Analisi del dataset delle vaccinazioni anti Covid-19 con Hadoop/Spark

Relazione Primo Progetto

Petrolo Melissa
Corso di Laurea Magistrale in Ingegneria Informatica
Università degli Studi di Roma Tor Vergata
Matricola: 0286160
emelis.ptr@gmail.com

I. Introduzione

In questa relazione viene mostrata l'implementazione delle query e l'utilizzo dell'architettura Spark.

L'obiettivo è quello di analizzare i dataset delle vaccinazioni anti covid-19, forniti dal Commissario straordinario per l'emergenza Covid-19, Presidenza del Consiglio dei Ministri [1] e di utilizzare i framework Hadoop/Spark per il processamento dei dati.

Per poter ottenere i risultati necessari per l'analisi, sono state effettuate delle query:

- Query uno: calcolare il numero medio di somministrazioni effettuate mensilmente in una determinata area in base al numero di centri di vaccinazioni.
- Query due: determinare le prime cinque aree per le quali è previsto il maggior numero di vaccinazioni per il primo giorno del mese successivo.

II. Architettura e framework

Per poter eseguire e realizzare il progetto si è fatto uso di Docker; una piattaforma software che permette di creare, testare e distribuire applicazioni con la massima rapidità.
È stato creato un file docker-compose per la creazione di container dei servizi utilizzati:

- Apache Spark: framework per l'elaborazione del processamento dei dati.
- Apache Hadoop: framework per l'archiviazione dei dati in un ambiente distribuito.

III. Implementazione query

All'inizio delle operazioni entrambe le query sono state filtrate per utilizzare solamente le colonne necessarie per lo sviluppo di queste.

A. Query uno

Per la prima query sono state prese in considerazioni due dataset: punti-somministazione-tipologia.csv e somministrazioni-vaccini-summary-latest.csv, per poter calcolare il numero medio di vaccinazioni per area in base al numero di centri di vaccinazioni.
Di seguito, viene mostrato il diagramma delle operazioni che sono state effettuate per lo svolgimento della prima query.

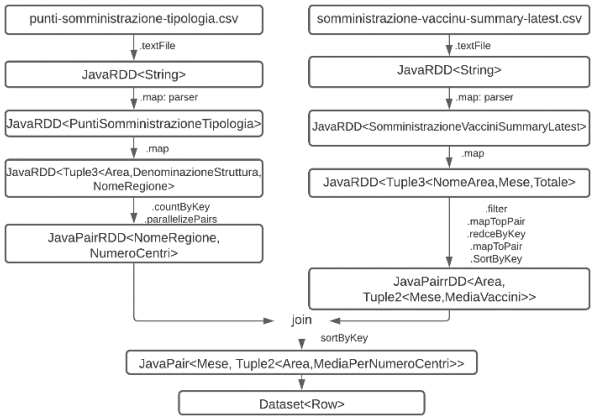

Figura 1- Diagramma di flusso - Query Uno

Per il primo dataset sono state utilizzate le seguenti colonne: *area*, *denominazione struttura* e *nome regione*, per poter calcolare il numero di centri di vaccinazioni per ogni area. La query è stata sviluppata nel seguente modo:

- *JavaRDD<String>:* RDD è stato creato attraverso l'acquisizione di un file di testo, caricato precedentemente in HDFS;
- *JavaRDD<Tuple3<Area,DenominazioneStruttuta,NomeRegione>:* come detto in precedenza, sono state selezionate solo le colonne necessarie;
- *JavaPairRDD<NomeRegione,NumeroCentri> :* si è creata una coppia RDD per associare ad

ogni regione il numero dei centri di vaccinazione attraverso la funzione countByKey e successivamente è stato necessario effettuare parallelizePairs per trasformare la mappa dei valori in una coppia RDD;

Per il secondo dataset, invece, sono state utilizzate le seguenti colonne: *data di somministrazione, area* e *totale del numero di vaccinazioni*, per poter calcolare il numero medio mensili di vaccinazioni. La query è stata sviluppata nel seguente modo:

- *JavaRDD<String>:* RDD è stato creato attraverso l'acquisizione di un file di testo, caricato precedentemente in HDFS;
- *JavaRDD<Tuple3<Area,Mese,Totale>>:* anche in questo caso, attraverso il parsing sono state selezionate solamente le colonne necessarie;
- *JavaPairRDD<Area,Tuple2<Mese,MediaVaccini>>:* inizialmente i dati sono stati filtrati per eliminare le date precedenti al 1° gennaio 2021. In seguito, si è mappata una tupla2<Area, Mese> per poter calcolare attraverso reducedByKey il numero totale di vaccinazioni per quel determinato mese. Infine, si è creato una coppia Area e una tupla con mese e media vaccini.

Dopo aver filtrato ed eseguito le operazioni per ottenere i dati necessari, è stata eseguita una join avente come chiave in comune l'area di riferimento; in questo modo è stato possibile calcolare il numero medio di vaccinazioni mensili per il numero di centri di vaccinazioni di quella determinata area.

B. Query due

Per la seconda query è stato preso in considerazione solamente un dataset: somministrazione-vaccini-lastest.csv.
Di seguito, viene mostrato il diagramma delle operazioni che sono state effettuate per lo svolgimento della seconda query.

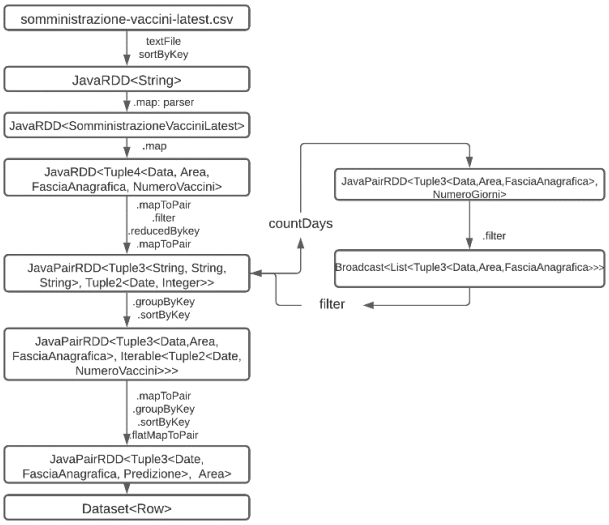

Figura 2- Diagramma di Flusso - Query Due

In questa determinata query vengono considerati il numero di vaccinazioni per ogni fascia anagrafica ma solo per il sesso femminile.

Per la seconda query sono state considerate le seguenti colonne: *data di somministrazione, area, fascia anagrafica e numero di vaccini effettuati*. La query è stata sviluppata nel seguente modo:

- *JavaRDD<String>:* RDD è stato creato attraverso l'acquisizione di un file di testo, caricato precedentemente in HDFS;
- *JavaRDD<Tuple4<Data,Area,FasciaAnagrafica,NumeroVaccini>>:* sono state prese in considerazioni i quattro valori necessari per il processamento della query;
- *JavaPairRDD<Tuple3<Mese,Area,FasciaAnagrafica>,Tuple2<Data,NumeroVaccini>>:* inizialmente è stata creata una coppia RDD con una tupla contenente il mese di riferimento, area e fascia anagrafica e una tupla con la data e il numero di vaccinazioni; a cui è stato applicato il filtro per eliminare tutte le date antecedenti il 1° febbraio 2021. Dato che per ogni mese ci sono diverse tipologie di vaccini è stata effettuata una somma delle vaccinazioni attraverso reduceBykey.
- *JavaPairRDD<Tuple3<Mese,Area,FasciaAnagrafica>, NumeroGiorni>:* in questa coppia RDD viene calcolato il numero di giorni corrispondente alla chiave: mese, area, fascia anagrafica.
- *Broadcast<List<Tuple3<Mese,Area,FasciaAnagrafica>>>:* le variabili broadcast sono variabili condivise di sola lettura memorizzate nella cache e disponibili in tutti i nodi di un

cluster per accedere o utilizzare le attività. Invece di inviare questi dati insieme a ogni attività, spark distribuisce variabili broadcast alla macchina utilizzando algoritmi di trasmissione efficienti per ridurre i costi di comunicazione. È stato necessario utilizzare questa variabile per poter filtrare la tupla di chiavi il cui numero di giorni di vaccinazioni risulti minore di tre;

- *JavaPairRDD<Tuple3<Mese,Area,FasciaAnagrafica>,Iterable<Data,NumeroVaccini>>:* necessario per raggruppare i valori e ritornare una lista per ogni chiave;
- *JavaPairRDD<Tuple3<Mese,FasciaAnagrafica, Predizione>, Area>:* attraverso l'algoritmo di regressione lineare, in particolare Simple Regressione, è stato possibile predire per ogni mese e per ogni fascia anagrafica il numero di vaccinazioni per il mese successivo. Inoltre, ritorna per ogni mese e per ogni fascia anagrafica le prime cinque aree con il numero di vaccinazioni maggiori.

IV. ESECUZIONE

Come detto in precedenza, è stato utilizzato Docker compose per poter avviare i servizi necessari per l'esecuzione del progetto. Inoltre, all'interno dello script è stato inserito un comando:

docker compose up –scale spark-worker=2 –scala hdfs-datanode=2

per specificare il numero di worker che si vogliono istanziare.
Prima di poter eseguire lo script è necessario effettuare mvn package per la creazione di un file jar.

Prima di sottomettere la query a Spark, sono stati caricati i file csv necessari per l'esecuzione delle query ed è stata creta una cartella per inserire i risultati ottenuti alla fine del processamento dei dati su hdfs tramite i comandi:

docker cp data hdfs-namenode:/data/
docker exec -it hdfs-namenode hdfs dfs -put /data /data
docker exec -it hdfs-namenode hdfs dfs -mkdir /results
docker exec -it hdfs-namenode hdfs dfs -chmod 0777 /results

Successivamente, viene copiato il file jar, generato dal mvn package, all'interno di una directory che corrisponde al volume specificato all'interno del file docker-compose.
A questo punto è possibile eseguire il comando che permette di sottomettere le query a spark per la sua esecuzione attraverso:

docker exec spark-master /bin/bash -c "spark-submit" --class queries.Main --master "spark://spark-master:7077" /spark_data/Progetto-1.0-SNAPSHOT.jar

Alla fine della sua esecuzione verranno caricati i risultati ottenuti all'interno della cartella results su hdfs.

Durante l'esecuzione di Spark è possibile visualizzare il DAG Visualization di entrambe le query.
In totale vengono eseguite quattordici jobs, di cui sei jobs per la query uno e otto per la seconda query. Inoltre, possiamo notare che il DAG scheduler separa le operazioni in stage multipli, e per ogni job a volte i task precedenti vengono inseriti nel DAG del job corrente anche se vengono skippati; per questo motivo possiamo notare un alto numero di stages.

Di seguito, una porzione del DAG Visualization per la prima query nello stage finale:

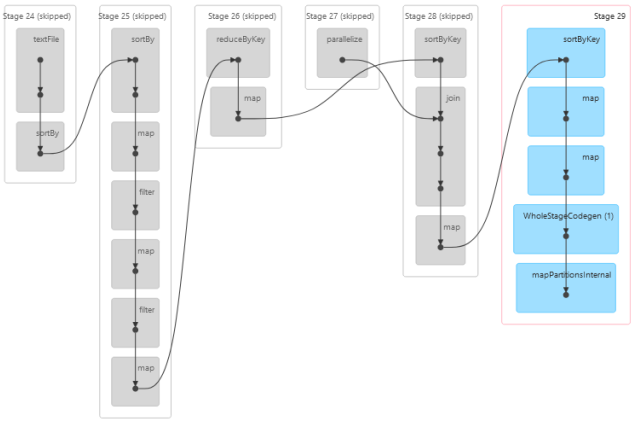

Figura 3- DAG Visualization - Query Uno

Di seguito, una porzione del DAG Visualization per la seconda query nello stage finale:

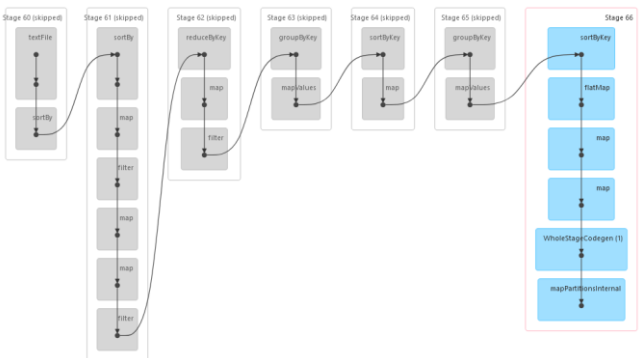

Figura 4 - DAG Visualization - Query Due

V. Analisi della performance

Le query sono state eseguite su un processore Intel(R) Core(TM) i7-6500U CPU @ 2.50GHz 2.60 GHz.

I tempi di esecuzione variano a seconda della query; inoltre, sono state eseguite con un numero diverso di spark worker da 1 a 4 e per due run consecutivi.

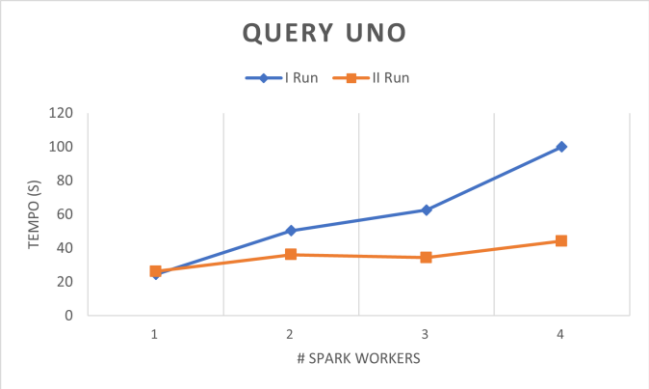

Figura 5- Performance - Query Uno

Per la prima query possiamo notare come il tempo di esecuzione aumenti all'aumentare del numero di spark worker utilizzati; nella seconda run possiamo invece notare che i tempi di esecuzione diminuiscono rispetto alla prima.

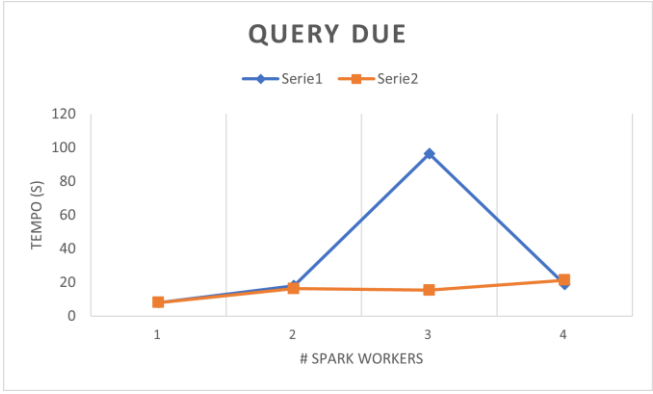

Figura 6 - Performance - Query Due

Per la seconda query, è possibile notare che i tempi di esecuzione sono molto simili tra i due run, tranne quando vengono utilizzati tre spark worker con un tempo di esecuzione della prima run maggiore rispetto alla seconda.

VI. Conclusioni

In conclusione, possiamo notare come all'aumentare del numero di spark workers aumenti anche il tempo di esecuzione per ogni query, e diminuisce all'aumentare dei run.

VII. References

[1] https://github.com/italia/covid19-opendata-vaccini/tree/master/dati