Progetto I

SABD 19-20

Marco Giorgi
Matricola: 0266724
marco.giorgi90@outlook.it

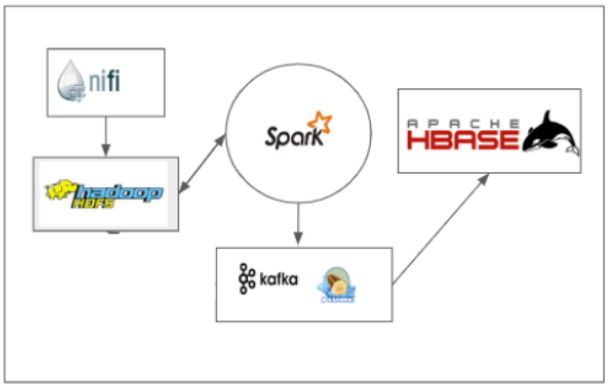
Andrea Conidi
Matricola: 0277418
conidi.andrea7@gmail.com

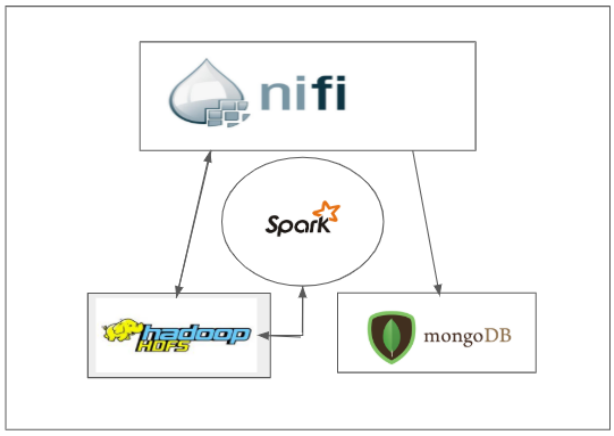
ABSTRACT

L'obiettivo del progetto è quello di soddisfare le query assegnate utilizzando alcuni dei framework trattati durante la prima parte del corso di SABD. Nella seguente relazione verranno mostrate l'architettura, le scelte progettuali e i risultati sia in termini di metriche che di progetto conseguenti allo sviluppo.
Il primo step affrontato durante lo sviluppo è stato quello di pre-processare i dati provenienti dai dataset necessari ai fini del progetto: sono stati filtrati i dataset selezionando solo l'insieme dei dati utili, in modo tale da ridurre i tempi di processamento. La scelta progettuale adottata in questa prima fase è stata quella di non effettuare operazioni onerose, come ad esempio rendere i dati puntuali, in modo da ridurre i tempi di pre-processing che incidono maggiormente rispetto a quelli di processamento. Come framework di pre-processing ed ingestion si è scelto di utilizzare Apache Nifi.
Nella fase di processamento si è scelto di sfruttare Apache Spark per le caratteristiche aggiuntive che mette a disposizione rispetto ad Hadoop Mapreduce: developer-friendly e utilizzo di data model come RDD e Dataframes.
Per la memorizzazione dei risultati sono stati proposti e realizzati due sistemi di ingestion:

1. Tramite Apache Flume e Apache Kafka per trasferire i dati da HDFS a HBase
2. Tramite Apache Nifi per trasferire i dati da HDFS a MongoDB.

In tutta la fase di progettazione si è scelto come riferimento per quanto riguarda il calcolo delle settimane le API del calendario gregoriano fornite dal linguaggio di programmazione scelto.
Per quanto riguarda il calcolo del coefficiente di trendline sono stati utilizzati i metodi della libreria fornita da Apache per il calcolo della regressione lineare.
In seguito vengono mostrate le due **architetture** ideate e implementate.





KEYWORDS

K-means Mllib, Algoritmo di Lloyd, Ingestion, Spark, Flume, Nifi, Kafka, Hbase, HDFS.

1 INGESTION e MEMORIZZAZIONE

1.1 Ingestion to HDFS

Una volta effettuato l'upload dei dataset in formato csv, tramite **Apache Nifi**, i dati sono stati filtrati attraverso un

processor chiamato **ExecuteStreamCommand** che permette mediante uno script ***python*** di leggere il set di dati, filtrarlo e restituire in output un nuovo dataset. Quest'ultimo sarà salvato direttamente su HDFS tramite il processor di Nifi **PutHDFS**.

Per il dataset necessario alla prima query sono state selezionate solo le colonne relative alla data, ai dimessi_guariti e ai tamponi.
Per il dataset necessario alle altre due query si è eseguito il replacing dei dati contenenti il carattere virgola per poter poi effettuare correttamente il parsing dei dati nella fase di processing.
Per la realizzazione dei risultati della seconda query si è scelto di utilizzare un ulteriore dataset sviluppato a partire dalla ISO 3166 contenente per ogni stato il continente di appartenenza.
Dopo il processing dei dati la memorizzazione ad HDFS è stata fatta sfruttando il metodo *saveAsTextFile* fornito da Spark in Java.

1.2 Transfert to HBase

Una volta terminato il processing e memorizzati i risultati all'interno di HDFS, tramite la realizzazione di un producer di **Kafka** e tramite le configurazioni di **Apache Flume,** è stato possibile trasferire i dati all'interno di una tabella di **HBase** disponibile all'interno di un container docker. Per fare ciò è stato necessario configurare all'interno di un file conf di flume un agent contenente come source kafka, come sink hbase e un channel in memory. Per poter eseguire l'ingestion in HBase è stata creata una tabella denominata 't' con famiglia di colonne 'c' tramite lo script in linea di comando:

>create 't', 'c'

1.2 Transfert to MongoDB

In alternativa a HBase si è scelto di effettuare ingestion dei risultati da HDFS a **MongoDB** utilizzando i processor di Apache Nifi che permettono di eseguire il fetching dei risultati da HDFS per poi trasferirli su MongoDB.

2 PROCESSING

Il processing è stato realizzato attraverso un progetto maven scritto in linguaggio **JAVA** utilizzando le API fornite da **Apache Spark**.

2.1 Query1

Con l'obiettivo di calcolare il numero medio di guariti e tamponi per ogni settimana in Italia si è scelto di utilizzare come data model gli RDD a partire dal primo dataset filtrato con Nifi.

Una volta creato lo ***Spark Context*** e parsato i dati da HDFS, il primo passo è stato quello di realizzare degli RDD contenenti i dati non cumulativi: sono stati realizzati RDD con chiave shiftata di un giorno a partire da quelli creati in precedenza per poter effettuare tramite una *join* una trasformazione che rendesse i dati puntuali.

Il passo successivo è stato quello di mappare gli RDD tale da avere come chiave il numero della settimana (al posto della data) e come valori il numero di guariti e tamponi, per poi tramite *reduceByKey* ottenere un RDD contenente la somma dei guariti e dei tamponi effettuati per ogni settimana.

Dato che non tutte le settimane potrebbero contenere lo stesso numero di giorni, si è calcolato un *count* delle chiavi per ottenere il numero di occorrenze per ogni chiave.

Per calcolare la media si è eseguita una operazione di *join* degli RDD ottenuti da entrambe le reduce precedenti e tramite una operazione di *map* si è ricavato un RDD contenente il valore medio dei guariti e dei tamponi per ogni settimana.

2.2 Query2

Partendo dal secondo dataset è stato creato un RDD che successivamente è stato filtrato e adattato per il processing. Come nel dataset della Query1 si è scelto di eliminare il dato cumulativo durante il parsing dei dati, mentre come soluzione al problema dei dati decrescenti (quindi errati) si è scelto di sostituire, nel caso di differenza negativa tra un dato giorno e il precedente, il valore 0 così che in quel giorno non risultassero nuovi contagiati. Es: *Francia [167605, 165093]* diventa *Francia [valore puntuale, 0]*. Fatto ciò, attraverso il calcolo del **trend** su tutto il periodo di riferimento sono stati selezionati i primi 100 stati per numero di contagiati. Data la necessità di associare gli Stati ai relativi continenti è stato utilizzato un ulteriore dataset contenente per ogni nazione il relativo **continente**. Attraverso l'operazione di *filter* sul RDD iniziale sono stati selezionati i primi 100 stati in base al trend, poi con una operazione di *join* è stato possibile ottenere un nuovo RDD in cui l'array di valori di ognuno dei 100 stati è associato al relativo continente es: <Europe,[1,2,3]>. Con tale RDD tramite reduce sul continente si è ricavato un RDD con valore un array relativo alla somma giornaliera del numero di contagiati, ad esempio <Europe, [1,2,3]> , <Europe, [3,4,5]> si riducono in <Europe, [4,6,8]>.

Con una operazione di *flatMap* si è trasformato l'RDD precedente in RDD formato da: continente, settimana relativa al giorno, numero contagiati totali in quel giorno, es: da <Europe, [3,6,8]> otteniamo <<Europe, W1>,3>, <<Europe, W1>,6>, <<Europe, W2>,8>.

Infine, sono state calcolate le statistiche richieste dalla query:

1. Media, dal rapporto numero contagiati settimana su numero totale giorni della settimana ottenuti tramite due *reduce*

2. Max e Min, attraverso i metodi forniti dalla libreria Math

3. Deviazione Standard.

2.3 Query3-3a

In questa query l'obiettivo è stato quello di calcolare per ogni mese i primi 50 stati più colpiti secondo il coefficiente di trendline e successivamente applicare un **algoritmo di clustering K-Means**: in un caso fornito da **Spark MLlib** e nell'altro sviluppando in modo **naive**.

Per fare ciò sono state realizzate due classi che differiscono solamente dal tipo di algoritmo adottato. Per la fase di processing iniziale si è scelto di trattare i dati nello stesso modo della query precedente.

Data la necessità di ottenere un coefficiente di trendline mensile, è stata eseguita una trasformazione generando un RDD <<Stato,Mese>,<Giorno,Valore>>. Il passo successivo è stato raggruppare per chiave e trovare il trend mensile per ogni stato.

In seguito, per determinare i primi 50 stati per ogni mese in base al coefficiente di trend, sono stati fatte le seguenti trasformazioni in un ciclo *for* iterato per il numero di mesi:

1. è stato filtrato l'RDD del passo precedente nel mese corrispettivo del ciclo
2. una *flatMap* per ottenere RDD del tipo <Trend, Stato>
3. una *sortByKey* e una *take* per ottenere i primi 50 stati in base al coefficiente di trend

Il risultato ottenuto alla fine del ciclo iterativo è stata una lista mensile che è stata trasformata tramite *parallelize* in un RDD contenente Mese e un insieme di valori contenenti trend e stato. Quest'ultimo RDD è stato il punto di partenza per i due algoritmi.

2.3.1 Query3

Per ogni mese, dopo aver filtrato opportunamente l'RDD sorgente, tramite algoritmo clustering naive è stato assegnato per ogni stato il relativo cluster di appartenenza. Per fare ciò è stato implementato l'**algoritmo k-means di Lloyd** scegliendo i 4 centroidi iniziali in modo **random** tra l'insieme di valori contenuti nel RDD. Successivamente si sono ripetuti i passi dell'algoritmo con un numero di iterazioni stabilito a priori (es. 20).

2.3.2 Query3a

A differenza della classe precedente, è stato utilizzato l'algoritmo di k-means fornito dalla libreria MLib di Spark, quindi per ogni mese si è trasformato l'insieme dei valori in RDD di tipo **Vector** specifico della libreria usata. I valori così ottenuti sono stati utilizzati per eseguire il **training** di un **k-means model** specificando il numero di cluster da utilizzare e il numero di iterazioni dell'algoritmo. Infine il modello addestrato precedentemente è stato sfruttato per ottenere tramite il metodo ***predict*** l'appartenenza mensile della coppia stato e trend ad uno dei 4 cluster. I risultati ottenuti dalle due classi sono stati parallelizzati in un RDD composto da : <Mese,Stato,Cluster> così da essere pronto per la memorizzazione finale.

3 PERFORMANCE

Ambiente di sviluppo:

Intel core i7 8th gen, 16gb RAM, SO : Ubuntu 20.04 LTS su VM.

Intel core i7 5th gen, 8gb RAM, SO: Ubuntu 18.04 LTS

I tempi sono stati calcolati utilizzando la funzione *nanoTime* fornita da Java, in particolare il tempo è stato misurato durante tutto il periodo di vita dello *SparkContext*, dalla sua inizializzazione al suo stop. Nella prima tabella sono stati riportati i tempi medi della fase di processing per ciascuna query processata 10 volte, nel dettaglio i tempi medi sono stati calcolati sia considerando il primo tempo, in cui avviene anche la creazione delle **JVM,** sia i tempi "puri" di processamento.

Query	Avg [s]	Avg (without first) [s]
Query1	1,33	0,75
Query2	3	1,67
Query3	13,9	11,77
Query3a	10,5	8,55

I valori calcolati sono presi considerando anche i tempi di ingestion ad HDFS.

Dai dati riportati è possibile constatare che i tempi di processing della query 3, in cui è implementato l'algoritmo naive, sono maggiori di quelli trascorsi per la finalizzazione della query 3a sviluppata usando l'algoritmo k-means di Mlib.

Nella tabella seguente sono riportati:

nella prima colonna i tempi di ingestion in locale ad HDFS usando il framework di data ingestion Nifi, i valori sono stati forniti graficamente dai processor di Nifi.

Nella seconda colonna i tempi di ingestion in HBase usando i **framework Flume + Kafka**, tenendo in considerazione il **timestamp** fornito da bash di Flume.

	Ingest Local To HDFS	HDFS To Hbase	HDFS to Mongodb
Dataset 1	343 ms		
Dataset 2	373 ms		
Result Query1		151 ms	2.640 s
Result Query2		394 ms	3.108 s
Result Query3		2,226 s	2.698 s
Result Query3a		1,162 s	2.394 s

3 RISULTATI

Nella seguente sezione è riportato il formato dei risultati ottenuti dopo l'esecuzione delle query (gli stessi sono presenti nella cartella **result** del progetto).

Query1:

Week,Healed,Swabs,Days
W10,77.0,4115.714285714285,7

Query2:
Continent,Week,Max,Min,Avg,DevStd
europe,W19,28654,21624,24458.0,3026.9661379011163

Query3-3a:
Month,State,Cluster
1,hubei,2

4 CONCLUSIONI

Nella parte di ingestion iniziale per trasferire i dati su HDFS si è scelto di utilizzare un approccio diretto, cioè Nifi, anche se era possibile utilizzare un approccio basata su Kafka, ovvero pubblicare i dati su un topic ed usare un framework Kafka connect per trasferire i dati su HDFS, oppure un approccio basato su Kafka+Flume. Sebbene questi ultimi approcci risultano più performanti, la grandezza dei dataset elaborati da Nifi non hanno dimensioni notevoli, per tanto il tempo di trasferimento su HDFS è stato abbastanza soddisfacente, inoltre utilizzare più framework può alcune volte portare latenze considerevoli.
Per quanto riguarda la memorizzazione verso Hbase è stato scelto di usare un framework composto da Flume e Kafka per motivi di performance ma soprattutto di affidabilità, infatti gli eventi nel channel dell'agent configurato in Flume sono rimossi solo dopo che questi sono consegnati alla destinazione. Flume si è rivelato difficile per la definizione dell'environment, ma una volta configurato l'agent il trasferimento dei dati elaborati avviene in maniera "fluida". Dall'analisi dei tempi, infatti, è possibile osservare come il trasferimento degli output utilizzando l'approccio Nifi-Mongo risulti avere latenze maggiori rispetto all'approccio Flume-Hbase.

Note:

- I dati processati fanno riferimento fino al giorno 6 maggio 2020, quindi le successive modifiche al dataset non sono state considerate.
- Gli RDD a cui si fa riferimento nel testo sono sia JavaRDD che JavaPairRDD.