYPE_MOVIE)) {
            String genero = (tokens[10].split( regex: "\\t" )[0]);
            GENRE.set(genero);
            Float ratings = Float.parseFloat(tokens[8]);
            Text filme = new Text(tokens[2]);
            //se não existir a chave cria um TreeMap (para facilitar trabalho)
            TreeMap<Float, Text> t = mapa.putIfAbsent(genero, new TreeMap<>());
            try
            {
                //Limitar o tamanho do TreeMap para 2. Para cada genero uma lista com os 2 melhores filmes
                t.put(ratings, filme);
                if(t.size() > 2)
                    t.remove(t.firstKey());
            }
            catch (NumberFormatException ex){}
        }
    }
}
```

Figura 2.21: Classe Mapper RecommendationMovie

Através do cleanup da classe Mapper *RecommendationMovie* é enviado os dois melhores filmes para cada valor do map. É executado depois de todos os dados serem processados nesta partição [Figura 2.22].

```
//Quando acabar todas as funções de map.
//No cleanup vou enviar o Top10 para cada valor do mapa. Ele executa depois de todos os dados serem processados desta partição.
@Override
protected void cleanup(Mapper<LongWritable, Text, Text, Text>.Context context) throws IOException, InterruptedException {
    for(Map.Entry<String, TreeMap<Float, Text>> t : mapa.entrySet()) {
        for(Map.Entry<Float, Text> v : t.getValue().entrySet()) {
            context.write(new Text(t.getKey()), new Text( string: v.getKey() + "\\t" + v.getValue()));
            //(genre), (ratings + filme)
        }
    }
}
//Isto manda o Top10 para cada ano
```

Figura 2.22: Classe Mapper RecommendationMovie - Cleanup

Através da classe *RecommendationMovieReducer* são apresentados os 2 filmes recomendados por ordem decrescente. [Figura 2.24].

```
public static class RecommendationMovieReducer extends Reducer<Text, Text, Text, Text> {
    @Override
    protected void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {
        String[] LINE_values;
        String output = "";
        int vote = 0;
        float rating;
        TreeMap<Float, Text> moviesMap = new TreeMap<>();

        for (Text value: values) {
            LINE_values = value.toString().split( regex: "\\t");

            try
            {
                //Limitar o tamanho do treeMap para 2
                moviesMap.put(Float.parseFloat(LINE_values[0]), new Text(LINE_values[1]));
                if(moviesMap.size() > 2)
                    moviesMap.remove(moviesMap.firstKey());
            }
            catch(NumberFormatException ex){}
        }

        //Para colocar por ordem decrescente
        for(Map.Entry<Float, Text> t : moviesMap.descendingMap().entrySet()) {
            context.write(new Text(t.getKey().toString()), t.getValue());
        }
    }
}
```

Figura 2.23: Classe Reducer RecommendationMovie

```
public static void main(String[] args) throws Exception {

    Job job = Job.getInstance(new Configuration(), jobName: "RecommendationMovieJob");
    job.setJarByClass(RecommendationMovie.class);
    job.setMapperClass(RecommendationMovieMapper.class);
    job.setReducerClass(RecommendationMovieReducer.class);

    job.setMapOutputKeyClass(Text.class);
    job.setMapOutputValueClass(Text.class);

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(Text.class);
    job.setOutputFormatClass(TextOutputFormat.class);

    AvroParquetInputFormat.setRequestedProjection(job, getSchema( schemaPath: "src/main/schemas/ToParquet.parquet"));
    AvroParquetInputFormat.addInputPath(job, new Path( pathString: "outputParquet"));
    TextOutputFormat.setOutputPath(job, new Path( pathString: "resultsRecommendationMovie"));

    job.waitForCompletion( verbose: true );
}
```

Figura 2.24: Classe Main RecommendationMovie

A Class *MovieGenre* vem complementar o Sistema de Recomendação. Enquanto que a Class *RecommendationMovie* devolve apenas os 2 melhores filmes, na class *MovieGenre* será onde vamos efetivamente recomendar o melhor filme. Para tal, é criado um array com os dois melhores filmes, de forma a que no índice 0 conste o melhor filme e no índice 1 conste o segundo melhor filme. Se o melhor filme for selecionado, o filme recomendado passará a ser o segundo filme do array. [Figura 2.25]. Desta forma conseguimos garantir que para o filme do mesmo género é recomendado o filme com a melhor classificação.

```
public static class MovieGenreMapper extends Mapper<Object, Text, Void, Text> {
    private Map<String, TreeMap<Float, Text>> mapa = new HashMap<>();
    //Assumindo que isto corresponde para cada género o melhor filme.

    // Suposto formato [tconst] [titleType] [primaryTitle] [originalTitle] [isAdult] [startYear]
    // [endYear] [runtimeMinutes] [averageRating] [numVotes] [genres[genre]]
    @Override
    protected void map(Object key, Text value, Context context) throws IOException, InterruptedException {
        String token = value.toString();
        String[] tokens = token.split( regex: "\\t");

        if(tokens.length >= 9) {
            String nome = (tokens[2]);
            String genero = (tokens[10].split( regex: "\\t" )[0]);
            TreeMap<Float, Text> t = mapa.get(genero);
            //Cria um array com os 2 melhores filmes
            // no índice 0 o melhor filme e no índice 1 o segundo melhor filme
            List<Text> array = new ArrayList(t.values());

            String melhorFilme = array.get(0).toString();

            if(melhorFilme.equals(nome)) {
                melhorFilme = array.get(1).toString();
            }

            context.write( keyout: null, new Text( string: nome + "\\t" + melhorFilme));
        }
    }
}
```

Figura 2.25: Classe Mapper MovieGenre

2.3 Resultados

Nestas secção iremos apresentar alguns resultados, ainda que estes não sejam os verdadeiros. Como já foi referido tivemos alguns problemas técnicos que nos impossibilitaram de executar o código, consequentemente não obtendo resultados reais.

Desta forma, através da Preparação e Processamento dos Dados a expectativa era obter os seguintes outputs dos schemas:

```
1 {
2   "movies": [
3     {
4       "tconst": "tt0232499",
5       "titleType": "movie",
6       "primaryTitle": "Toy Story",
7       "originalTitle": "Toy Story",
8       "isAdult": false,
9       "startYear": 1998,
10      "endYear": 1998,
11      "runtimeMinutes": 120,
12      "genres": [
13        "Action",
14        "Adventure",
15        "Family"
16      ],
17      "avgRating": 9.1,
18      "numVotes": 6890
19    },
20    { ... }
21  ]
22 }
```

Listing 1: Output do Processo 1 - Ficheiro AvroParquet

```
1 {
2   "2005": {
3     "totalMovies": 54,
4     "mostVoteMovie": {
5       "tconst": "tt104423580",
6       "primaryTitle": "Toy Story",
7       "numVotes": 6890
8     },
9     "Top10Movies":{
10      "1":{
11        "tconst": "tt2237780",
12        "primaryTitle": "The NoteBook",
13        "avgRating": 9.1
14      },
15      "2": { ... },
16      ...
17    },
18  },
19  "1948": {
20    "totalMovies": 30,
21    "mostVoteMovie": {
22      "tconst": "tt104429990",
23      "primaryTitle": "12 Angry Man",
24      "numVotes": 8888
25    },
26    "Top10Movies":{
27      "1":{
28        "tconst": "tt104429990",
29        "primaryTitle": "12 Angry Man",
30        "avgRating": 9.6
31      },
32      "2": { ... },
33      ...
34    },
35  }
36 }
```

Listing 2: Output do Processo 2 - Filmes

Capítulo 3

Conclusão

O principal objetivo deste projeto era analisar, explorar e processar os dataset *title.basics.tsv.gz* e *title.ratings.tsv.gz* com o auxílio de ferramentas como Hadoop HDFS, Avro+Parquet e MapReduce. Inicialmente fizemos uma preparação dos dados onde definimos o nosso ficheiro *pom.xml* com todas as dependências e plugins necessários. De seguida, realizamos a leitura dos datasets e os organizamos num único ficheiro AvroParquet.

Supostamente a partir do ficheiro obtido, e através do MapReduce, realizamos alguns cálculos para responder a algumas questões, sendo estas o número total de filmes lançados por ano, o filme que recolheu mais votos por ano e os 10 melhores filmes segundo a classificação por ano. O processo estaria concluído com o armazenar dos resultados obtidos num ficheiro AvroParquet. Terminamos por fazer um sistema de recomendação utilizando também o ficheiro obtido na primeira alínea, onde para cada filme recomendou outro filme com melhor classificação. Com a realização deste projeto e apesar das nossas dificuldades e contratempos, compreendemos a importância e usabilidade de ferramentas como o AvroParquet para criar schemas para os nossos dados, o MapReduce, onde os dados são divididos em blocos que podem ser tratados separadamente de forma a tornar o processo de distribuição mais eficiente e o Hadoop HDFS para um Armazenamento escalável.

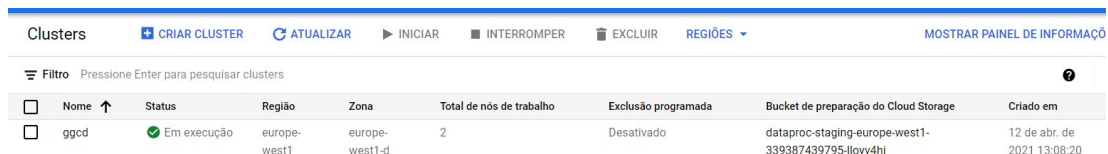
Bibliografia

- [1] T. White, *Hadoop: The definitive guide, Chapter 4*. "O'Reilly Media, Inc.", 2012.
- [2] M. H. Almeer *et al.*, "Cloud hadoop map reduce for remote sensing image analysis," *Journal of Emerging Trends in Computing and Information Sciences*, vol. 3, no. 4, pp. 637–644, 2012.
- [3] G. Cloud, "gcloud dataproc." <https://cloud.google.com/sdk/gcloud/reference/dataproc>.
- [4] D. Flair, "Inputsplit in hadoop mapreduce – hadoop mapreduce tutorial." <https://data-flair.training/blogs/inputsplit-in-hadoop-mapreduce/>.
- [5] Ansible, "Google.cloud." <https://docs.ansible.com/ansible/latest/collections/google/cloud/>.
- [6] M. J. M. Chuquicusma, S. Hussein, J. Burt, and U. Bagci, "How to fool radiologists with generative adversarial networks? a visual turing test for lung cancer diagnosis," 2018.

Apêndice A

Informações Relevantes adicionais

A.1 Razão pela qual não conseguimos obter resultados



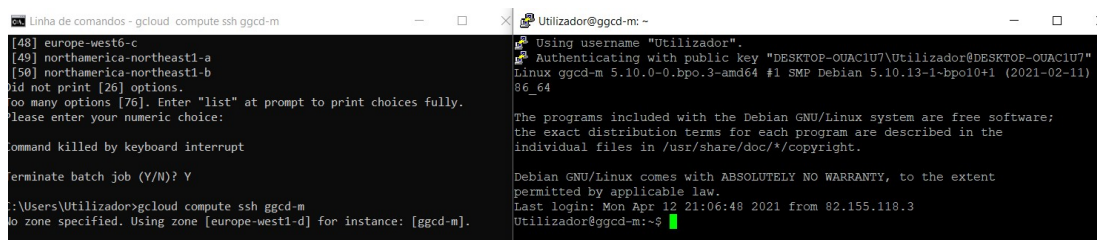
	Nome	Status	Região	Zona	Total de nós de trabalho	Exclusão programada	Bucket de preparação do Cloud Storage	Criado em
<input type="checkbox"/>	ggcd	Em execução	europe-west1	europe-west1-d	2	Desativado	dataproc-staging-europe-west1-339387439795-lloyv4hj	12 de abr. de 2021 13:08:20

Figura A.1: Cluster Criado



	Nome	Zona	Recomendações	Em uso por	IP interno	IP externo	Conectar
<input type="checkbox"/>	ggcd-m	europe-west1-d			10.132.0.7 (nic0)	35.205.46.204	SSH
<input type="checkbox"/>	ggcd-w-0	europe-west1-d			10.132.0.9 (nic0)	104.199.7.249	SSH
<input type="checkbox"/>	ggcd-w-1	europe-west1-d			10.132.0.8 (nic0)	35.205.247.170	SSH

Figura A.2: Instâncias geradas pelo cluster



```

Linha de comandos - gcloud compute ssh ggcd-m
[48] europe-west6-c
[49] northamerica-northeast1-a
[50] northamerica-northeast1-b
did not print [26] options.
too many options [76]. Enter "list" at prompt to print choices fully.
Please enter your numeric choice:
command killed by keyboard interrupt
terminate batch job (Y/N)? Y
[Users\Utilizador>gcloud compute ssh ggcd-m
no zone specified. Using zone [europe-west1-d] for instance: [ggcd-m].

Utilizador@ggcd-m:~$
Using username "Utilizador".
Authenticating with public key "DESKTOP-OUAC1U7\Utilizador@DESKTOP-OUAC1U7"
Linux ggcd-m 5.10.0-0.bpo.3-amd64 #1 SMP Debian 5.10.13-1~bpo10+1 (2021-02-11) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon Apr 12 21:06:48 2021 from 82.155.118.3
Utilizador@ggcd-m:~$
  
```

Figura A.3: utilização do cluster e instancia main

Nós seguimos todos os tutoriais do professor, criamos o cluster na cloud [Figura A.1 e A.2], definimos tudo direito como pretendido, porém quando tentamos

correr o código utilizando a cloud aparece-nos o seguinte erro: [Figura A.5]
Para tal, utilizamos o código:

```
gcloud dataproc jobs submit hadoop
--jars=target/ggcd_TP1-1.0-SNAPSHOT.jar
--class=ToParquet --cluster=ggcd --region=europe-west1 --
```

Fizemos os testes todos e tudo aparenta estar normal, o cluster possui dois ficheiros hdfs hadoop, tudo como esperado, porém dá sempre erro.

Da mesma forma, tentamos então fazer o exercício localmente, porém deparamo-nos com outro problema. Todos nós usamos windows portanto a instalação do Hadoop foi muito complicada. Inicialmente aparecia-nos um erro que dizia que o "HADOOP_HOME and hadoop.home.dir are unset". Tentamos resolvê-lo instalando o Hadoop por completo e aplicando os Paths necessários.

Seguimos este tutorial para a instalação: <https://www.datasciencecentral.com/profiles/blogs/how-to-install-and-run-hadoop-on-windows-for-beginners>

Mesmo instalando o Hadoop localmente estava sempre a dar o mesmo erro do qual não conseguimos resolver, passamos horas e horas a tentar solucionar problemas, até pedimos ajuda externa e sem sucesso. [Figura A.6].

Porque não conseguimos testar nenhuma Class efetivamente, durante todo o trabalho tivemos imensas dificuldades em perceber qual o caminho que deveríamos seguir, daí a nossa demora e dificuldade com o projeto. O trabalho que estamos a submeter não foi efetivamente testado e corrido, portanto durante todo o relatório nós explicamos o que fizemos e apresentamos os resultados que esperamos que serão gerados a partir do código.

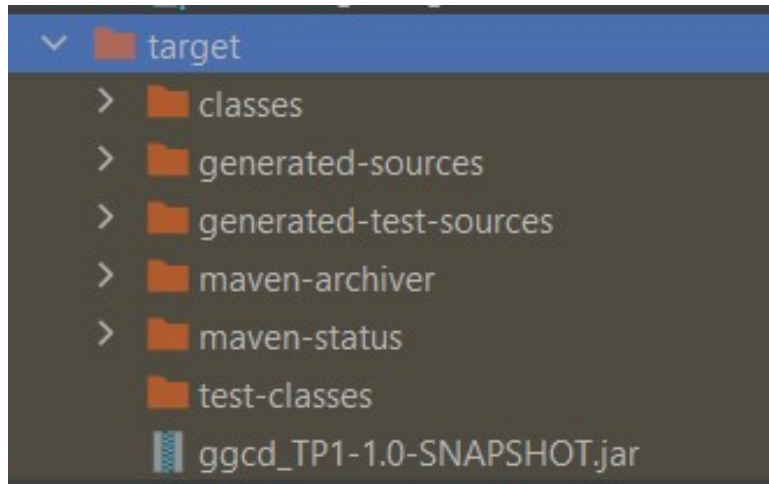


Figura A.4: Jar gerado pelo package

```
C:\Users\Utilizador\Desktop\GGCD\ggcd_TP1>gcloud dataproc jobs submit hadoop --jars=target/ggcd_TP1-1.0-SNAPSHOT.jar --class=ToParquet --cluster=ggcd --region=europe-west1 --
Job [c33ef0c16f6f4923a4554bf77f741cb34] submitted.
Waiting for job output...
/usr/lib/hadoop/libexec/hadoop-functions.sh: line 2400: HADOOP_COM.GOOGLE.CLOUD.HADOOP.SERVICES.AGENT.JOB.SHIM.HADOOPRUNCLASSSHIM_USER: invalid variable name
/usr/lib/hadoop/libexec/hadoop-functions.sh: line 2365: HADOOP_COM.GOOGLE.CLOUD.HADOOP.SERVICES.AGENT.JOB.SHIM.HADOOPRUNCLASSSHIM_USER: invalid variable name
/usr/lib/hadoop/libexec/hadoop-functions.sh: line 2460: HADOOP_COM.GOOGLE.CLOUD.HADOOP.SERVICES.AGENT.JOB.SHIM.HADOOPRUNCLASSSHIM_OPTS: invalid variable name
Exception in thread "main" java.lang.reflect.InvocationTargetException
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
    at java.lang.reflect.Method.invoke(Method.java:498)
    at com.google.cloud.hadoop.services.agent.job.shim.HadoopRunClassShim.main(HadoopRunClassShim.java:19)
Caused by: java.lang.NoClassDefFoundError: org/apache/parquet/avro/AvroParquetOutputFormat
    at ToParquet.main(ToParquet.java:131)
    ... 5 more
Caused by: java.lang.ClassNotFoundException: org.apache.parquet.avro.AvroParquetOutputFormat
    at java.net.URLClassLoader.findClass(URLClassLoader.java:382)
    at java.lang.ClassLoader.loadClass(ClassLoader.java:418)
    at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:352)
    at java.lang.ClassLoader.loadClass(ClassLoader.java:351)
    ... 6 more
ERROR: (gcloud.dataproc.jobs.submit.hadoop) Job [c33ef0c16f6f4923a4554bf77f741cb34] failed with error:
Job failed with message [Exception in thread "main" java.lang.reflect.InvocationTargetException]. Additional details can be found at:
https://console.cloud.google.com/dataproc/jobs/c33ef0c16f6f4923a4554bf77f741cb34?project=trabalho-pratico-1-ggcd&region=europe-west1
gcloud dataproc jobs wait 'c33ef0c16f6f4923a4554bf77f741cb34' --region 'europe-west1' --project 'trabalho-pratico-1-ggcd'
https://console.cloud.google.com/storage/browser/dataproc-staging-europe-west1-339387439795-lloyv4hj/google-cloud-dataproc-metainfo/55e8e58d-769b-43b5-8f66-b84cd447a14d/jobs/c33ef0c16f6f4923a4554bf77f741cb34/
gs://dataproc-staging-europe-west1-339387439795-lloyv4hj/google-cloud-dataproc-metainfo/55e8e58d-769b-43b5-8f66-b84cd447a14d/jobs/c33ef0c16f6f4923a4554bf77f741cb34/driveroutput
```

Figura A.5: Erro gcloud

```
C:\Users\Utilizador\.jdk\openjdk-15.0.2\bin\java.exe -Xmx
Did not find winutils.exe: {}
java.io.FileNotFoundException: Create breakpoint : Hadoop bin directory does not exist: C:\Users\Utilizador\Desktop\GGCD\hadoop-3.1.0\bin\bin -see https://wiki.apache.org/hadoop/Windows
    at org.apache.hadoop.util.Shell.getQualifiedBinInner(Shell.java:688)
    at org.apache.hadoop.util.Shell.getQualifiedBin(Shell.java:592)
    at org.apache.hadoop.util.Shell.<clinit>(Shell.java:689)
    at org.apache.hadoop.util.StringUtils.<clinit>(StringUtils.java:78)
    at org.apache.hadoop.conf.Configuration.getBootstrap(Configuration.java:1689)
    at org.apache.hadoop.security.SecurityUtil.setConfigurationInternal(SecurityUtil.java:104)
    at org.apache.hadoop.security.SecurityUtil.<clinit>(SecurityUtil.java:88)
    at org.apache.hadoop.security.UserGroupInformation.initialize(UserGroupInformation.java:312)
    at org.apache.hadoop.security.UserGroupInformation.ensureInitialized(UserGroupInformation.java:300)
    at org.apache.hadoop.security.UserGroupInformation.getCurrentUser(UserGroupInformation.java:567)
    at org.apache.hadoop.mapreduce.task.JobContextImpl.<clinit>(JobContextImpl.java:72)
    at org.apache.hadoop.mapreduce.Job.<init>(Job.java:152)
    at org.apache.hadoop.mapreduce.Job.getInstance(Job.java:195)
    at org.apache.hadoop.mapreduce.Job.getInstance(Job.java:214)
    at MovieCount.main(MovieCount.java:92)
Exception in thread "main" java.io.FileNotFoundException: Create breakpoint : AvroParquetResult (0 sistema não conseguiu localizar o ficheiro especificado)
    at java.base/java.io.FileInputStream.open0(Native Method)
    at java.base/java.io.FileInputStream.open(FileInputStream.java:211)
    at java.base/java.io.FileInputStream.<init>(FileInputStream.java:153)
    at java.base/java.io.FileInputStream.<init>(FileInputStream.java:108)
    at ToParquet.readFile(ToParquet.java:47)
    at ToParquet.getSchema(ToParquet.java:55)
    at MovieCount.main(MovieCount.java:73)
Process finished with exit code 1
```

Figura A.6: Erro local