Sistemi e Architetture per Big Data - AA 2020/2021

Primo progetto

Giuseppe Lasco
Dipartimento di Ingegneria dell'Informazione
Università degli studi di Roma "Tor Vergata"
Roma, Italia
giuseppe.lasco17@gmail.com

Marco Marcucci
Dipartimento di Ingegneria dell'Informazione
Università degli studi di Roma "Tor Vergata"
Roma, Italia
marco.marcucci96@gmail.com

***Abstract*—Questo documento riporta i dettagli implementativi riguardanti l'analisi mediante *Spark* dei dataset contenenti informazioni relative all'andamento nazionale italiano dei vaccini effettuati. Viene, inoltre, descritta l'architettura a supporto dell'analisi e gli ulteriori *framework* utilizzati.**

I. Introduzione

L'analisi effettuata si pone lo scopo di valutare delle statistiche relative ai vaccini contro il COVID-19, su dati resi disponibili dal Commissario straordinario per l'emergenza Covid-19, Presidenza del Consiglio dei Ministri.

Dataset

Il primo file preso in considerazione è *punti-somministrazione-tipologia.csv*, il quale contiene dati sui punti di somministrazione per ciascuna Regione e Provincia Autonoma.

Il secondo file preso in considerazione è *somministrazioni-vaccini-latest.csv*, il quale contiene dati sulle somministrazioni giornaliere dei vaccini suddivisi per regioni, fasce d'età e categorie di appartenenza dei soggetti vaccinati. Tale dataset risulta ordinato per data, inoltre è stata riscontrata l'assenza di numerose tuple relative a delle specifiche regioni, fasce d'età e mesi. Questo fenomeno ha reso necessario un intervento di preprocessamento utile a inserire date mancanti per rendere più accurato il lavoro di regressione sui dati, sotto l'assunzione che i dati mancanti fossero dovuti all'assenza di vaccinazioni in un determinato giorno.

Il terzo file preso in considerazione è *somministrazioni-vaccini-summary-latest.csv*, il quale contiene dati sul totale delle somministrazioni giornaliere per regioni e categorie di appartenenza dei soggetti vaccinati. Il dataset in questione risulta, invece, non ordinato, per cui si è reso necessario un effort di preprocessamento al fine di ordinarlo.

L'ultimo file preso in considerazione è *totale-popolazione.csv*, che tiene traccia della popolazione totale residente in una data Regione o Provincia Autonoma.

Query

L'obiettivo di questo progetto è quello di implementare ed eseguire tre query utilizzando *Spark*.

La prima query ha come scopo quello di calcolare il numero medio di vaccinazioni giornaliere in ciascun centro di ciascuna area.

La seconda consiste nel determinare le prime 5 aree per le quali è previsto il maggior numero di vaccinazioni il primo giorno del mese successivo, per le donne, per ogni fascia anagrafica e per ogni mese solare. A tale scopo si utilizza una retta di regressione, addestrata sui dati relativi al mese precedente a quello per cui viene fatta la predizione al primo giorno. I dati presi in considerazione partono dal 1 Febbraio 2021.

L'ultima query prevede di effettuare una previsione della percentuale totale delle somministrazioni dei vaccini al 1 Giugno 2021 per ogni regione, utilizzando tutti i dati relativi ai mesi precedenti, a partire dal 27 Dicembre 2020. Inoltre, vengono utilizzati due algoritmi di clustering in grado di raggruppare le Regioni in base alla previsione sopra citata.

Framework

Come *framework* di processamento batch è stato utilizzato *Apache Spark* che comunica con lo storage distribuito *Hadoop Distributed File System*. Per la raccolta dei risultati è stato impiegato *HBase*, uno storage No-SQL column family. Infine, come *framework* di data ingestion è stato utilizzato *NiFi*.

II. Architettura

L'architettura si compone di un insieme di container *Docker*, su cui eseguono i servizi introdotti precedentemente. Inoltre, sempre sulla stessa macchina, una JVM ospita l'esecuzione di *Apache Spark*. I container comunicano attraverso la stessa rete, creata appositamente.

NiFi

NiFi è il servizio che permette di recuperare i dataset in formato *comma separated value* da *GitHub* e dal server dell'università, trasformarli in formato *parquet* e inviarli al servizio di storage distribuito *HDFS*. Inoltre, tale servizio si occupa di eliminare le colonne considerate di scarso interesse ai fini dell'applicazione, conservando quelle utili, anche per eventuali aggiornamenti delle query o aggiunta di nuove. Tale *framework* è stato istanziato su container *Docker* utilizzando l'immagine *apache/nifi*. L'uso di *parquet* ha permesso di

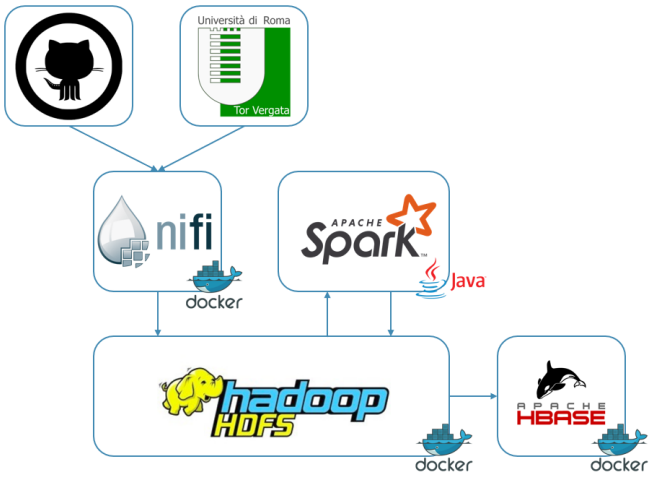

Fig. 1: Schema dell'architettura

comprimere i dati migliorando le prestazioni in termini di occupazione di memoria.

Al fine di eseguire le operazioni elencate, sono stati impiegati tre *processori*, uno che permette di collegarsi ai servizi di hosting, ovvero *GitHub* e il server dell'università, e scaricare i dati, uno che permette l'eliminazione delle colonne e uno che permette la trasformazione in *parquet* di questi ultimi e l'upload su *HDFS*. La struttura è definita mediante il template in figura 2.

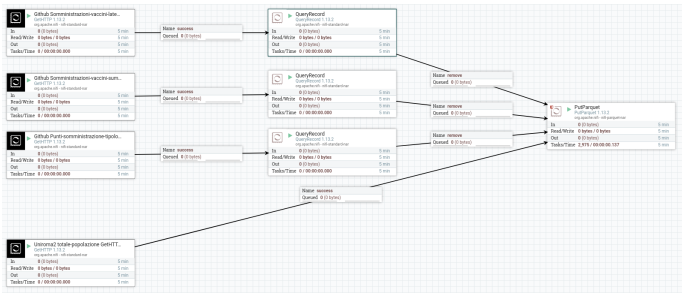

Fig. 2: NiFi template

HDFS

HDFS rappresenta il mezzo che permette l'archiviazione dei dati in maniera distribuita. Il servizio si compone di un nodo *master* e tre nodi *worker* con un livello di replicazione pari a 2. Tale servizio permette di rendere disponibili i dati su cui *Spark* esegue la computazione e memorizza gli *output* dell'analisi, che vengono, in seguito, esportati su *HBase*, per eventuali analisi, manipolazione e rappresentazione dei dati. Il deployment del framework avviene attraverso l'utilizzo dell'immagine *Docker effeerre/hadoop*, istanziata su container. In seguito all'avvio del servizio, uno script permette di eseguire lo sturtup del *Namenode* e dei *Datanode*, e crea le directory `/data`, dove *NiFi* inserisce i dati, e `/output`, in cui risiedono i risultati dell'analisi, concedendo i permessi di lettura, scrittura ed esecuzione.

Spark

Al fine di preprocessare i dati ed eseguire le *query*, viene utilizzato *Apache Spark* in locale, tramite lo script `$SPARK_HOME/bin/spark-submit`. Oltre allo *Spark Core*, che espone un set di API di *trasformazioni* ed *azioni*, è stata impiegata la libreria di *Machine Learning MLlib*, utile per effettuare *clustering* sui risultati della terza *query*.

HBase

HBase è stato utilizzato come datastore *NoSQL* sul quale importare i risultati delle query, anche questo servizio è stato istanziato utilizzando un container *Docker* realizzato, questa volta, a partire dall'immagine *harisekhon/hbase*. Affinchè fosse possibile l'esportazione dei risultati da *HDFS* a *HBase*, è stata creata la classe `HBaseQueries.java` che permette la creazione delle tabelle e l'inserimento dei dati, sfruttando la classe `HBaseClient.java`. Quest'ultima contiene informazioni riguardo la configurazione di *HBase* e *Zookeeper*, e le principali operazioni di gestione del datastore.

III. Query

Query 1

Al fine di soddisfare la seguente *query*, si è reso necessario l'utilizzo di due file, *somministrazioni-vaccini-summary-latest.parquet* e *punti-somministrazione-tipologia.parquet*.

Tali file sono stati caricati in `Dataset` e trasformati in `JavaPairRDD`, considerando le sole colonne di interesse: *data_somministrazione*, *area* e *totale* per il primo e *area* per il secondo. È stato effettuata la *trasformazione "filter"*, scartando i dati precedenti al 1 Gennaio 2021 e successivi al 31 Maggio 2021, seguito da un'ordinamento dell'`RDD` *somministrazioni-vaccini-summary-latest* in base alla data. Una *trasformazione* di *"reduceByKey"* ha permesso di ottenere il totale di vaccinazioni per ogni mese. Utilizzando un approccio simile al *word count*, sono stati contati i centri riferiti ad una determinata Regione relativi all'`RDD` *punti-somministrazione-tipologia*. Successivamente, la *"join"* ha permesso di unire i due `RDD`, utilizzando come chiave la Regione. Il risultato finale è stato ottenuto dividendo il totale per il numero di giorni del mese di riferimento e per il numero di centri della regione di riferimento, ordinando, infine, il risultato in termini di mese e regione.

In figura 3 è possibile osservare lo scheduling di Spark dei passi appena descritti attraverso il *DAG*.

In figura 4 è possibile osservare *l'event timeline* che descrive il susseguirsi degli eventi rispetto al tempo, evidenziando il parallelismo di alcune operazioni.

Query 2

Relativamente alla seconda *query* è stato utilizzato il file *somministrazioni-vaccini-latest.parquet*, il quale, in seguito al caricamento da *HDFS*, è stato trasformato in `JavaPairRDD`. Durante questa fase sono state scartate le colonne irrilevanti ai fini della richiesta. La *trasformazione "filter"* ha permesso l'eliminazione delle entry relative a date precedenti al 1

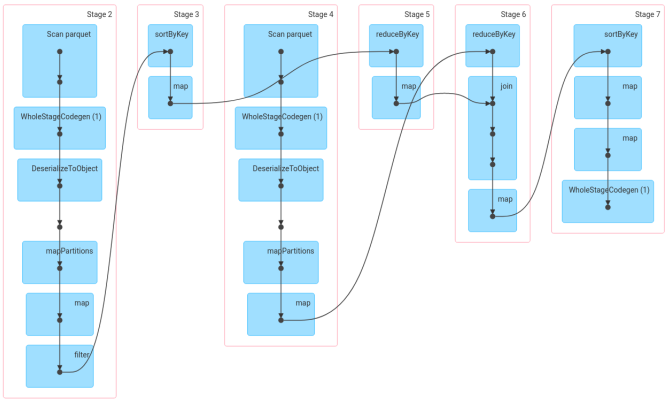

Fig. 3: DAG query 1

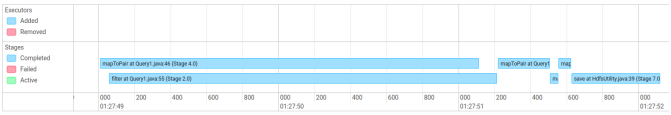

Fig. 4: Event Timeline query 1

Febbraio 2021 e successivi al 1 Giugno 2021. Considerando come chiave la tupla *area, data e fascia anagrafica* sono stati sommati i vaccini relativi ad aziende farmaceutiche differenti, ordinando, in seguito, per data. La *trasformazione "groupByKey"* è stata applicata al fine di raggruppare tutte le tuple *data, numero somministrazioni giornaliere* relative ad una certa regione e fascia anagrafica. Per ogni mese è stato eseguito una operazione di inserimento di date e valori macanti ed è stata effettuata *regressione lineare*, in modo da prevedere il numero di donne vaccinate al primo giorno del mese successivo. Quest'ultimo passo ha previsto operazioni di raggruppamento e ordinamento. Il modello di regressione lineare è stato addestrato attraverso l'implementazione fornita dalla libreria di regressione di `Apache Commons`.

In figura 5 e 6 è possibile osservare rispettivamente il *DAG* e *l'event timeline* relativi alla seconda *query*.

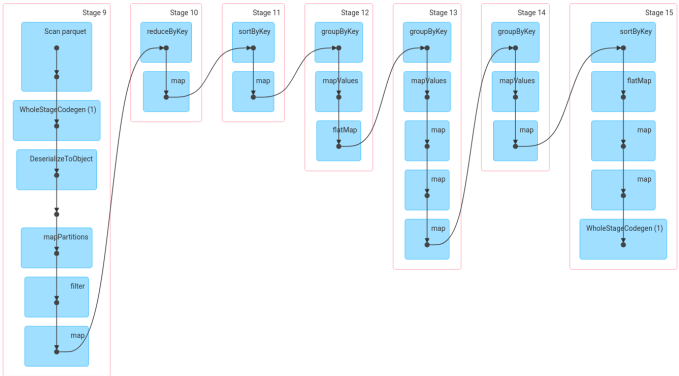

Fig. 5: DAG query 2

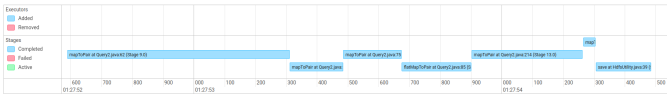

Fig. 6: Event Timeline query 2

Query 3

L'ultima *query* fa uso dei dati presenti nei file *somministrazioni-vaccini-summary-latest.parquet* e *totale-popolazione.parquet*. In seguito è stato effettuato il caricamento dei file e la trasformazione in `JavaPairRDD`. Sui dati relativi a *somministrazioni-vaccini-summary-latest* si è proceduto al raggruppamento delle tuple *data, numero somministrazioni giornaliere* per ogni regione, questa operazione ha permesso di svolgere regressione lineare su tutti i giorni dal 27 Dicembre 2020 al 31 Maggio 2021, in modo da prevedere il numero di vaccini effettuati in data 1 Giugno 2021. Una *"reduceByKey"*, sulle regioni del medesimo file, ha, invece, permesso di calcolare il totale di vaccini effettuati dal 27 Dicembre 2020 al 31 Maggio 2021. Infine, l'operazione di somma tra le proiezioni e il totale calcolato, consentita dall'operazione di *"join"*, ha decretato il numero totale previsto di vaccinati per regione al 1 Giugno 2021. Il *"join"* tra l'*RDD* in questione e quello contenente il numero totale di abitanti residenti in ciacuna regione (*totale-popolazione*), he reso possibile calcolare la percentuale prevista di vaccinati complessivi al 1 Giugno 2021. Utilizzando due modelli presenti in *MLlib*, è stato effettuato *clustering* utilizzando i risultati precedenti come dataset, con numero di cluster variabile da 2 a 5. Gli algoritmi utilizzati sono *K-means* e *Bisecting K-means*, mentre per la regressione è stata utilizzata l'implementazione fornita dalla libreria di regressione di `Apache Commons`.

In figura 7 e 8 è possibile osservare il *DAG* e *l'event timeline* relativi alla terza *query*. In riferimento agli stage 18 e 19 del DAG, è possibile notare l'uso dell'operazione `cache` sull'*RDD* filtrato relativo a *somministrazioni-vaccini-summary-latest*, per via dell'impiego indipendente di quest'ultimo negli stage 20 e 21. Come si può inferire dall'*event timeline*, alcuni stage vengono eseguiti parallelamente.

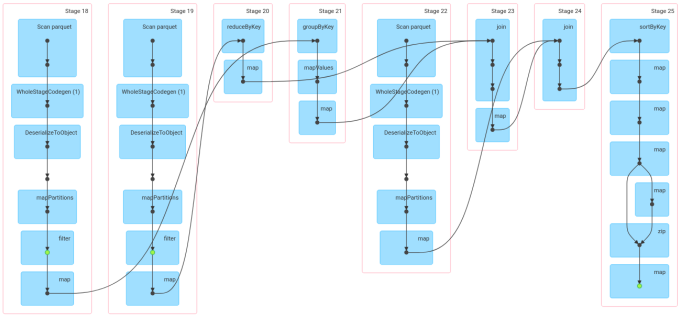

Fig. 7: DAG query 3

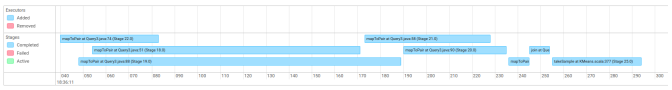

Fig. 8: Event Timeline query 3

IV. BENCHMARK

L'esecuzione del progetto e la valutazione delle prestazioni sono state eseguite su *Linux Xubuntu* basato su *Ubuntu 20.04 LTS* virtualizzato tramite *VBox*, *CPU AMD Ryzen 5 3600*, 6 core, 12 thread (di cui 10 assegnati alla VM) e 16 GB di RAM (di cui 9 assegnati alla VM), con archiviazione su *SSD*.

TABLE I: Tempi esecuzione query

Query	Media	Varianza
Query 1	221.1	41.1
Query 2	925.2	34.2
Query 3	2778.4	260.3

*I tempi sono espressi in millisecondi.

In tabella I sono riportati i tempi di processamento delle query. Sono state considerate le performance al netto di startup della *Java Virtual Machine* su cui *Spark* opera e caricamento dei file dall'*HDFS*. I tempi scrittura dei risultati sul *file system distribuito*, invece, sono stati considerati, in modo da rendere effettive tutte le operazioni (trasformazioni) eseguite da *Spark*. Come si può notare, la query 3 risulta molto più lenta delle altre due, che, invece, mostrano risultati migliori. Tale evidenza è causata dall'inclusione, nel totale, dei tempi di addestramento degli algoritmi di *clustering*, che possono essere osservati in tabella II.

TABLE II: Tempi esecuzione clustering

Numero cluster	Modello			
	K-means		Bisecting K-means	
	Media	Varianza	Media	Varianza
2	347.3	43.6	203.2	62.4
3	136.2	21.6	127.1	24.8
4	153.1	39.5	148.2	25.5
5	148.9	51.9	144.3	42.1

*I tempi sono espressi in millisecondi.

Sempre dalla tabella II è possibile notare che i tempi di addestramento del primo modello (*K-Means con numero di cluster pari a 2*) risultano superiori a quelli relativi alle successive combinazioni. Questo dovuto al caching effettuato da MLlib dopo la prima esecuzione del modello di clustering, come si evince dalla figura 9 che raffigura il DAG della seconda esecuzione di tale modello.

In tabella III, invece, è possibile osservare come il *Within Set Sum of Squared Error* medio, per i due algoritmi di *clustering*, decresca al crescere del numero di cluster, anche se questo non è scontato per via dell'inizializzazione randomica dei centroidi.

TABLE III: Costo clustering: WSSSE

Numero cluster	Modello	
	K-means	Bisecting K-means
2	0.005392	0.005639
3	0.003129	0.002325
4	0.001469	0.001585
5	0.000751	0.000751

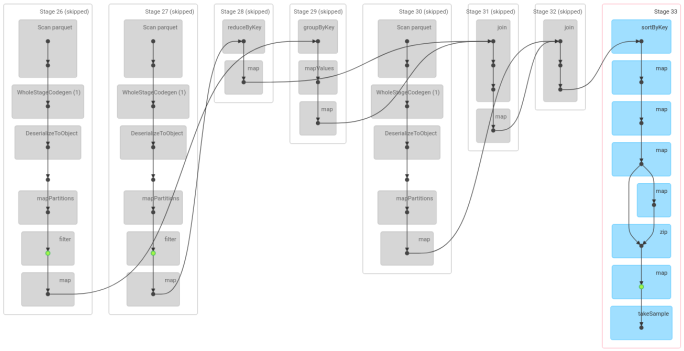

Fig. 9: DAG query 3, secondo run cluster

REFERENCES

[1] https://spark.apache.org/docs/latest/
[2] https://stackoverflow.com/
[3] https://nifi.apache.org/
[4] https://hadoop.apache.org/docs/stable/
[5] http://spark.apache.org/docs/latest/ml-guide.html
[6] https://commons.apache.org/proper/commons-math/javadocs/api-3.3/org/apache/commons/math3/stat/regression/SimpleRegression.html