SPBD - Projecto

Borges R., Vítor; Cardoso, Alexandre; Marques, Filipe

Resumo — Neste trabalho utilizaram-se 5 tecnologias de processamento de Big Data, por forma a analisar dados sobre poluentes tóxicos do ar, recolhidos pela EPA, e assim responder a diferentes questões sobre o tema. As tecnologias utilizadas foram: Hadoop Map-Reduce, Apache Spark, SparkDF, SparkSQL e Hive. Desenvolveram-se algoritmos para responder às diferentes questões e de seguida implementaram-se com sucesso nas tecnologias referidas. Verificou-se que, apesar de todas possuírem uma capacidade de processsamento assinalável, há tecnologias mais intuitivas, fáceis e rápidas.

Palavras-chave — Processamento de Big Data, Hadoop Map-Reduce, Apache Spark, SparkDF, SparkSQL e Hive.

INTRODUÇÃO

Neste projecto é efetuda e discutida a primeira categoria do processo de análise de dados - a *Descriptive Analytics (DA)*[1] - sobre um conjunto de dados relacionados com a poluição nos Estados Unidos da América e a sua influência na saúde pública.
A *DA* foi efetuada com recurso a diferentes tecnologias de processamento de *Big Data* aprendidas no âmbito da disciplina SPBD: **1) Hadoop MapReduce**, onde as intruções são passadas em funções Map e Reduce de forma sucessiva de modo a organizar o output final nos pares k*ey,values* desejáveis. Isto é possivel porque, entre as operações MapReduce, são implementadas funções intermédias (*combiner, partitioner*) e (s*huffle, sorting*) que organizam e ordenam os dados consoante a chave nos *nodes* e provenientes dos mesmos, respectivamente [2]; **2) Apache Spark**, que assenta a sua arquitectura na abstração *Resilient Disributed Datasets* (*RDD*s), tolerante a falhas e de processamento *lazy,* onde são criadas sequências de transformações, que apenas são executadas quando chamada uma instrução de acção [3]; **3) Os módulos SparkDF** e **4) sparkSQL** que integram na mesma interface o sistema relacional e procedimental que facilita a implementação e execução de operações mais complexas [4] e **5) Hive**, uma solução de *DATA Where House*, que tira vantagem da estrutura Hadoop e permite ao usário criar buscas em *Hive Query Language* (linguagem declarativa do tipo SQL) que são implementadas em trabalhos MapReduce [5].
Assim, com recurso a estas tecnologias, foram respondidas cinco questões, que requereram o processamento de um conjunto de dados, recolhidos pela *Environmental Protection Agency* (EPA), sobre alguns poluentes presentes na atmosfera, que se crê estarem na origem de diferentes comorbidades.
O conjunto de dados a analisar está em formato .csv e organizado da seguinte forma: a primeira linha contém os cabeçalhos das 29 colunas divididas por ‘,’. As restantes entradas seguem o mesmo formato e cada linha contém informação sobre o rastreio atmosférico efetuado pelos monitores num determinado dia.
Por forma a facilitar a leitura e compreensão deste projecto, este documento está dividido em cinco capítulos que correspondem a cada uma das questões. Em todos eles é explicado de forma sucinta a lógica de resolução da questão. De seguida é discutido de que forma foram aplicadas as diferentes tecnologias e no final é mostrada uma amostra do *output*.

- *Vitor Borges Rodrigues, aluno do Mestrado em Análise e Engenharia de BIG Data -FCT-UNL, 61831. E-mail: vm.rodrigues@campus.fct.unl.pt*
- *Filipe Marques, aluno do Mestrado em Engenharia Eletrotécnica e de Computadores-FCT-UNL, 52755, E-mail: fmdb.marques@campus.fct.unl.pt*
- *Alexandre Cardoso, aluno do Mestrado em Engenharia Eletrotécnica e de Computadores- FCT-UNL, 53202, E-mail: ams.cardoso@campus.fct.unl.pt*

1 QUESTÃO 1:

1.1 Lógica de Resolução

Para responder a esta questão, consideraram-se apenas as colunas ‘*state_name*’, ‘*latitude*’ e ‘*longitude*’, pois como a posição geográfica do monitor é unica, a contagem das entradas distintas na base de dados permitirá obter o total de monitores existentes para cada estado.

1.2 Map-Reduce

Para resolver esta questão foi necessário efetuar duas operações *map-reduce*. Na primeira, iterou-se sobre as linhas e imprimiu-se a concatenação ‘*state_name + posição monitor+\n+1’*, com este formato e devido às operações intermédias de ordenação, foi possível durante o *reduce* considerar apenas as entradas distintas e somar os valores referentes a cada estado. Na operação *map-reduce* final, organizou-se o *output* (estado, número monitores) de forma decrescente considerando a quantidade de monitores.

1.3 Spark, SparkDF, SparkSQL

Em spark foram aplicadas quatro funções de tranformação e duas de acção: *.filter(),* para retirar possíveis entradas em branco e o cabeçalho (foi utilizada a função acção *.first()* para obter primeira linha); *map()* considerando o “*state_name*” como *key* e a concatenação da longitude e latitude como *value* num *set*, para depois aplicar a transformação *reduceByKey,* que permite agrupar os valores de cada chave, e neste caso, como os valores (posição monitor) estão num *set*, é possivel considerar apenas os distintos através da operação “|”. Estando os dados organizados em *key {Values},* foi efetuada uma nova transformação *.map* para verificar o tamanho do set e assim obter a quantidade

de monitores por estado. De seguida efetuou-se uma transformação *.sortBy()* considerando a quantidade de monitores. Por fim, iterou-se sobre o *array* obtido através da acção *.collect()* por forma a imprimir cada linha.
A resolução deste problema foi simplificada através das *sparkDF/SQL API*. Após preparados os dados através de funções de tranformação, o *rdd* foi convertido em *DataFrame* (DF) e de seguida foram aplicadas sobre ele as funções necessárias para obter o *output* pretendido. Em SparkDF agruparam-se os estados pelo nome através da função *groupBy()* e de seguida, com recurso à função de agregação *(.agg(countDistinct()))*, contaram-se todas as latitudes/longitudes distintas por estado. Por fim, utilizou-se a função *orderBy()* para obter um *output* ordenado por quantidade de monitores. Em sparkSQL após criada uma *temporary view* do DF, aplicou-se uma *query* que seleciona e conta de forma agrupada todas as linhas onde o conteúdo (state_name, posição) é distinto.
Sem recorrer às funções de transformação de rdd e subsequente conversão em DF, resolveu-se este exercício utilizando a função *spark.read.option().csv()* e verificou-se existirem ganhos assinaláveis na velocidade de processamento das tecnologias sparkDF/SQL.

1.4 Hive

Em HIVE criou-se uma tabela epa_hap_daily_summary e de seguida fez-se uma *view* "epa_hap", onde se retirou o cabeçalho e os valores *NULL*, usando a propriedade "*skip.header.line.count*" = "1" e *where is not Null*. A seguir, utilizou-se uma *query* semelhante à utilizada em sparkSQL, onde foram selecionadas e contadas de forma agrupada todas as entradas distintas, considerando as colunas *"state_name, latitude/longitude"*, o que possibilitou a identificação do número de monitores distintos por estado, apresentando uma tabela com o conteúdo organizado de forma decrescente, considerando a coluna criada ("*monitors_number*").

1.5 Output (amostra)

Estado	*Quantidade*
California	170
Texas	133
Minnesota	94
Michigan	92
Ohio	91

Tabela 1. Os 5 estados com maior quantidade de monitores.

2 Questão 2:

2.1 Lógica de Resolução

Para responder a esta questão, consideraram-se as colunas *'county_name'* e *'arithmetic_mean'* que representam o valor médio de poluentes diário registado por cada monitor. Assim, fazendo a média dos registos por *county*, é possivel ordená-los por qualidade de ar.

2.2 Map-Reduce

Para resolver esta questão, efetuaram-se duas operações map-reduce: No primeiro map, imprimiu-se o nome do *county* e a média de poluentes observada em cada entrada. No reducer realizou-se a soma dos valores e a contagem de quantos são lidos, por forma a imprimir a média do *county*, que é efetuada quando surge uma key (nome county) diferente. Na última operação é ordenado o *output* por valor crescente (média de poluentes no ar). Para tal, no map é alterado a relação *key-value* para as funções intermédias entregarem à função reduce, um *input* correctamente ordenado, sendo que no reduce apenas se reorganiza a relação *key-value* de acordo com o original (*county*, média).

2.3 Spark, SparkDF, SparkSQL

Recorrendo à tecnologia pySpark, após o processamento inicial (retirar cabeçalho, linhas vazias e *null*), foram aplicadas quatro funções de transformação e uma de acção: *.map(),* para considerar a *tuple* (county_name, aritmethic_mean, 1) o que torna possível, através da função *.reduceByKey(),* agrupar os *countries,* considerando a soma das médias de cada entrada e a sua quantidade. De seguida, através de uma nova função *.map(),* efetuou-se a média para cada estado. Por fim, organizou-se o *output* pela melhor qualidade do ar aplicando a função *sortBy().*
Através da tecnologia sparkDF, após a transformação rdd em dataframe, resolveu-se a questão da seguinte forma: utilizando *groupBy(),* agruparam-se os *counties* de todas as linhas do DF, e através das funções *.agg(avg())* calculou-se a média de quantidade de poluentes referente a cada estado. Por fim, ordenou-se o DF de forma crescente recorrendo ao orderBy(). Em sparkSQL, após criada a *temporary view* selecionaram-se os *counties* e criou-se uma nova coluna que recebe os dados do agrupamento por *county* à posteriori.

2.4 Hive

Utilizando a *view* previamente criada como mencionado em 1.4, foi implementada uma *query,* semelhante à aplicada em SparkSQL, onde selecionamos a coluna *county_name* e definimos a variável "media_poluicao", que recebe a média dos valores da coluna arithmentic_mean, considerando os agrupamentos por county_name.

2.5 Output (amostra)

Condado	*Valor Médio Poluiçao* ($x10^{-6}$)
Sweet Grass	0
Martin	0
Wrangler Petersburg	45.4
Northwes Arctic	63.3
Aleutians	109.6

Tabela 2. Os 5 condados com melhor qualidade de ar.

3 Questão 3:

3.1 Lógica de Resolução

Para responder a esta questão, consideraram-se as colunas referentes ao nome do estado, à média de poluentes e também a data de observação, pois pretende-se efetuar a ordenação de acordo com o ano. Assim, ordenaram-se os respectivos estados pela média da quantidade de poluentes anual.

3.2 Map-Reduce

Na resolução desta questão, foram efetuadas duas operações map-reduce: Na primeira operação, com o intuito de organizar o *output* por ano e estado, imprimiu-se a concatenação "ano+county" e separadamente o valor médio. De seguida, com as linhas ordenadas, é calculada e impressa a média para cada concatenação ano_county. Como foi necessário ordenar os estados por valor (média poluente) em cada ano, foi efetuada uma nova operação, onde no map se imprimiu a concatenação "ano+valor \n state_name", que após ordenadas pelas funções intermédias, foram de novo colocadas no formato original no reduce (ano, estado, média).

3.3 Spark, SparkDF, SparkSQL

Utilizando o pyspark, após o processamento inicial, foi aplicada uma função de transformação *.map()* que considera a concatenação "ano+county" como *key* e a média e o valor 1 como *values*. De seguida, recorreu-se à função *reduceByKey()* por forma a agrupar as diferentes *keys* e calcular a soma das suas médias e a quantidade de entradas, para ser possível calcular a média anual através de uma função .map(). Nessa função map() desfez-se a concatenaçaõ e considerou-se apenas o ano como *key*, colocando o *tuple* (nome do county e a respectiva média) num *set* - (k,{(v1,v2)}). Repetiu-se a função *reduceByKey* por forma a agrupar por ano os *sets*. Recorrendo ao *orderBy()* ordenou-se a listagem por ano.
Por fim, fez-se uma dupla iteração sobre o *array* retornado pela função *.collect()* e sobre o segundo valor da primeira iteração (*set*) por forma a imprimir o *output* no formato ano, estado, média anual (ordenado por ano e média).
Recorrendo ao pySpark DF para resolver esta questão após criado o dataframe foram aplicadas as funções seguintes: *goupBy(substring(data), estado)* por forma agrupar as linhas do dataset por ano e por estado, para logo de seguida agregar a médias dos diferentes agrupamentos através das funções *agg(avg())*. Por fim ordenou-se o novo DF recorrendo à função *orderBy()*. Em SQL o processo lógico foi semelhante, selecionaram-se as colunas (substring (data,…), county e foi criada uma nova coluna que receberá as média do agrupamento data,county, que será ordenada de seguida por ano,média.

3.4 Hive

Baseado na view "epa_hap" foram criadas duas queries. Na sub-query são selecionadas três colunas, data, state_name e arithmetic_mean, sendo que a data irá ser o resultado da operação substring(date_local,0,4) na qual é retirado apenas o ano, tendo o date_local nos primeiros 4 caracteres o valor correspondente ao ano. Na main query são selecionadas as colunas da sub-query referentes à data(t.data), nome do state(t.state_name), e é calculada a média das entradas da coluna t.arithmetic_mean, declarando o resultado como média, sendo feito um agrupamento deste por t.data, t.state_name, e o resultado final ordenado duplamente por ordem crescente dos valores t.data,t.media.

3.5 Output

Optou-se por um gráfico, pois representa com maior clareza o output

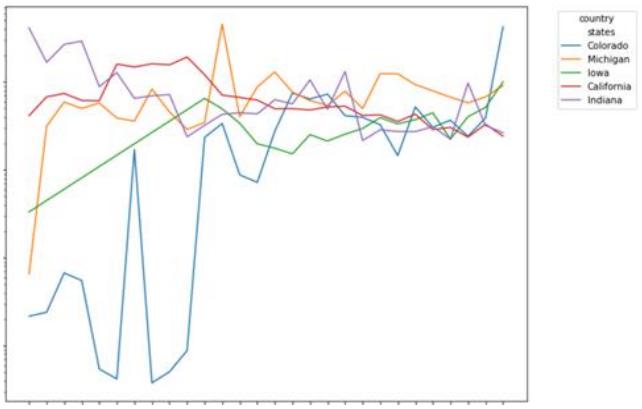

Figura 1. Evolução (1990-2017) dos 5 estados com pior qualidade de ar em 2017. (efectuado em sparkDF com Pandas)

4 QUESTÃO 4:

4.1 Lógica de Resolução

Considerando os limites de cada estado, foi calculado o centro do mesmo, somando as suas longitudes e latitudes e dividindo por dois. Assim, com as coordenadas do centro de cada estado, realizou-se para cada monitor o cálculo da sua distância ao centro e de seguida fez-se a média para cada estado.

4.2 Map-Reduce

Para a resolução desta questão, foram aplicadas duas operações *map-reduce*. No map criou-se um dicionário com os limites e centro geográficos de cada estado, e recorrendo à técnica utilizada na questão 1 imprimiu-se para cada linha a posição do monitor e o respectivo centro estatal. Assim, durante a fase reduce apenas as entradas distintas (monitores) eram consideradas e foi possível calcular a suas distâncias ao centro bem como a respectiva média de distâncias estatal. A segunda operação map-reduce garantiu a ordenação do output através da troca na relação *key, value*.

4.3 Spark, SparkDF, SparkSQL

Em pyspark, após carregar ambos os csv(*main data* e state *data*) e processar o seu formato, foram aplicadas as técnicas de resolução da questão 1 para considerar apenas as posições distintas de cada monitor e foi criado um outro rdd com os centros estatais utilizando a função .map. De seguida, recorrendo à operação *rdd.join(rdd2)*, combinaram-se ambos rdd (estado, monitores, estado centro) pela *key*, o que possibilitou o calculo distância ao centro de cada monitor através do *.map()*. Por último, foi aplicada uma função *reduceByKey()* que permitiu agrupar os estados e calcular as suas distâncias médias, aplicando de seguida a *orderBy*.
No pySparkDF foram criados dois dataframes contendo a informação dos diferentes .csv através *read.option().csv()* e de seguida juntaram-se através .join() pelo valor da chave. Recorrendo à função *withColumn()*, foi possível criar uma nova coluna com as distâncias ao centro de cada monitor. Após esta operação agruparam-se os estados e calculou-se

a média das distancias através das funções *groupBy()* e *agg(avg()).* Por fim, utilizou-se o *orderBy().*
Em sparkSQL foram criadas duas temporary views considerando os DF. De seguida, foram aplicados *queries* sucessivos com respetivas tranformações em *temporary views* até obter a tabela conjunta no formato (estado, posição monitor, center_lat, center_log), por forma a aplicar o *query* que seleciona os estados, as posições dos monitores e calcula sua distância ao centro através do teorema de Pitágoras. Distâncias que são agrupadas por estado para calcular a média de dispersão dos seus monitores.

4.4 Hive

Em Hive utilizou-se a view principal e criou-se mais uma que contêm os dados provenientes do .csv dos limites estatais, que foram juntas numa nova view posteriormente. Criou-se uma nova *view* a partir da última *joint view,* selecionando os estados e as respectivas média de dispersão de monitores, e por fim aplicou-se a querie final para ser processada em map-reduce.

4.5 Output (amostra)

Estado	*Distância ao centro (km)*
Virginia	715.4
Alaska	603.7
Texas	512.2
Vermont	504.1
Illinois	440.9

Tabela 3. Estados com monitores mais longe do centro

5 Questão 5:

5.1 Lógica de Resolução

Considerando o centro de cada estado, foi verificada através de operações de comparação a posição de cada monitor em relação ao mesmo.

5.2 Map-Reduce

O map-reduce utilizado nesta questão é semelhante ao utilizado na questão 4, diferindo apenas no reduce realizado, pois em vez do cálculo da distância, efetuou-se a comparação entre a posição dos monitores/centro e efetuou-se a contagem de monitores por quadrante.

5.3 Spark, SparkDF, SparkSQL

Pare resolver esta questão através da tecnologia pyspark, foram adotadas as mesmas técnicas que na resolução da questão 4 até à operação .join(). De seguida, utilizando a função .map() e as condições built-in do python criou-se um rdd contendo a informação da posição de cada monitor. Por forma a calcular o somatório de cada quadrante, utilizou-se a função *reduceByKey().* No final imprimiu-se o valor no formato (key, (v1,v2,v3,v4)) em que os values correspondem aos quadrantes NE,SE,SW,NW respetivamente.
Recorrendo ao sparkDF foram também adoptadas as tecnicas de resolução do exercício anterior até à junção dos diferentes dataframes e de seguida criaram-se quatro colunas através de funções *.withcolumn()* e com recorrendo à comparação de valores definia-se o quadrante de determinado monitor. Por último, utilizando as funções *groupBy* e *agg(sum())* obteve-se o output desejado.
A resolução em sparkSQL foi semelhante ao exercicio anterior, apenas diferindo nos 2 ultimos *queries*; no penultimo foi necessário definir o quadrante de cada monitor através do comando SQL "*CASE WHEN*" e no último agruparam-se os estados e foram somados os totais de cada quadrante.

5.4 Hive

Em Hive, foram seguidas as mesmas técnicas que a alínea anterior até obter a joint view, e de seguida criou-se uma nova view contendo colunas referentes aos diferentes quadrantes. No final, foi aplicada uma query, que agrupa os estados e fornece o somatório de monitores por quadrante.

5.5 Output (amostra)

Estado	*NE*	*SE*	*SW*	*NW*
Alabama	5	5	7	14
California	2	68	16	84
Maryland	16	0	0	1
Texas	34	72	3	24
Wisconsin	2	21	1	2

Tabela 4. Distribuição de monitores por quadrante

6. Conclusões

Através do trabalho desenvolvido, foi possível verificar que todas as tecnologias são capazes de resolver de forma eficiente problemas relacionados com *descriptive analytics*, contudo há diferenças na sua implementação, intuição e performance.
Considera-se que as tecnologias sparkDF, sparkSQL e Hive são mais intuítivas e fáceis de implementar, pois a sua sintaxe e lógica de implementação de alto nível favorecem o entendimento humano.
No que diz respeito à performance, foi medido o tempo de processamento real das diferentes tecnologias, utilizando um I9 – 11900H 11th gen 2.5 GHz, 32 Gb RAM DDR4 3200 MHz, 1Tb SSD, e verificou-se que as tecnologias spark têm um desempenho superior em todas as questões (tabela 5), o que se deve à sua abstração RDD.
Em suma, o trabalho efetuado permitiu desenvolver perícias e testar diferentes tecnologias na resolução de problemas "reais" relacionados com a *Big Data,* contribuindo de forma indubitável para a nossa formação académica.

tecnologia	Q1	Q2	Q3	Q4	Q5
MapReduce	5,594	4,436	5,477	3,339	5,409
Spark	0,941	1,035	1,144	1,150	0,909
SparkDF	0,816	1,261	1,318	0,844	1,025
SparkSQL	0,993	1,309	1,310	0,201	0,201
Hive	7,809	7,126	7,578	9,599	9,751

Tabela 5. Tempos de processamento para as diferentes questões em cada tecnologia

7. Contribuições

O contributo dos elementos do grupo foi igualmente repartido considerando a complexidade de implementação das diferentes tecnologias. Assim, considera-se que a taxa de avaliação deve ser repartida de forma igual por todos os elementos.

8. Agradecimentos

Um especial agradecimento a todos os colegas que partilharam dúvidas e sugestões nas plataformas partilhadas e ao Professor João Santos Lourenço pela sua disponibilidade, compreensão e mentoria.

References

[1] Thomas Erl, Wajid Khattak, and Paul Buhler ; Big Data Fundamentals Concepts, Drivers & Techniques. Prentice Hall (2015)

[2] Jeffrey Dean and Sanjay Ghemawat, "*MapReduce: Simplified Data Processing on Large Clusters.*" OSDI 2004

[3] Matei Zaharia et Al., "Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster" University of California, Berkley

[4] Michael Armbrust et Al., "Spark SQL: Relational Data Processing in Spark" SIGMOD15, May 31–June 4, 2015, Melbourne, Victoria, Australia

[5] Ashish Thusoo et Al. *"Hive A Warehousing Solution Over a MapReduce"* VLDB '09, August 24-28, 2009, Lyon, France