Gestão de Grandes Conjuntos de Dados

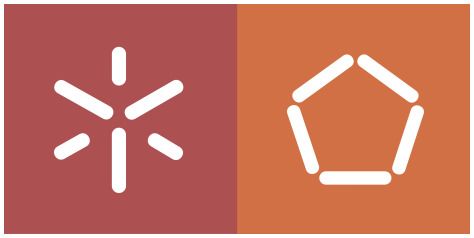
2º Trabalho Prático

-

Mestrado em Engenharia Informática
Universidade do Minho

Grupo nº 8	
PG41080	João Ribeiro Imperadeiro
PG41081	José Alberto Martins Boticas
PG41091	Nelson José Dias Teixeira
PG41851	Rui Miguel da Costa Meira

4 de junho de 2020



Conteúdo

Lista de Figuras

Capítulo 1

Introdução

Neste trabalho prático é requerida a concretização e avaliação experimental de tarefas de armazenamento e processamento de dados através do uso da ferramenta computacional *Spark* (*batch* e *streaming*). Por forma a realizar estas tarefas, são utilizados os dados públicos do *IMDb*, que se encontram disponíveis em:

https://www.imdb.com/interfaces/

Para além destes dados, é também utilizado um gerador de *streams*, baseado nos mesmos, que simula uma sequência de votos individuais de utilizadores. Este utensílio foi desenvolvido pelo docente desta unidade curricular e foi-nos disponibilizado através da plataforma *Blackboard*.

Ao longo deste documento vão também ser expostos todos os passos tomados durante a implementação das tarefas pedidas neste projeto, incluindo as decisões tomadas pelos elementos deste grupo a nível de algoritmos e parâmetros de configuração. Para além disso são ainda apresentadas todas as instruções que permitem executar e utilizar corretamente os programas desenvolvidos. Por fim, na fase final deste manuscrito, são exibidos os objetivos atingidos após a realização das tarefas propostas.

Capítulo 2

Implementação

Para a realização com sucesso deste trabalho prático, é solicitada a elaboração de três tarefas. Apresentam-se de seguida as mesmas:

1. Desenvolver uma componente de processamento de *streams* que produza os seguintes resultados:
   - ***Log***: armazenar todos os votos individuais recebidos, etiquetados com a hora de chegada aproximada ao minuto, em lotes de 10 minutos. Cada lote deve ser guardado num ficheiro cujo nome identifica o período de tempo;
   - ***Top3***: exibir a cada minuto o *top* 3 dos títulos que obtiveram melhor classificação média nos últimos 10 minutos;
   - ***Trending***: apresentar a cada 15 minutos os títulos em que o número de votos recolhido nesse período sejam superiores aos votos obtidos no período anterior, independentemente do valor dos votos.
2. Implementar uma componente de processamento em *batch* que permita realizar as seguintes tarefas:
   - ***Top10***: calcular o *top* 10 dos atores que participaram em mais títulos diferentes;
   - ***Friends***: computar o conjunto de colaboradores de cada ator (i.e., outros atores que participaram nos mesmos títulos);
   - ***Ratings***: atualizar o ficheiro *"title.ratings.tsv"* tendo em conta o seu conteúdo anterior e os novos votos recebidos até ao momento.
3. Escolher a configuração e a implementação que, para o mesmo *hardware*, permite receber e tratar o maior débito de eventos. Esta tomada de decisão deve ser devidamente justificada com recurso a resultados experimentais.

Nas próximas secções são expostas as implementações para cada uma destas tarefas bem como algumas sugestões alternativas que poderiam ser tomadas em consideração.

2.1 Configuração

Nesta secção do relatório são apresentadas as configurações relativas ao arranque do *cluster*, instanciando-se tanto as respetivas entidades (*master*, *worker1* e *worker2*) como as máquinas virtuais pertencentes à plataforma *Google Cloud*, e, ainda, a configuração genérica para a execução individual de cada exercício proposto.

2.1.1 Arranque do *cluster*

A configuração escolhida para a realização deste projeto coincide com a que foi sugerida pelo docente desta unidade curricular, isto é, o ***docker swarm***. Esta configuração permite não só tirar partido da ferramenta computacional *Google Cloud* como também possibilita o uso de plataformas como o *Hadoop HDFS* e o *Apache Spark*. Como seria de esperar, todos os ficheiros de *input* utilizados para atingir os objetivos traçados neste trabalho prático são armazenados no sistema *Hadoop HDFS*.

Exibe-se de seguida todos os passos de configuração associados ao *docker swarm*:

1. criação das instâncias relativas às 3 entidades intrínsecas à arquitetura do sistema, isto é, as entidades *master*, *worker1* e *worker2*:

```
docker-machine create \
               --driver google --google-project ferrous-aleph-271712 \
               --google-zone europe-west1-b \
               --google-machine-type n1-standard-4 \
               --google-disk-size=100 \
               --google-disk-type=pd-ssd \
               --google-machine-image \
      https://www.googleapis.com/compute/v1/projects/centos-cloud/global/images/centos-7-v20200309 \
               master
```

Figura 2.1: Configuração - Criação da instância associada à entidade *master*

```
docker-machine create \
               --driver google --google-project ferrous-aleph-271712 \
               --google-zone europe-west1-b \
               --google-machine-type n1-standard-4 \
               --google-disk-size=100 \
               --google-disk-type=pd-ssd \
               --google-machine-image \
      https://www.googleapis.com/compute/v1/projects/centos-cloud/global/images/centos-7-v20200309 \
               workerName
```

Figura 2.2: Configuração - Criação de uma instância *worker*

De salientar que a designação `ferrous-aleph-271712` corresponde ao identificador do projeto presente na plataforma *Google Cloud* de um dos elementos que compõem este grupo. Assim, esta denominação deve ser substituída pelo nome do projeto do utilizador. Relativamente à designação *workerName*, esta deve ser subtituída pelos nomes das entidades *worker* referidas no ponto acima (*worker1* e *worker2*).

2. configuração do *swarm* relativo às entidades *master*, *worker1* e *worker2*:

```
docker-machine ssh master sudo docker swarm init
```

Figura 2.3: Configuração - *swarm master*

```
docker-machine ssh workerName sudo docker swarm join --token \
               SWMTKN-1-5zfy2iio54tma997pnt96gq5095fimqn2hxr2a8j16ogq0n3c9-0kp6mi5iuj956gpl9sfccd5bo\
               10.132.0.8:2377
```

Figura 2.4: Configuração - *swarm worker*

3. ativação do ambiente da entidade *master*:

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

Figura 2.5: Configuração - Ativação do ambiente da entidade *master*

4. compilação do gerador de *streams*, isto é, o *streamgen*:

```
mvn package
docker build -t streamgen .
```

Figura 2.6: Configuração - Compilação do utensílio *streamgen*

5. arranque do sistema com a configuração *swarm* especificada:

```
docker stack deploy -c ../swarm-spark/docker-compose.yml mystack
```

Figura 2.7: Configuração - Arranque do sistema

É de realçar que no ficheiro *"docker-compose.yml"* encontra-se uma configuração *docker* semelhante à que foi utilizada no guião nº 8 desta unidade curricular. A única diferença presente no mesmo diz respeito à integração do gerador de *streams* (*streamgen*) desenvolvido pelo docente como um *container docker* do sistema. Assim, no momento do arranque do sistema, a ferramenta *streamgen* é convenientemente invocada, ficando à espera de novas conexões por parte dos utilizadores. Apresenta-se de seguida a configuração presente no ficheiro *"docker-compose.yml"* relativa ao *streamgen*:

```
streamgen:
    image: streamgen
    command: hdfs:///data/title.ratings.tsv 120
    env_file:
        - ./hadoop.env
    deploy:
        mode: replicated
        replicas: 1
        placement:
            constraints:
                - "node.role==manager"
```

Figura 2.8: Configuração - Ficheiro *"docker-compose.yml"* - *streamgen*

6. acesso à plataforma *Hadoop HDFS* por parte do utilizador:

```
docker run --network mystack_default --env-file ../swarm-spark/hadoop.env -it bde2020/hadoop-base \
       bash
```

Figura 2.9: Configuração - Acesso à plataforma *Hadoop HDFS*

Com a concretização dos 6 passos descritos acima, fica concluída a configuração do *docker swarm*.

2.1.2 Execução de tarefas

Para a execução dos 3 exercícios descritos na secção relativa à implementação, é disponibilizado um ficheiro com a designação *Dockerfile*. Neste é especificado o nome da classe relativa à proposta que se presente executar, procedendo corretamente à sua invocação. Como tal, em primeiro lugar é preciso desempenhar a seguinte instrução:

```
mvn package
```

Figura 2.10: Configuração - Obtenção do ficheiro *jar* do projeto

De seguida, invoca-se o *dockerfile* em causa. Expõe-se agora o seu conteúdo:

```
FROM bde2020/spark-base:2.4.4-hadoop2.7
COPY target/TP2-1.0-SNAPSHOT.jar /
ENTRYPOINT ["/spark/bin/spark-submit", "--class", "package.className", "--master", \
            "spark://spark-master:7077", "/TP2-1.0-SNAPSHOT.jar"]
```

Figura 2.11: Configuração - *Dockerfile*

A este ficheiro estão associadas alguma opções de execução que garantem uma comunicação fidedigna com o sistema inicialmente declarado. Eis as mesmas:

```
-p 4040:4040 --network mystack_default --env-file ../swarm-spark/hadoop.env
```

Figura 2.12: Configuração - Opções de execução do ficheiro *Dockerfile*

De salientar que é escolhida a opção *master* no campo relativo ao *docker machine* para, mais uma vez, garantir uma configuração válida do sistema em causa.

2.2 1ª Tarefa

Na 1ª tarefa deste projeto é pedido o desenvolvimento de uma componente de processamento de *streams*. Nesta é solicitada a realização de vários exercícios com diferentes caraterísticas. Como tal, apresenta-se de seguida as respetivas implementações e, ainda, possíveis alternativas nas suas concretizações.

2.2.1 *Log*

Neste exercício, tal como foi indicado no início deste capítulo, é imposto o armazenamento de todos os votos individuais recebidos com a indicação da hora de chegada

aproximada ao minuto, em lotes de 10 minutos, sendo que cada lote é guardado com o nome relativo ao período de tempo em causa.

Dito isto, para proceder ao tratamento da aproximação da hora recebida ao minuto, foi implementada uma função para o efeito. Optou-se por não só guardar a hora referida como também a data em questão, exibindo, desta forma, um maior detalhe da informação recebida. Uma vez implementada esta função, foi utilizado o método *transform* que faz uso da noção de tempo. Com esta vertente, é possível associar a cada *RDD* do conjunto de dados a hora, aproximada ao minuto, em que este foi processado.

Por fim, com os aspetos computacionais mencionados acima, resta armazenar os dados recebidos pela ferramenta *streamgen* com recurso à definição de uma janela com 10 minutos de duração e 10 minutos de deslocamento. Os *RDD*'s em causa são guardados na diretoria `Log/Lot{i} - dd-MM-yyyy HH-mm`, sendo que `i` diz respeito ao número do lote (que começa no número 1 e que é constamente incrementado) e `dd-MM-yyyy HH-mm` refere-se ao padrão do período temporal.

De maneira a guardar apenas um ficheiro (*part-00000*), a todos os *RDD*'s resultantes foi aplicado o método *coalesce(1)*. Esta última função, quando comparada com a *repartition()*, não necessita de efetuar um *shuffle* completo dos dados. Para além disto, esta abordagem foi considerada nesta implementação dado que no exercício *Ratings* (*batch*) será necessário realizar a leitura do ficheiro mencionado. Consequentemente, em vez de operar várias leituras de ficheiros, efetua-se apenas uma leitura, permitindo, assim, uma maior eficiência.

2.2.2 *Top3*

Para este exercício é necessário recolher os novos votos e calcular as classificações médias dos últimos dez minutos, a cada minuto.

De forma a recolher e tratar os novos votos gerados pelo *streamgen*, mapeia-se cada um dos mesmos para um par em que a chave é o identificador do filme e o valor um outro par: o voto e o número 1.

Desta forma, é possível, depois, utilizar o método *reduceByKeyAndWindow*, de forma a que, para a duração (10) e deslizamento (1) definidos para a janela, seja possível realizar as operações necessárias. Com o método anterior, é possível definir duas funções para aplicar aquando do deslizamento da janela, agrupando previamente os dados pela sua chave (identificador do filme): a primeira diz respeito ao que fazer com os novos dados que chegam (novos votos) e a segunda diz respeito ao que fazer com os dados que deixam de ficar dentro da duração da janela (votos de há mais de dez minutos).

Dito isto, a primeira função deve somar ambos os componentes do par gerado (voto, 1), por forma a obter um novo par com a soma total dos votos e o número total de votos, para cada filme. A segunda função deverá subtrair os mesmos, de forma a que o votos gerados há mais de dez minutos deixem de ter influência nos resultados.

Por fim, mapeiam-se os resultados para um par em que a chave mantém o identificador do filme e o valor passa a ser a divisão entre os componentes do par que representava o valor anteriormente. Assim, o novo par associará a classificação média dos últimos dez minutos ao identificador do filme.

Os últimos passos serão juntar os resultados da computação anterior com uma primeira computação que extrai os títulos dos filmes do ficheiro "*title.basics.tsv*" e os associa ao seu identificador. Por fim, são exibidos os resultados obtidos. A junção implica o uso do método *transformToPair* de forma a que se proceda à junção de

cada *RDD*, resultante do processamento de *stream* explicado acima, ao *JavaPairRDD* explicado anteriormente.

Ao resultado desta junção aplica-se o método *forEachRDD* e calcula-se, com o método *top*, os 3 filmes com melhor classificação média. Para isso foi necessário explicitar um novo comparador, uma vez que o tipo de dados *Tuple2* não é serializável. Por fim, imprime-se o resultado. Esta impressão deve ocorrer uma vez a cada minuto.

2.2.3 *Trending*

O último exercício da primeira tarefa foi talvez o mais desafiante de todos, pelo facto de ser necessário comparar os novos dados com dados anteriores. Para resolver o problema, foi decisivo o uso do método *mapWithState*.

Começou-se da forma mais simples, retirando-se as informações pertinentes do ficheiro "*title.basics.tsv*", para mais tarde serem usadas.

Passando ao processamento de *stream*, este, inicialmente, foi muito semelhante ao do exercício anterior, sendo que, em vez de um par associado ao identificador do filme, foi apenas associado o inteiro 1. Desta forma, cada voto terá o mesmo peso, como é exigido, já que se querem comparar números de votos e não classificações. O método *reduceByKeyAndWindow* foi aplicado da mesma forma que no exercício anterior, com a alteração dos valores de duração e deslizamento da janela, agora 15 em ambos os casos.

A diferença marca-se depois, com o *output* gerado anteriormente a alimentar o método *mapWithState*, por forma a ser possível comparar os dados (votos) dos últimos 15 minutos com os gerados nos 15 minutos anteriores a estes. Neste último método, opera-se sobre um três variáveis: a chave `k`, o novo valor `v` (pode existir ou não) e o estado anterior `s` (pode existir ou não; representa a quantidade de votos dos 15 minutos anteriores aos em questão). É declarada uma variável booleana `trending`. Se `v` não existir, `trending` passa a *false* e o estado `s` é apagado. Se `v` existir mas `s` não existir, então `trending` passa a *true* e `s` assume o valor de `v`. Caso existam as duas variáveis, então as mesmas são comparadas e `trending` fica com o resultado dessa comparação (`v` > `s`). Independentemente do resultado desta última comparação, `s` assume o valor de `v`. Por fim, é retornado um par que associa `trending` à chave `k`.

De forma a simplificar a junção desta informação com a retirada do ficheiro "*title.basics.tsv*", é ainda executado o método *mapToPair*, de forma a que o resultado de toda esta computação seja um *JavaPairDStream*.

Posteriormente, segue-se à junção das informações. Para isto, procede-se da mesma forma que no exercício anterior, sendo que ainda se aplica uma filtragem aos *RDD*'s que tenham os valores de `trending` a falso. Assim, teremos apenas os filmes que estão *trending*, ou seja, que tiveram mais votos no último período de 15 minutos, quando comparados com o período anterior. Por último, antes de imprimir, procede-se ainda ao mapeamento do resultado anterior, de forma a descartar os identificadores dos filmes e os valores booleanos, ficando-se apenas com os nomes dos filmes que estão em alta.

Resta então apenas a impressão desses nomes. Para isso, aplica-se o método *forEachRDD* ao resultado da execução anterior e aplica-se o método *collect* a cada *RDD*.

2.3 2ª Tarefa

Na 2ª tarefa deste trabalho prático é proposta a implementação de uma componente de processamento em *batch*. Nesta é solicitada a realização de 3 exercícios com

diferentes caraterísticas e, como tal, foram adotapadas abordagens distintas para cada um deles. Divulgam-se, de seguida, as respetivas implementações.

2.3.1 *Top10*

Nesta subtarefa é pedido o cálculo dos 10 atores que participaram em mais filmes distintos. Ênfase para distintos, uma vez que, no caso das séries, muitos dos episódios apresentam o nome da série, o que poderia levar a um erro na implementação caso estes fossem tomados em conta.

Durante o processamento inicial do ficheiro *"title.principals.tsv"* é, tal como seria de esperar, ignorado o respetivo cabeçalho. Posteriormente, é extraída, linha após linha, a informação pertinente do mesmo, isto é, os identificadores do filme e do ator em questão, agrupando os dados pela segunda componente. Esta última ação é efetuada com recurso à chamada do método *groupByKey*. Com esta computação, obtém-se para cada ator a lista de filmes em que este participou. Atendendo ao resultado exigido neste exercício, basta, nesta etapa do processamento, efetuar a contagem dos filmes associados a cada ator. Para tal, mapeia-se o resultado da agregação anterior de forma a associar a cada ator, o número de filmes em que o mesmo participou. Por razões de ordenação e seleção dos atores mais ativos, trocou-se a ordem mencionada acima, sendo que, desta forma, temos o número de filmes como chave e o identificador do ator como valor.

A recolha dos 10 atores que participaram em mais filmes é finalizada com a chamada do método *top*. Esta função permite extrair os k maiores registos de um *RDD*, segundo uma determinada ordem. Para este exercício, houve a necessidade de implementar um comparador explícito, numa classe à parte, dado que o tipo de dados *Tuple2* não é, por definição, serializável.

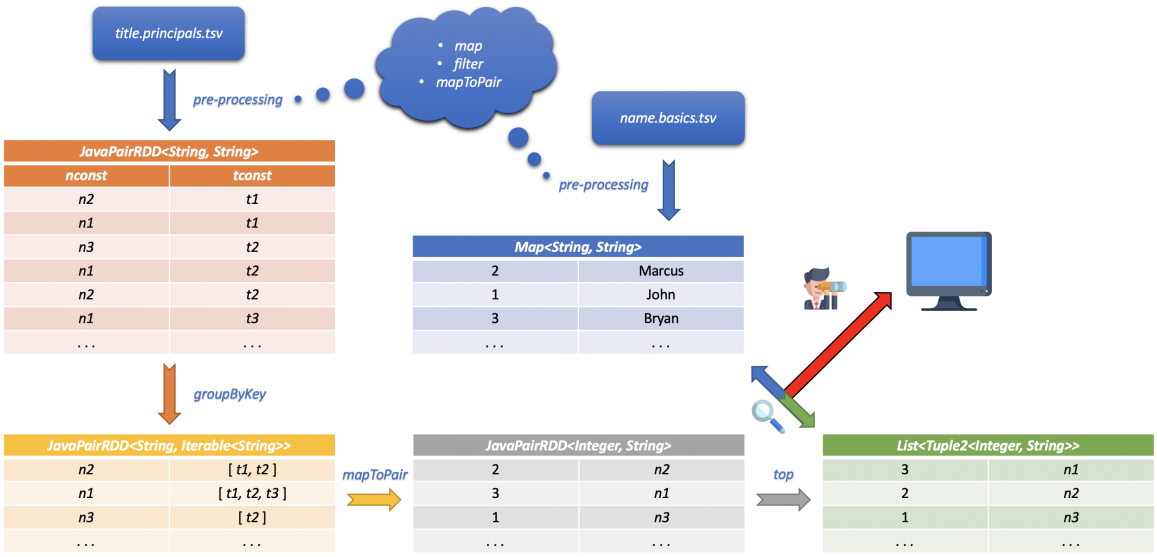


Figura 2.13: 2ª Tarefa (*batch*) - Esquema do processamento relativo à subtarefa *Top10*

2.3.1.1 Alternativa

Uma forma alternativa de resolver este exercício seria, na última fase do processamento, utilizar o método *take* em detrimento da função *top*. Esta escolha não foi tomada em consideração na implementação uma vez que o primeiro método necessita previamente que a informação esteja devidamente ordenada. Esta ordenação teria

de ser realizada com a invocação do método *sortByKey(false)*, colocando a contagem dos filmes em que cada ator participou de forma decrescente. Este último facto representa uma ineficiência no cálculo do resultado pretendido uma vez que é efetuada a ordenação completa da informação em causa e, para além disso, realiza-se desnecessariamente um passo computacional extra.

2.3.2 *Friends*

Neste exercício é requerido a computação do conjunto de colaboradores associado a cada ator, ou seja, o grupo dos atores que particapam nos mesmos filmes.

Durante o processamento inicial do ficheiro *"title.principals.tsv"* é, tal como seria de esperar, ignorado o respetivo cabeçalho. Posteriormente, é extraída, linha após linha, a informação pertinente do mesmo, isto é, os identificadores do filme e do ator em questão, agrupando os dados pela primeira componente. Esta última ação é efetuada com recurso à chamada do método *groupByKey*.

De forma a obter o resultado solicitado nesta subtarefa, é necessário, nesta fase da computação, proceder à realização de uma operação denominada por produto cartesiano. Nesta operação computa-se, num dado momento, todos os pares dos atores que colaboraram num determinado filme, passando a existir apenas pares de atores, com a chave a representar um ator e o valor um dos seus colaboradores. De realçar que estes pares são guardados num *Set*, de forma a eliminar repetições.

Uma vez realizado este cálculo, é invocado novamente o método *groupByKey* de forma a obter o resultado pretendido, isto é, o conjunto de colaboradores para cada ator.

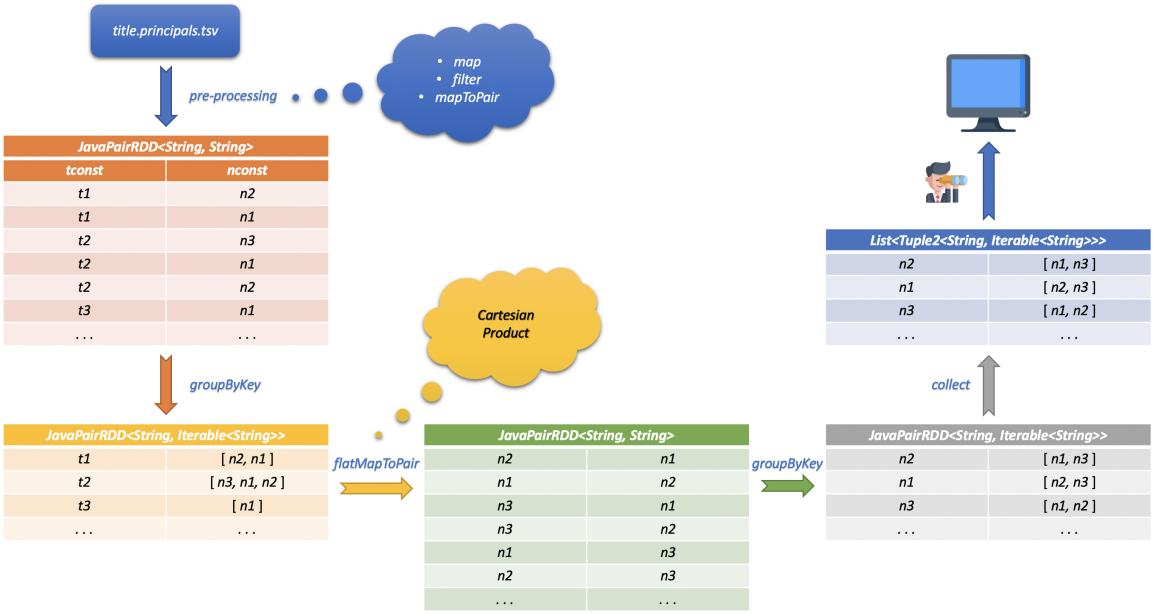


Figura 2.14: 2ª Tarefa (*batch*) - Esquema do processamento relativo à subtarefa *Friends*

2.3.3 *Ratings*

Este último exercício da segunda tarefa divide-se em três fases, tratar os votos, aplicar as alterações provocadas pelos mesmos e, por fim, atualizar o ficheiro "*title.ratings.tsv*".

Assim, foram desenvolvidas duas funções auxiliares: a primeira trata de coletar os votos apresentados nos ficheiros de *Log* produzidos pelo primeiro exercício da primeira

tarefa, passando-os para um novo ficheiro de texto; a segunda atualiza o ficheiro "*title.ratings.tsv*", gerando um novo, "*title.ratings.tsv.new*".

A função principal começa por a primeira das funções descritas anteriormente.

De seguida, carrega as informações de *rating* dos filmes, aplicando-lhe a filtragem que permite remover o cabeçalho. A isto, aplica uma alteração, com recurso ao método *mapToPair*, de forma a guardar apenas o identificador do filme associado ao par: multiplicação entre a classificação média obtida até agora e o número total de votos; número total de votos. Esta multiplicação permite obter uma aproximação da soma de todos os votos, útil para calcular a nova classificação média.

Posto isto, passa-se à fase de carregar os novos votos, por forma a concretizar as alterações necessárias. Utilizando o método *textfile* do `JavaSparkContext`, é possível ler e manipular o ficheiro de texto gerado pela primeira função auxiliar, como mencionado anteriormente. Assim, carregam-se os novos votos, agrupando os mesmos por identificador do filme. Deste modo, obtém-se, para cada filme, uma lista de todos os novos votos. A este resultado, é aplicada uma última transformação: gera-se um par com a soma de todos os votos e o número total de votos. Este par fica associado ao identificador do filme.

De forma a cruzar as informações dos novos votos com as médias já existentes, é realizado um *left outer join*. Desta operação obtemos, associado ao identificador de um filme, os valores extraídos na primeira fase e os valores extraídos na segunda, caso existam. De seguida, somam-se os valores da soma total dos votos e do número total de votos, obtendo-se um par com os novos totais associado ao identificador do filme correspondente. De realçar que, para os filmes que não tiveram novos votos, não é feita esta operação, mantendo-se os valores iniciais.

Por fim, é dividida a soma total dos votos pelo número total de votos, de forma a obter-se a nova classificação média, sendo que se preserva o número total de votos. É ainda aplicado o método *map*, no sentido de cada par ser convertido numa *string* que irá ser uma linha do novo ficheiro.

A última etapa passa por guardar todas as alterações, pelo que é chamada a segunda das funções descritas acima. Esta função trata de criar um novo ficheiro, "*title.ratings.tsv.new*", onde guarda todos os filmes e respetivos classificações e número de votos, mantendo o formato do ficheiro anterior.

2.4 3ª Tarefa

Para esta tarefa foram realizados alguns testes, com diferentes configurações, de forma a averiguar quais destas permitiam obter o melhor desempenho. Os testes em causa foram efetuados em máquinas com 4 *CPUs* e 15 *GB* de memória.

Para o caso da primeira tarefa (*stream*), observou-se o uso dos recursos computacionais, sendo que, tanto o uso de *CPU* como o de memória se mantiveram reduzidos. Consequentemente, conseguiu-se inferir que, de facto, a utilização de apenas 1 *CPU* e 2 *GB* de memória deveriam ser suficientes para o efeito. De realçar também que, para este caso, foi testado unicamente o exercício *Top3* por ser o mais conveniente dos da componente *stream*.

Relativamente à segunda tarefa (*batch*), o tempo de execução foi o critério de desempate. Foi testado o exercício *Top10*, sendo que o aumento de memória não teve efeito no tempo de execução, pelo que um mínimo de 2 *GB* de memória deverá ser o mais apropriado. Quanto à quantidade de *CPUs*, os resultados já foram distintos. A utilização de um maior número de *CPUs* traduziu-se num melhor desempenho. Como tal, após realizar os testes, os elementos deste grupo suspeitam que o uso de 4 *CPUs* seria o mais adequado.

Para além disso, testou-se ainda o exercício *Friends* relativo à componente *batch*. Neste caso, a memória parece ter um impacto significativo no tempo de execução, pelo que se aconselha o uso de, pelo menos, 8 *GB* de memória.

Também foi testado o impacto do tamanho dos blocos de memória aquando do carregamento, para o sistema *Hadoop HDFS*, dos diferentes ficheiros utilizados. Estes últimos testes indicaram que o tamanho dos blocos de memória não têm grande influência na execução dos algoritmos desenvolvidos neste projeto.

Capítulo 3

Conclusão

Para concluir, neste trabalho de cariz prático tivemos a oportunidade de aprofundar os nossos conhecimentos ao nível do processamento de dados em grande escala, através do uso da *framework Spark*, nas suas vertentes de *batch* e *streaming*.

Em comparação com o primeiro trabalho prático, onde se abordava o paradigma *MapReduce*, esta ferramenta apresenta diversas vantagens, nomeadamente a maior flexibilidade nas operações efetuadas sobre os dados. Mais, foi possível observar que o tempo de resposta a eventos por parte da estratégia *streaming* é bastante mais rápido do que o que se verifica na arquitetura *MapReduce*. Outros dos grandes benefícios na utilização deste utensílio é a capacidade do mesmo em processar grandes quantidades de informação sem os mesmos estarem armazenados localmente e, ainda, o facto desta estratégia admitir o processamento de uma sequência de eventos infinita.

Para além disto, tivemos a oportunidade de interagir com uma plataforma de *cloud*, mais propriamente a *Google Cloud (GCP)*. Esta foi utilizada dinâmica e exaustivamente tanto na configuração do sistema adotado como na fase de testes sobre o mesmo. Consequentemente, foi possível compreender de forma precisa a utilidade de um sistema deste tipo.

Por último, podemos afirmar que os objetivos inicialmente traçados foram cumpridos na íntegra, uma vez que todos os exercícios propostos foram resolvidos com sucesso e, na leitura do grupo, eficientemente.

Apêndice A

Observações

- Documentação *Java* 8:
  https://docs.oracle.com/javase/8/docs/api/
- *Maven*:
  https://maven.apache.org/
- *Docker*:
  https://www.docker.com/
- *Apache Spark*:
  https://spark.apache.org/
- *Apache Hadoop*:
  http://hadoop.apache.org/