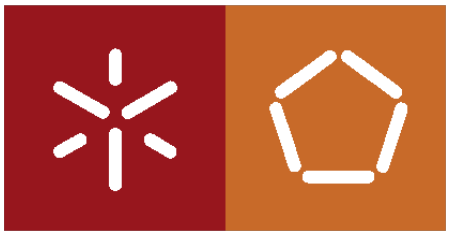

Universidade do Minho
MIEI - 4º ano - 2º semestre

Gestão de Grandes Conjuntos de Dados

Hadoop HDFS, HBase e MapReduce

Grupo 2:

A76516 - João Vieira
A74357 - António Lopes

23 de abril, 2020

Conteúdo

1 Introdução

O primeiro trabalho prático da U.C. de Gestão de Grandes Conjuntos de Dados, consiste na concretização e avaliação de tarefas de armazenamento e processamento de grandes quantidades de dados. Como tal, as tarefas serão implementadas com **Hadoop MapReduce**, com o auxílio do **Hadoop HDFS** para disponibilizar um sistema de ficheiros distribuído, e do **Apache HBase** como nossa base de dados *NoSQL*.

Os dados a utilizar são do *dataset* público do **IMDB**, e o objetivo será realizar todas as computações necessárias a estes dados, de modo a dar resposta às tarefas do enunciado, procurando também sempre o melhor desempenho em termos de implementação.

Este relatório documenta todo o processo de trabalho, desde as configurações iniciais do ambiente a utilizar, passando pelo desenvolvimento efetivo das tarefas, até aos resultados finais obtidos.

2 Configurações Iniciais

Nesta secção, está explicitado o ambiente e configurações iniciais para a execução das tarefas necessárias, que serão todas feitas localmente, numa só máquina. Para tal, é usada uma instalação *Docker* para obter o ambiente que permite realizar o projeto.

Este ambiente permite configurar o sistema de ficheiros distribuído do **Hadoop HDFS**, que armazena os ficheiros de dados, assim como outros ficheiros que se ache necessário. Também disponibiliza a configuração do **Apache HBase**, que se trata da nossa base de dados *NoSQL*, e que armazena todas as informações que pretendemos guardar.

2.1 Configuração Docker

Primeiro é necessário ter o serviço de *Docker* em execução na máquina. De seguida, apenas é necessário escrever na linha de comandos os seguintes dois comandos, um para realizar o *download* do ambiente pelo *GitHub*, e outro para executar o ambiente com *docker-compose* dentro da diretoria, respetivamente:

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

$ docker-compose -f docker-compose-distributed-local.yml up
```

2.2 Hadoop HDFS

Tendo já o nosso sistema de ficheiros distribuído, resta apenas carregar os respetivos ficheiros de dados, que serão lidos pelas tarefas do projeto.

Para isso, criou-se uma pasta no sistema de nome *input*, que será dedicada a armazenar exclusivamente os ficheiros do IMDB. Estando na diretoria do *docker-hbase*, executamos dois diferentes comandos, um para criar a pasta no **HDFS** e outro que será executado sempre que carregamos um novo ficheiro para o sistema:

```
$ docker run --network docker-hbase_default --env-file hadoop.env
bde2020/hadoop-base hdfs dfs -mkdir /input

$ docker run --network docker-hbase_default --env-file hadoop.env
--mount type=bind,source=/...path-to.../data,target=/data
bde2020/hadoop-base hdfs dfs -put /data/...file-name... /input
```

Este segundo comando, será executado quatro vezes para carregar no **HDFS** os quatro diferentes ficheiros de *input* que serão usados nas tarefas, em formato **bzip2**: **name.basics.tsv.bz2**, **title.basics.tsv.bz2**, **title.principals.tsv.bz2** e **title.ratings.tsv.bz2**.

2.3 Apache HBase

De modo a termos acesso à *shell* da nossa base de dados **HBase**, para realizar todas as verificações necessárias aquando da implementação de tarefas, apenas se executa o seguinte comando dentro da diretoria do *docker-hbase*:

```
$ docker run -it --network docker-hbase_default
--env-file hbase-distributed-local.env bde2020/hbase-base hbase shell
```

2.4 Dockerfile

Para se poder executar as tarefas no ambiente *Docker*, é necessário ter um **Dockerfile** que permite executar o código dentro do ecossistema configurado inicialmente.

O conteúdo do **Dockerfile** terá que ser o seguinte:

```
FROM bde2020/hadoop-base
COPY target/jarname.jar /
ENTRYPOINT ["hadoop", "jar", "/jarname.jar", "mainclassname"]
```

Finalmente, resta apenas definir as opções de execução do respetivo ficheiro:

```
--network docker-hbase_default
--env-file /...path-to.../docker-hbase/hadoop.env
--env-file /...path-to.../docker-hbase/hbase-distributed-local.env
```

3 Desenvolvimento das Tarefas

Tendo já o ambiente necessário à realização do projeto, esta secção irá abordar os objetivos propostos, assim como explicitar a implementação realizada para dar resposta a cada um deles.

Para uma melhor organização e leitura do código, dividiram-se os métodos de Map/Reduce consoante cada tarefa do projeto.

3.1 Carregamento dos Dados de Filmes

A primeira tarefa do projeto consistia no carregamento para uma tabela HBase, de um conjunto de dados de filmes, existentes no ficheiro **title.basics.tsv.bz2**. Este ficheiro contém informações por identificador de filme, como título, ano de estreia, géneros, duração, entre outros extra. Para além destes dados, foi aproveitada a existência do ficheiro **title.ratings.tsv.bz2** que contém informação sobre a avaliação, por cada *id* de filme, para enriquecer a quantidade de informação presente na tabela de filmes.

Sendo assim, foi criada uma instância de tabela HBase com o nome *movies* e a respetiva família de colunas, que para este caso será apenas uma, de nome *Details*.

Para realizar o processamento, criaram-se dois *mapper's*, um para cada ficheiro. O primeiro **MovieMapper**, processa as informações sobre os filmes. É feito um *split* em cada linha de texto, para obtermos os campos a inserir. Respetivamente, é criado um objeto *Put*, e adicionados os campos necessários, um por coluna, à família *Details* assim como o respetivo qualificador de cada. No final de cada iteração, o objeto *Put* é escrito para o contexto, e por último são inseridas as informações do filme na base de dados.

O segundo **RatingMapper**, que processa o ficheiro de avaliações, segue a mesma lógica do primeiro. Portanto, cada avaliação é adicionada à mesma tabela HBase, e à mesma família de colunas *Details* de cada filme respetivamente.

É de notar que, não temos nenhum *reducer* nesta tarefa, pois não são necessárias realizar computações aos dados, sendo estes apenas inseridos na tabela HBase.

Inicialmente, a execução desta tarefa estava dividida em dois *job's*, um por ficheiro a processar, mas posteriormente foi convertida para apenas um. Este único *job* recebe dois *input's* e aplica o *mapper* necessário a cada um, sendo que o *output* é do formato tabela. Concluiu-se que não havia uma diferença significativa em tempo de execução, entre as duas implementações.

3.2 Computação dos Dados de Atores

A segunda tarefa do projeto, tem como objetivo processar vários ficheiros e realizar computações sobre estes. Os ficheiros contêm diferentes dados sobre atores, que após processamento, são inseridos numa nova tabela HBase de nome *actors* e mais uma vez com só uma família de colunas de nome *Details*.

Posto isto, esta tarefa está dividida em várias sub-tarefas, que darão resposta às três alíneas do enunciado, assim como uma última sub-tarefa extra relativa ao conjunto de colaboradores de um ator. Cada uma destas sub-tarefas são realizadas com um ou mais *job's*, em que o resultado final é um encadeamento de todos para criar a página de cada ator.

3.2.1 Informação Básica

No que diz respeito à primeira sub-tarefa, era pedido para inserir informações como o nome, datas de nascimento e morte de um ator. Estes dados estão presentes no ficheiro **name.basics.tsv.bz2**.

A implementação desta interrogação, segue novamente a lógica que foi utilizada para responder à primeira tarefa, sobre os dados de filmes. Ou seja, temos um *job* que tem apenas um *mapper* que processa o ficheiro, e tem como formato de *output* uma tabela. O **ActorInfoMapper**, recolhe os dados necessários do ficheiro, e coloca num objeto *Put*, na família *Details*, cada um dos três campos e respetivo qualificador, por cada ator. No final temos a tabela *actors* contendo a informação básica de todos os atores.

Mais uma vez, este é um *job* sem *reducer* pois não foi necessário realizar qualquer computação sobre os dados.

3.2.2 Nº Total de Filmes

A seguinte sub-tarefa, pedia o nº total de filmes de cada ator ou seja, teríamos que contar o nº de ocorrências de cada ator para todas as entradas de filmes. Os dados estão presentes no ficheiro **title.principals.tsv.bz2**. Este ficheiro está organizado de maneira a que cada filme tem várias linhas, cada uma correspondente a um elemento do elenco e a respetiva informação.

Tendo isto em mente, foram usados dois *job's*, um para agrupar os filmes de cada ator, e outro para realizar a contagem e inserir no HBase:

O primeiro *job*, tem um *mapper*, **ActorMoviesMapper**, que organiza os dados em entradas do tipo **ator-filme**. O *reducer*, **ActorMoviesReducer**, tem o papel de iterar as várias entradas e concatenar os filmes de cada ator. Logo escreve para contexto entradas do tipo **ator-filmes**, em que os filmes são os *id's* de cada um separados por vírgula. Estes dados serão imprimidos para um ficheiro, que estará armazenado no **HDFS**, na diretoria **"/output/actorMovies"**.

Quanto ao segundo *job*, este tem a função de ler o ficheiro retornado pelo *job* anterior, e inserir para o HBase o nº de filmes de cada ator. Sendo assim temos apenas um *mapper*, **TotalMoviesMapper**, que para cada entrada, conta o nº de filmes e insere no HBase esta contagem para cada ator, com o qualificador *totalMovies*.

3.2.3 Três Filmes mais Cotados

A sub-tarefa dos Top3 filmes utiliza três *job's* encadeados.

O primeiro *job* tem a função de criar múltiplas entradas do tipo **ator-filme'#'rating** e escrever para um ficheiro na diretoria **"/output/top3List"**. Para isso, realiza um *shuffle join* a partir de dois ficheiros diferentes, logo utiliza dois *mapper's* e um *reduce*. O *mapper* **MovieActorMapper**, lê

o ficheiro criado na tarefa da alínea anterior, com a lista de filmes por ator, e cria várias entradas do tipo **filme-'A 'ator**. O segundo *mapper* **MovieRateMapper**, lê o ficheiro **title.ratings.tsv.bz2**, que contém informação sobre a avaliação de cada filme. Este irá criar entradas do tipo **filme-'R 'rating**. O *reducer* irá portanto ler os valores de cada *key*, e guardar tanto o ator como o *rate* do filme, realizando a verificação do caractere 'A' ou 'R'. Isto vai resultar na escrita das várias entradas no formato **ator-filme'#'rating**, como pretendido.

O segundo *job* desta tarefa, tem o objetivo de agrupar por ator, todas as entradas do formato anteriormente referido. Sendo assim o *mapper* **Top3GroupMapper**, lê o ficheiro escrito pelo primeiro *job* e escreve exatamente a mesma informação sem alterações. O *reducer* **Top3GroupReducer**, irá então agrupar todos os valores **filme'#'rating** existentes para um mesmo ator, separando-os novamente por vírgulas. Estas linhas serão escritas para um ficheiro na diretoria do **HDFS "/output/top3Group"**.

Por fim, o terceiro e último *job* percorre o ficheiro do *job* anterior, e realiza a computação dos três melhores filmes de cada ator através do *mapper* **Top3Mapper**. É feito um *split* sobre os valores **filme-rate** separados por um **'#**', e inseridos num *HashMap*. Depois são filtrados os três filmes com maior *rating* e guardados numa *LinkedHashMap* de maneira a estarem ordenados por decrescente. Esta estrutura é convertida para *String* e inserida no HBase, com o qualificador *top3*.

3.2.4 Conjunto de Colaboradores

Na última tarefa deste projeto, o objetivo passava por construir o conjunto de colaboradores de um ator, ou seja, todas as restantes pessoas que participaram nos mesmos filmes do ator. Para tal, a implementação desta alínea está dividida em quatro *job's* encadeados.

O primeiro *job* tem como fim criar um ficheiro, com entradas do tipo **filme-atores**, em que **atores** são os *id's* de cada ator participante, separado por vírgulas. Este ficheiro será guardado no **HDFS** na diretoria **"/output/movieActors"**. O *mapper* **MovieActorsMapper**, vai portanto receber como *input* o ficheiro **title.principals.tsv.bz2** e escrever para contexto várias entradas **filme-ator**. O *reducer* **MovieActorsReducer**, vai concatenar os vários atores de cada filme com vírgulas, e escreve para o ficheiro.

O segundo *job* trata-se de mais um *shuffle join*, que tem dois *mapper's* que vão ler respetivamente o ficheiro criado no *job* anterior, assim como o ficheiro **title.principals.tsv.bz2**. O primeiro *mapper* **MovieCollabMapper**, apenas lê o ficheiro e imprime os mesmos valores, acrescentando apenas o caractere **'C'** antes da lista de atores, resultando em entradas do tipo **filme-'C 'atores**. O segundo *mapper* **MovieActorEntryMapper**, vai ler o ficheiro que contém a informação de cada ator para um filme, e escrever entradas do tipo **filme-'A 'ator**. Definiu-se este código com caractere 'A' para identificar um ator, e o 'C' para identificar a lista de colaboradores, tudo para um mesmo filme. O *reducer* **ActorCollabReducer**, irá pois agrupar estes diferentes valores de um mesmo filme, e escrever entradas do tipo **ator-colabs**, em que **colabs** são os *id's* dos colaboradores, separados por vírgula. Estes dados serão inscritos novamente para ficheiro, na diretoria **"/output/actorCollab"**.

O terceiro *job* lê o ficheiro anterior, e tem a função de agrupar as várias entradas para um mesmo ator e torná-las única, concatenando assim os colaboradores de cada. Sendo assim o *mapper* **CollabGroupMapper**, apenas lê os dados e passa os mesmos para o *reducer* **CollabGroupReducer**, que concatena todos os valores de uma *key*. Este escreve para o contexto outra vez entradas do tipo **ator-colabs**, sendo que a diferença é que desta vez cada ator tem apenas uma entrada com a lista dos seus colaboradores. É criado um ficheiro com estes dados, na diretoria **"/output/actorCollabGroup"**.

Por fim, o último *job* vai ler o ficheiro do *job* anterior, realizar a computação final e inserir para o HBase a informação dos colaboradores de um ator. Como tal, apenas temos um *mapper* **CollabMapper**, que cria um *Set* e insere neste cada um dos colaboradores, que estão separados por vírgula. De seguida este *Set* é convertido para *String* e é feito um *Put* no respetivo ator, inserindo a informação do *Set* no qualificador *collaborators*.

4 Resultados

Nesta secção de resultados, demonstra-se o conteúdo das duas tabelas HBase criadas, *movies* e *actors*, através da *shell* do HBase.

4.1 Carregamento dos Dados de Filmes

Para obtermos a linha da tabela *movies* referente a um filme específico, por exemplo com o $id = tt0001211$, utilizamos o seguinte comando:

```
get 'movies', 'tt0001211'
```

Resultado:

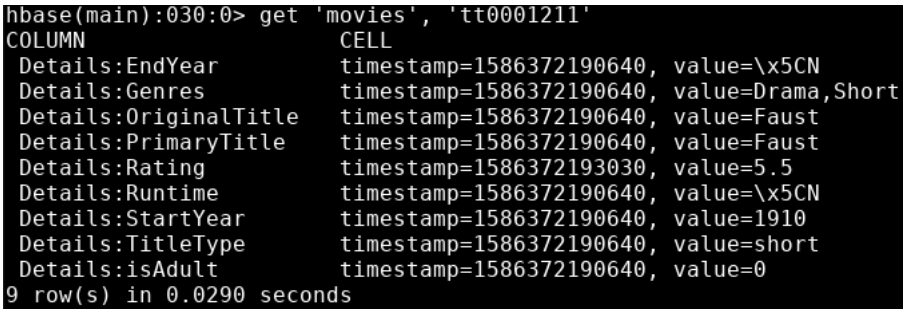

Figura 1: Resultado de um Filme no HBase

4.2 Computação dos Dados de Atores

No que diz respeito à tabela *actors* para obtermos um ator específico, por exemplo com o $id = nm0000305$, utilizamos o comando:

```
$ get 'actors', 'nm0000305'
```

Resultado:

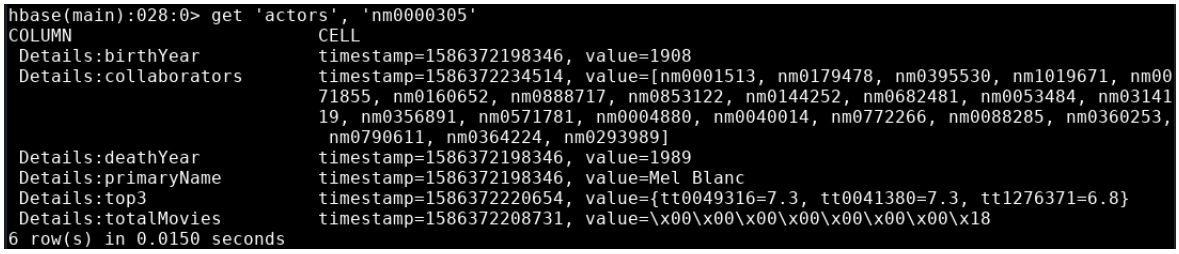

Figura 2: Resultado de um Ator no HBase

5 Conclusão

Finalizado todo o processo de implementação, é da opinião do grupo que a resolução do trabalho prático foi realizada com sucesso. Contudo, é claro que a computação de algumas tarefas poderia ser melhorada, tanto a nível de algoritmo como de desempenho e rapidez.

Um dos fatores a melhorar, depara-se com o facto de estarmos sempre a ler os dados de ficheiros do **HDFS**, e para algumas tarefas até acontece lermos o mesmo ficheiro de dados mais que uma vez, em *job's* diferentes. Também o facto de para praticamente todas as tarefas, utilizámos um ou mais ficheiros auxiliares, que guardamos no **HDFS**, para realizar as computações necessárias entre *job's*. Ou seja, estamos sempre a ler de ficheiros e a escrever para ficheiros, antes de inserir no **HBase**. Isto poderia ser colmatado por exemplo, lendo os dados a partir do **HBase** utilizando *mapper's* para tabela, em vez de fazer múltiplas leituras dos ficheiros. Apesar deste *bottleneck*, procurou-se sempre reduzir o número de *job's* de cada tarefa.

Apesar de alguma falta de prática e limitação quanto à implementação de tarefas *Map/Reduce* por parte do grupo, as soluções encontradas dão uma boa resposta aos objetivos do projeto, havendo sempre espaço para algumas melhorias.

A finalização deste trabalho, permitiu assim aprender e consolidar sobre esta forma de processamento de um grande número de dados, do *Map/Reduce*. Identificou-se que nem sempre é fácil lidarmos com todos os fatores que estes conjuntos de dados acarretam, assim como a dificuldade de encontrar a melhor solução dentro deste ambiente de processamento.