Sistemas para Processamento de Big Data
Projeto 1

Ana Beatriz Breia
MAEBD - Nº 61877

Catarina Afonso
MAEBD - Nº 62206

João Funenga
MAEBD - Nº 61635

1. Introdução

Este projeto foi elaborado no âmbito da cadeira de Sistemas para Processamento de Big Data e pretende documentar todo o trabalho realizado ao longo do semestre. Para tal, foi fornecido um *dataset* que contém a informação diária sobre todos os monitores do Environmental Protection Agency (EPA), e um *dataset* auxiliar que contém a informação das coordenadas geográficas (latitude e longitude) de todos os estados dos EUA, com o objetivo de responder a cinco queries, recorrendo às seguintes tecnologias: MapReduce, Spark, SparkDataFrame, SparkSQL e Hive. Assim, as próximas secções são reservadas à resolução dos exercícios propostos, utilizando as tecnologias referidas.

2. Procedimentos Gerais

Visto que a primeira linha dos dois *datasets* contém o nome das colunas, foi necessário removê-la. Assim, aquando a explicação dos exercícios o leitor deve assumir que esta primeira linha já foi removida.

Para as *queries* em que foi necessário utilizar o *dataset* auxiliar (em particular nas questões 4 e 5), recorreu-se à biblioteca *'pickle'*, para guardar e ler as informações do *dataset* referido ao mesmo tempo que o *dataset* principal.

Para a resolução deste projeto e para a contagem dos tempos de execução utilizou-se apenas uma versão reduzida do *dataset* principal (epa_hap_daily_summary-small.csv).

3. MapReduce

Para alguns *jobs* de *Map-Reduce* que envolvam ordenar o *output* do *Mapper*, utilizou-se argumentos extra do comando para a execução do hadoop, que consistem em definir o tipo de *keys* (inteiros ou *strings*) e se a ordenação é descendente ou ascendente.

Q1) Which states have more/less monitors? (Rank states!)

Para a resolução desta *query* recorremos a duas etapas de *Map-Reduce*. Na primeira fase, definimos como *'key'* o nome do estado e como *'value'* um tuplo correspondente às coordenadas desse monitor. Deste modo, ao definirmos a *key* como o estado sabemos que todas as linhas que digam respeito ao mesmo estado serão enviadas para o mesmo *reducer* e, como consequência, poderemos somar o número de monitores para cada estado paralelamente em cada *reducer*, aumentando assim a eficiência. Neste caso, o *value* também será único visto que assumimos apenas existir um monitor por coordenadas e assim facilitar o cálculo no próximo passo. O *reducer* eliminará as ocorrências duplicadas (correspondentes às mesmas coordenadas) para que apenas contemos os monitores diferentes. Na segunda fase, o *mapper* e o *reducer* apenas irão trocar a ordem do *output* (estado, nº de monitores), tendo como *key* o número de monitores para que os estados sejam organizados de forma descendente relativamente a este número.

Q2) Which counties have the best/worst air quality? (Rank counties considering pollutants' level!)

Para esta *query*, a abordagem utilizada foi a mesma que na primeira pergunta, i.e., o segundo *Map-Reduce* apenas troca a ordem do *key*/*value* para que no final tenhamos o *output* ordenado. Em relação à primeira parte, a *key* definida no *mapper* foi "Estado|County" visto que existiam alguns *counties* com o mesmo nome em estados diferentes, e para o *value* da poluição decidimos usar a coluna "*Arithmetic Mean*". No *reducer*, calculámos a média destes *values*, ficando assim com um valor final de poluição por cada "Estado|County".

Q3) Which states have the best/worst air quality in each year? (Rank states per year considering pollutants' level!)

Para esta *query* necessitámos de dois *mappers* e dois *reducers*. O primeiro *mapper* apenas vai devolver como *output* uma *key*, constituída pelo ano (que corresponde à coluna "Date Local") e pelo estado (coluna "State Name"), e um *value* que será a qualidade do ar ("Arithmetic Mean"). Assim, o primeiro *reducer* vai receber como *input* tuplos ordenados pelo ano e pelo estado. Como existe a possibilidade de haver mais do que um registo da qualidade do ar, no mesmo estado e durante o mesmo ano, a função do *reducer* é calcular a média destas qualidades do ar (o que é fácil de se fazer uma vez que os anos e os estados já vêm agrupados do *mapper*). Assim, o *output* deste *reducer* são tuplos da forma (ano, estado, qualidade do ar média), sendo que agora a função do segundo *mapper* é apenas reorganizar este tuplo de forma a que quando "enviados" para o segundo *reducer* sejam ordenados por ano e qualidade do ar média, isto é, a *key* será constituída por estes dois atributos. Note-se que a qualidade do ar média é um *"float"* pelo que, para a função

sort funcionar da forma desejada, utilizamos a estratégia de calcular a diferença de um número superior a todas as qualidades de ar (por exemplo, 99999) e a própria qualidade do ar média. Por fim, o último *reducer* vai apenas voltar a transformar a qualidade do ar média no valor original e vai imprimir os tuplos organizados na forma pedida, por ordem decrescente de qualidade do ar.

Q4) For each state, what is the average distance (in km) of the monitors in that state to the state center? For simplicity, assume that 1 degree of latitude or longitude equals to 111 km. (Monitor dispersion per state!) Para esta pergunta necessitámos de fazer comparações com um ficheiro auxiliar que contém as coordenadas relativas aos estados americanos. Assim, começando pelo ficheiro "usa_states.csv", utilizámos apenas uma sequência de *Map-Reduce* para guardar num ficheiro auxiliar os valores das coordenadas médias de cada um dos estados presentes neste ficheiro para que depois os possamos comparar com as coordenadas dos monitores, visto este ser um ficheiro bastante pequeno e que apenas servirá para comparações. Posto isto, utilizámos três *map-reducers*. Para o primeiro *mapper*, formámos um tuplo intermediário cuja *key* atribuída corresponde ao nome do estado do monitor e o *value* corresponde às coordenadas do mesmo. Através dos pares emitidos pelo primeiro *mapper*, removemos os pares com coordenadas repetidas através do*reducer*. No segundo *mapper*, fizémos *load* do ficheiro *pickl*e guardado e a partir deste, calculámos a distância euclidiana entre os pontos do monitor do tuplo e os pontos guardados desse ficheiro. Através do segundo *reducer*, agrupámos para o mesmo estado as distâncias calculadas numa lista. O último *mapper* e *reducer* serviram para ordenar pela distância.

Q5) How many sensors there are per quadrant (NW, NE, SE, SW) in each state? To answer this question, you should approximate each state's area to a rectangle as defined in the file "usa_states.csv", and divide that area in 4 quadrants (NW, NE, SE, SW). (Count monitors per sate quadrant!)

A ideia para a resolução desta pergunta é começar por guardar os valores da latitude e longitude médias para cada um dos estados (ponto médio) e depois iremos comparar com as coordenadas de cada um dos monitores para determinar em qual dos quadrantes do estado é que se encontram. Primeiramente, iremos fazer o mesmo que na pergunta anterior com o ficheiro auxiliar. Agora, para o ficheiro principal das observações dos monitores, utilizámos 3 *Map-Reduces*. No primeiro, o *mapper* define a *key* como o nome do estado (para que depois possa ser comparado com o ficheiro auxiliar) e no *value*, um tuplo com as coordenadas [1]. No *reducer*, apenas serão removidas as repetições como feito na primeira pergunta. No segundo, no que toca ao *mapper*, iremos fazer as comparações entre as coordenadas do monitor com as coordendas do estado em que este se encontra para determinar o quadrante. Para determinarmos o quadrante, assumimos que cada estado tem um formato retangular para que as comparações sejam diretas, (e.g. caso a latitude do monitor seja inferior ou igual à latitude média desse estado e a longitude seja inferior ou igual à longitude média do estado, este monitor encontra-se no quadrante superior esquerdo (*NW*)) [4]. Para o *reducer*, repetiremos o processo de agruparmos os "Estados—Quadrantes" removendo as repetições e contarmos o número de monitores que se encontram em cada um. Para o último *job* de *Map-Reduce*, faremos a troca entre o *value* e a *key* para que fique organizado de forma descendente relativamente ao número de monitores.

4. Spark

Q1) Depois de importar o ficheiro para um *Spark Context*, e fazer as filtragens iniciais, fazemos um *map* para guardar os tuplos em que a *key* é o código do estado e o *value* o conjunto de coordenadas dos monitores pertencentes a esse estado, obtido através da transformação *reduceByKey*. Note-se que este é definido como um conjunto para que, ao adicionarmos as coordenadas dos monitores, não tenhamos repetições e possamos utilizar o número de monitores distintos, por estado, para ordar os tuplos pelo tamanho destes conjuntos. Para finalizar, mostramos as transformações feitas utilizando a ação *collect()*, que retorna todos os valores do RDD final.

Q2) Depois das filtragens iniciais, criamos um RDD em que a primeira posição dos tuplos corresponde a "Estado|County" e o valor à poluição média para esse dia (em lista). Depois disto, fazemos um *reduceByKey* que agrupa os valores com a mesma *key* (adicionando à lista). De seguida, podemos fazer outro *map* para calcular a qualidade média do ar para cada um dos *counties*, fazendo a soma de todos os valores de poluição sobre o número de monitores (para cada *county*). Finalmente faremos também a ordenação decrescente da qualidade do ar, usando um *sortBy*.

Q3) Para esta *query* cria-se tuplos, constituídos pelo ano ("Date Local"), pelo estado ('State Name') e por uma lista para as qualidades do ar ("Arithmetic Mean"). Para o cálculo dos valores médios das qualidades do ar utiliza-se a transformação *reduceByKey*, que adiciona à lista todos os valores com a mesma *key* (ano,estado), e de seguida faz-se uma transformação que calcula a média dos *values*. Posteriormente, aplicamos um *map* ao nosso RDD, de maneira a transformar tuplos na forma (ano,(estado,qualidade de ar)) e de seguida utilizamos a função *sortBy* para ordenar os tuplos por qualidade de ar. Para que tenhamos os tuplos organizados de forma crescente de qualidade de ar, dado o mesmo ano, é necessário aplicar a este RDD um *groupByKey* (note-se que a *key* corresponde ao ano) e para que os valores fiquem organizados numa lista utiliza-se a função *mapValues(list)*. Por fim, basta ordenar todos os tuplos pela *key*, (ano) utilizando o *sortByKey*, e fazer *print* de cada ano e dos respetivos valores, que correspondem aos pares (estado, qualidade de ar) já ordenados como pedido.

Q4) Depois de fazer as filtragens iniciais, atribuiu-se ao RDD do primeiro *dataset* um par, em que a *key* corresponde ao *StateName* e o *value* ao tuplo (Latitude,Longitude). Para o segundo *dataset*, a *key* corresponde ao *StateName* e o *value* corresponde ao tuplo com as médias da latitude e da longitude, criando o par

(StateName,((MinLat+MaxLat)/2,(MinLon+MaxLon)/2))). Uma vez que existem coordenadas de monitores repetidas, recorreu-se à transformação *distinct* para removê-las. A partir dos dois RDD's utilizou-se a transformação *join* de forma a associar as coordenadas de cada monitor às coordenadas do estado correspondente (tendo este *narrow dependencies*, uma vez que o RDD resultante terá apenas uma dependência a partir do RDD anterior, não necessitando de um esforço computacional tão grande como uma *wide dependency*) [5]. Novamente, recorreu-se a um *map* para calcular a distância euclidiana entre cada monitor e o centro do estado a partir do tuplo formado anteriormente. De seguida, agrupou-se as distâncias calculadas, através da mesma *key*, numa lista e, por fim, fez-se a média a partir da soma da lista dividindo pelo seu tamanho.

Q5) Começaremos por ler o ficheiro auxiliar com as coordenadas dos estados para guardarmos num dicionário "NomeEstado": (latMédia, longMédia). Isto foi feito usando um *map* em que, para cada estado, os valores da latitude/longitude médias são calculados como no exercício anterior. Depois de guardados estes valores num dicionário, poderemos carregar o ficheiro das observações, fazer as mesmas filtragens iniciais, e comparar as coordenadas dos monitores e dos estados para determinar o quadrante a que pertencem. Esta comparação foi feita à parte, numa *User-Defined Function*, chamada a partir de um *map* por serem bastantes comparações. Depois disto feito, removemos as repetições agrupando os valores com *keys* iguais (*reduceByKey*) e ordenamos pelo número de monitores.

5. Spark Data Frame

Q1) Depois de importado o ficheiro das observações e feitas as filtragens iniciais, fazemos um map para definir a estrutura das linhas da *Dataframe*. A primeira coluna corresponderá ao código do estado e a última às coordenadas do monitor (concatenadas numa só coluna). Depois de definida a estrutura da DF, selecionaremos estas duas, agruparemos por estado e contaremos o número de ocorrências distintas das coordendadas (*agg(countDistinct("coords"))*) [2]. Finalmente, ordenamos de forma descendente pelo número de monitores presentes em cada estado.

Q2) Depois do procedimento inicial, para a definição desta DF, usámos o código do estado, o *county* e o valor médio da poluição diária registada. De seguida, após agrupar as linhas pelo estado e pelo *county*, calculamos a média dos valores da poluição usando a função *.avg(poluicaoMedia)*. No final, damos um novo *alias* à coluna da média e ordenamos pela quantidade de poluição.

Q3) Depois de excluída a primeira linha e de separadas as colunas do ficheiro, define-se as linhas da *Dataframe* utilizando o ano, o estado e a qualidade do ar como float, e de seguida cria-se a mesma. Posteriormente, agrupa-se (utilizando a função *groupBy*) as linhas pela mesma *key* (ano,estado) e calcula-se a média (como foi descrito na *query* anterior) dos respetivos valores. Por fim, basta ordenar as linhas por ano e qualidade do ar média (utilizando *orderBy* e imprimir (*show*) a DF resultante.

TABLE 1. QUERY 1

State	Count
06	70
48	133

TABLE 2. QUERY 2

State	County	AvgPolution
47	167	2556.0
36	059	19.0

Q4) Para a criação da DF relativa ao ficheiro auxiliar dos estados, as linhas foram diretamente escritas com os valores médios das latitudes e longitudes. Para a DF constituída pelas observações do monitor, cada linha terá o nome do estado, latitude e longitude. A partir da função *join* combinou-se as duas tabelas a partir da mesma coluna "state". Seguidamente, selecionou-se a coluna correspondente à latitude e à longitude de cada monitor e de cada estado, para calcular a distância euclidiana entre os dois pontos, formando uma nova coluna com as distâncias calculadas. Por fim, agregou-se cada linha para calcular a média da distância, agrupando pelo nome de cada estado (com *GroupBy*).

Q5) Para a criação das duas DF fez-se o mesmo procedimento que a questão anterior. Para fazermos a comparação entre as duas, precisamos de as juntar usando como elo de ligação o nome do estado (presente nas duas) usando um *Inner Join*. Depois de termos as duas DF juntas, fazemos a comparação das coordendadas para determinar o quadrante respetivo a cada um dos monitores. Para isto podemos utilizar a função *withColumn(Quadrant, when(...)*, que cria uma coluna nova com esse nome e dentro desta, nos vários *whens*, compara as coordenadas e escreve o quadrante respetivo. No final, fazemos *drop* das colunas que não têm interesse e ordenamos pelo número de monitores, por quadrante de cada estado.

6. Spark SQL

Q1) Para esta tecnologia, a resolução foi muito semelhante ao SparkDF. As linhas foram definidas da mesma maneira e para a *query* utilizámos a função *count(distinct coordenadas)* de modo a calcularmos o número de coordendadas diferentes por estado (correspondente ao número de monitores). No final, agrupámos por estado e ordenámos à mesma pelo número de monitores.

Q2) Nesta pergunta, a resolução é análoga à da pergunta anterior mas, ao invés de usarmos a função *count* para contar o número de ocorrências, tivémos de utilizar o *avg* para a média de todos os valores médios de poluição por *county*. No final, ordenámos pela quantidade de poluição.

Q3) Para responder a esta *query* em SparkSQL, segue-se os mesmos passos que em DataFrame, até à criação da tabela. De seguida, define-se como *query* a seleção das colunas *year*,*state* e *avg(airq)*, a qual se obtém depois de agregadas as colunas *year* e *state*, à qual daremos o nome de "AveragePolution". Para além disto, ordena-se as linhas por *year* e *AveragePolution*.

TABLE 3. QUERY 3

Year	State	TotalAirQ
1990	Oklahoma	0.0
1990	Virgin Islands	0.0

TABLE 4. QUERY 4

State	AvgDistance
Virginia	715.433
Alaska	603.699

Q4) Para esta questão, recorreu-se às duas tabelas formadas no Spark SQL. O "*SELECT*" formado devolve o nome de cada estado e a distância média entre os monitores e o centro do estado. Assim, foi realizado um "*INNER JOIN*" entre as tabelas "table_epa_daily" e "table_usa_state", gerando uma nova relação cujos atributos são formados pelos atributos de ambas as tabelas cujos nomes dos estados (state) sejam iguais, com o objetivo de calcular a distância euclidiana entre os pontos do monitor relativamente aos pontos do estado. No final, agrupou-se os tuplos da tabela obtida (através do *GROUP BY*) em ordem ao nome de cada estado.

Q5) Nesta última pergunta, o tratamento dos dados do ficheiro auxiliar foi igual ao do *Spark Dataframe*. Para o ficheiro principal, a tabela definida para este ficou com o mesmo esquema. Para a *query*, a ideia também foi a mesma, no entanto, usando uma função diferente no lugar da *withColumn*, sendo esta a *CASE WHEN*. A ideia é a mesma, ou seja, funciona como um *if else*. Depois das comparações feitas, adiciona a *string* correspondente ao quadrante a uma coluna nova criada ("Quadrante"). Para a junção das duas tabelas, utilizámos à mesma um *Inner Join* pelo nome do estado e no final, ordenamos pelo número de monitores por quadrante por estado.

7. Hive

Primeiramente, dado que estas tabelas ficam em memória, começaremos por fazer *drop* das tabelas que criaremos. De seguida, definimos a estrutura das tabelas, criando os campos necessários com os tipos respetivos para cada um. Depois de definidas, carregaremos ambos os ficheiros (auxiliar e principal) para estas para podermos fazer as *queries*.

Q1), Q2), Q4) Para esta pergunta, a *query* foi construída exatamente da mesma forma que para o *Spark SQL*.

Q3) Esta foi construída exatamente da mesma forma que em *Spark SQL*, com a exceção de que agora, como a coluna "Date Local" é constituída por variáveis da forma "Date", para nos referirmos ao "year", basta fazer "YEAR(dateLocal)".

TABLE 5. QUERY 5

NrMonitors	StateName	Quadrant
84	California	NE
74	Michigan	SW

TABLE 6. TEMPOS DE EXECUÇÃO EM MILISEGUNDOS

	MR	Spark	SparkDF	SparkSQL	Hive
Q1	4239	1820	2360	2460	5461
Q2	4240	1890	2150	2110	5461
Q3	4313	1970	2230	2330	5843
Q4	6070	1880	2390	2220	6189
Q5	7403	1920	2690	2410	7480

Q5) Nesta *query*, tivémos alguns problemas ao usarmos a mesma que em *Spark SQL*. O primeiro prende-se com o facto de o *hive* não conseguir criar uma nova coluna dinamicamente aquando o uso do *CASE WHEN*. Assim, tivemos de adicionar *apriori* as colunas extra que seriam necessárias à tabela original. O segundo estava relacionado com o *COUNT (DISTINCT ...)* que, para as linhas distintas, não fazia o agrupamento entre estas para fazer a contagem o que resultava num valor de 1 para todas as linhas. Para o resolver, tornámos a *query* original numa *sub-query* e na principal voltámos a fazer o *count* dos monitores dando o resultado desejado. Fora estas duas mudanças, a construção da *query* foi igual a *Spark SQL*.

8. Tempos de Execução

Ao analisar a tabela 6 podemos verificar, como era esperado, que o grau de computação necessário para a última *query* é bastante mais elevado que para as quatro restantes, visto que esta é bastante mais complexa que as anteriores. No que toca às diferenças entre as várias tecnologias, é notório que o *Spark* é o que apresenta uma perfomance mais satisfatória, e não parece ser afetado pelo acréscimo da complexidade, isto porque toda a sua excecução é feita *'in-memory'* [3]. Por outro lado, as tecnologias que mais tempo levam a executar as *queries* são o *Map-Reduce* e o *Hive*. No entanto, com o aumento da complexidade do problema, o tempo de execução do Hive tende a convergir para o do *Map-Reduce*, possivelmente ultrapassando-o caso tivéssemos um problema computacionalmente mais exigente.

9. Conclusão

Relativamente às várias tecnologias, as que considerámos serem mais fáceis e práticas de utilizar foram o Spark, SparkDF e SQL. Em relação ao MapReduce, este revelou-se bastante mais extenso e com uma programação mais complexa, o que reduziu drasticamente a facilidade da leitura do código. Relativamente ao SparkDF e SparkSQL a resolução foi semelhante entre os dois mudando apenas a sintaxe. Finalmente, para o Hive, por este usar uma versão mais antiga do SQL, tivemos o trabalho adicional de adicionar as colunas extra *apriori*, por este não as conseguir criar dinamicamente, ao contrário do SQL atual, o que se pode tornar pouco eficiente para projetos mais elaborados.

10. Agradecimentos

João Filipe Traitolas Naves, Miguel Simões no Piazza na leitura de dois ficheiros em simultâneo!

11. Contribuições

Em relação à contribuição de cada membro do grupo temos a salientar a dedicação do João, que se destacou na parte do código das diferentes tecnologias. Em relação ao relatório todos os membros tiveram igual contribuição.

TABLE 7. CONTRIBUIÇÕES (EM PERCENTAGEM)

	Contribuição
João Funenga	38%
Beatriz Breia	31%
Catarina Afonso	31%

References

[1] Stackoverflow. Acedido a 13/12/2021, em https://stackoverflow.com/questions/25023018/convert-a-string-tuple-to-a-tuple?lq=1

[2] Stackoverflow. Acedido a 13/12/2021, em https://stackoverflow.com/questions/46421677/how-to-count-unique-id-after-groupby-in-pyspark

[3] Diep, N. (2020, 16 de janeiro). *Big Data Analytics: Apache Spark vs. Apache Hadoop*. Towards Data Science. Acedido a 13/12/2021, em https://towardsdatascience.com/big-data-analytics-apache-spark-vs-apache-hadoop-7cb77a7a9424

[4] Brooklin College. Acedido a 13/12/2021, em http://academic.brooklyn.cuny.edu/geology/grocha/mapsdistancelatlong/directionlatlong.htm

[5] Cao M. (2018, 27 de dezembro). *Wide vs Narrow Dependencies.* Hackx's Blog. Acedido a 17/12/2021, em https://untitled-life.github.io/blog/2018/12/27/wide-vs-narrow-dependencies/