UNIVERSITÀ DEGLI STUDI DI SALERNO

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE ED ELETTRICA E MATEMATICA APPLICATA

Corso di Laurea Magistrale in Ingegneria Informatica

Big Data: Hadoop MapReduce & Spark on BMW5 dataset

Report

Nome	Cognome	Matricola	E-Mail
Emanuele	Relmi	0622702368	e.relmi@studenti.unisa.it

ANNO ACCADEMICO 2024/2025

INDICE

CAPITOLO 1

INTRODUZIONE

1.1 Obiettivo

Il progetto ha come obiettivo l'analisi di un dataset reale attraverso l'utilizzo di due diversi framework distribuiti: **Apache Hadoop MapReduce** e **Apache Spark**.
La traccia prevede infatti due esercizi distinti:

1. **Hadoop MapReduce** (Esercizio 1) – realizzare un programma che analizzi i dati del dataset (statistiche, classifiche, indici, ecc.) mediante almeno tre trasformazioni;
2. **Apache Spark** (Esercizio 2) – sviluppare un'applicazione che, per ogni gruppo di età, individui il modello di auto BMW più venduto.

1.2 Dataset

Il dataset fornito, denominato `BMW_Car_Sales_Classification.csv`, contiene informazioni relative a clienti e iniziative di marketing, con l'obiettivo di analizzare i fattori che influenzano la vendita di automobili BMW.
I campi principali includono:

- `Model`: modello specifico della vettura (es. 5 Series, i8, X3, ecc.);
- `Year`: anno di produzione;
- `Region`: area geografica di riferimento (Asia, North America, Middle East, ecc.);
- `Color`: colore della vettura;

- `Fuel_Type`: tipologia di carburante (Petrol, Diesel, Hybrid, ecc.);
- `Transmission`: tipo di trasmissione (Manual, Automatic);
- `Engine_Size_L`: cilindrata del motore espressa in litri;
- `Mileage_KM`: chilometraggio percorso (in kilometri);
- `Price_USD`: prezzo dell'auto (in dollari americani);
- `Sales_Volume`: volume di vendite registrato;
- `Sales_Classification`: classificazione qualitativa delle vendite (`High Low`).

Il file è in formato CSV con separatore di campo "`,`" (*comma*). Per l'elaborazione, il job Hadoop e il job Spark ignorano la prima riga (*header*).

1.3 Docker Environment

Per l'esecuzione del progetto è stato predisposto un ambiente distribuito tramite **Docker**, che consente di replicare un cluster Hadoop (v. 3.3.6) in locale.
La configurazione si compone di:

- **Dockerfile**: definisce l'immagine base (`Ubuntu`) e installa i pacchetti necessari (`Java 8`, `Hadoop 3.3.6` e `OpenSSH` per la comunicazione interna);
- **docker-compose.yml**: avvia un cluster Hadoop minimale con un container `master` e tre container `slave`, collegati in una rete bridge interna (`hadoop_network`). Il master espone le porte 9870 (NameNode UI) e 8088 (YARN UI).
- **config/bootstrap.sh**: script di avvio nel container, che abilita SSH e può (se decommentato) lanciare `start-dfs.sh`.
- **config/hadoop-env.sh**: file di configurazione che definisce la variabile `JAVA_HOME`.
- **core-site.xml**: specifica la configurazione principale di Hadoop, inclusa la proprietà `fs.defaultFS`, ovvero l'URI del NameNode (`hdfs://master:54310`).
- **hddata/**: volume montato dal `docker-compose`, condiviso tra host e container come `/data`. Contiene dataset, sorgenti compilati, file JAR e output dei job.

All'interno del container `master`, l'utente può accedere a `/data` e da lì compilare ed eseguire i programmi Java. I file caricati su HDFS sono gestiti tramite i comandi `hdfs dfs -put`, `hdfs dfs -get`, `hdfs dfs -ls`, `hdfs dfs -cat` ecc.
Questa infrastruttura consente di eseguire esperimenti su larga scala simulando un cluster reale, ma mantenendo la semplicità di gestione offerta da Docker.

CAPITOLO 2

ESERCIZIO 1 – HADOOP MAPREDUCE

L'obiettivo del primo esercizio era realizzare un programma MapReduce che analizzasse i dati del dataset `BMW_Car_Sales_Classification.csv` applicando almeno tre trasformazioni. Per affrontare il problema è stata progettata una **pipeline composta da tre job sequenziali**, orchestrati dal driver `DriverBMWSales`. Ogni job prende in input l'output del precedente, con l'ultimo passo che produce il risultato finale richiesto.

2.1 Job 1: Aggregazione per regione e modello

Il primo job ha lo scopo di aggregare i dati di vendita per **coppia (regione, modello)**.

- **Mapper1:** legge ciascuna riga del file CSV (ignorando l'intestazione) ed emette come chiave la coppia `region-model`.
  Il valore associato è una tupla codificata nel formato `1|volume|prezzo|isHigh`, dove:
  - `1` rappresenta il conteggio di un record;
  - `volume` è il numero di unità vendute;
  - `price` è il prezzo della singola vendita;
  - `isHigh` vale 1 se la classificazione delle vendite è `High`, altrimenti 0.
- **Combiner1** e **Reducer1:** sommano i valori parziali generati dal mapper, producendo per ogni chiave (regione, modello) l'aggregato finale nel formato `count|sumVolume|sumPrice|highCount`.

L'output di questo job è, quindi, un insieme di statistiche aggregate per modello e regione.

2.2 Job 2: Calcolo dei totali per regione

Il secondo job utilizza l'output del Job 1 per calcolare i **totali complessivi delle vendite per ogni regione**.

- **Mapper2:** estrae da ciascuna riga la regione e il valore `sumVolume`;
- **Reducer2:** somma tutti i valori per regione e produce come output `region - regionTotalVolume`.

Questo passo fornisce i volumi totali regionali necessari per calcolare le percentuali di mercato.

2.3 Job 3: Top-K modelli per regione

Il terzo job combina i risultati dei due precedenti per ottenere statistiche avanzate e selezionare i **Top-K modelli per regione**.

- **Mapper3:** legge i dati del Job 1 e, tramite i totali regionali (caricati dall'output del Job 2), calcola per ogni modello:
  - la quota percentuale di mercato (`share%`);
  - il prezzo medio (`avgPrice`);
  - la percentuale di classificazioni alte (`highShare`).

  L'output del mapper ha come chiave la regione e come valore le metriche associate al modello.
- **Reducer3:** ordina i modelli di ciascuna regione in base al volume di vendite e ne seleziona i primi `K` (default 5).
  L'output finale ha la forma:

  `region  model  sumVol  share%  avgPrice  highShare`

2.4 Sintesi della pipeline

La catena di tre job realizza quindi il seguente flusso di trasformazioni:

1. aggregazione per `(regione, modello)`;
2. calcolo dei totali di vendita per regione;

3. calcolo delle metriche per modello e selezione dei Top-K.

Questa architettura permette di ottenere statistiche dettagliate sulle vendite BMW, evidenziando i modelli più rilevanti per ciascuna regione, sia in termini di quota di mercato che di prezzo medio.

CAPITOLO 3

ESERCIZIO 2 – SPARK

Il secondo esercizio prevede la realizzazione di un programma in **Apache Spark** che, per ogni gruppo di età (derivato dall'anno di produzione), individui il modello di automobile BMW più venduto. L'implementazione è stata sviluppata in Java, utilizzando le `RDD API` di Spark e le principali trasformazioni funzionali (`map`, `filter`, `reduceByKey`, `mapToPair`).

3.1 Obiettivo

L'obiettivo è quello di determinare, per ciascun intervallo temporale, il modello con il volume di vendite complessivo più alto. Gli intervalli sono stati definiti mediante un processo di **bucketing** sugli anni di produzione:

- `age<=2014`: veicoli prodotti fino al 2014;
- `2015_2018`: veicoli prodotti dal 2015 al 2018;
- `2019_2021`: veicoli prodotti dal 2019 al 2021;
- `>=2022`: veicoli prodotti dal 2022 in poi.

3.2 Workflow

Il driver `SparkDriver` implementa la seguente sequenza di trasformazioni:

1. **Caricamento dati**: i dati vengono letti da file CSV con l'operazione `textFile`. Le righe vuote o di intestazione vengono filtrate.

2. **Creazione coppie chiave/valore**: ciascuna riga viene trasformata in una coppia `((ageGroup, model), volume)`. Qui `ageGroup` è derivato dall'anno tramite la funzione di bucketing, mentre `volume` è estratto dal campo `Sales_Volume`.

3. **Aggregazione dei volumi**: le coppie vengono aggregate con `reduceByKey`, ottenendo il volume totale per ciascuna coppia `(ageGroup, model)`.

4. **Ristrutturazione**: i dati vengono rimappati nel formato `(ageGroup, (model, totalVolume))`.

5. **Selezione del massimo**: per ogni `ageGroup` viene selezionato il modello con il `totalVolume` più alto, utilizzando una riduzione che mantiene l'elemento massimo.

6. **Output**: i risultati vengono salvati in file di testo con formato

   `ageGroup  model  totalVolume`

CAPITOLO 4

ESECUZIONE, JAR E RISULTATI

4.1 Generazione dei JAR

Per poter eseguire i programmi sviluppati in ambiente distribuito è stato necessario creare degli archivi **JAR** contenenti il bytecode Java e le dipendenze essenziali.

- `BMWSales.jar`: include le classi sviluppate per l'esercizio MapReduce (`DriverBMWSales`, `Mapper1`, `Mapper2`, `Mapper3`, `Combiner1`, `Reducer1`, `Reducer2`, `Reducer3`).
  Viene eseguito con il comando:

  ```
  hadoop jar BMWSales.jar mapreduce.DriverBMWSales \
  <input> <out_1> <out_2> <out_3> [topK]
  ```

  dove `topK` indica il numero di modelli da estrarre per regione (opzionale, default = 5).

- `BMWSpark.jar`: contiene il driver `SparkDriver`.
  È stato eseguito con:

  ```
  spark-submit \
  --class spark.SparkDriver \
  --master local[*] \ # o con spark://spark-master:54310
  --deploy-mode client \
  --num-executors 3 \
  --executor-memory 1G \
  ```

```
--executor-cores 1 \
--driver-memory 1G \
BMWSpark.jar <input> <output>
```

4.2 Risultati MapReduce

L'esecuzione del job MapReduce ha prodotto tre directory di output corrispondenti ai tre job sequenziali:

- `bmw_out1`: aggregazione per `(regione, modello)`;
- `bmw_out2`: volumi complessivi per regione;
- `bmw_out3`: risultati finali con Top-K modelli per regione e metriche aggiuntive.

Esempio di output parziale Job 1 (aggregazione per regione e modello)

Regione	Modello	count | sumVol | sumPrice | highCount
Africa	3 Series	757 | 3,892,595 | 57,962,075 | 244
Africa	5 Series	789 | 4,020,702 | 60,270,409 | 240
Africa	7 Series	738 | 3,699,471 | 56,245,754 | 225
Africa	i3	783 | 3,967,283 | 58,288,045 | 222
Africa	i8	731 | 3,586,673 | 54,237,461 | 211
Africa	M3	694 | 3,448,709 | 52,295,352 | 209

Table 4.1: Aggregazione per (regione, modello) (estratto da `bmw_out1`).

Output parziale Job 2 (totali per regione)

Regione	Volume Totale
Africa	41,565,252
Asia	42,974,277
Europe	42,555,138

Table 4.2: Totali di vendita per regione (estratto da `bmw_out2`).

Output parziale Job 3 (Top-K modelli per regione)

Regione	Modello	Volume	Share %	Prezzo Medio	High %
Africa	5 Series	4,020,702	9.67	76,388.35	30.42
Africa	X5	3,972,541	9.55	73,532.00	30.04
Asia	X3	4,315,210	10.04	78,120.50	31.15
Asia	7 Series	4,201,870	9.77	80,540.20	29.87

Table 4.3: Top-K modelli per regione con metriche calcolate (estratto da `bmw_out3`).

4.3 Risultati Spark

Il job Spark ha prodotto l'output nella cartella `bmw_out_spark`. Ogni riga riporta il gruppo di età, il modello più venduto e il volume totale.

Fascia Età	Modello Top	Volume Totale
age $\leq$ 2014	X1	8,201,854
2015_2018	7 Series	6,465,540
2019_2021	M5	4,856,286
$\geq$ 2022	X6	5,094,533

Table 4.4: Output Spark: modello più venduto per gruppo di età (estratto da `bmw_out_spark`).

4.4 Confronto

Entrambi gli approcci hanno permesso di estrarre informazioni utili dal dataset, ma con differenze sostanziali:

- **MapReduce**: pipeline più articolata, che calcola molteplici metriche e classifica i modelli per regione;
- **Spark**: soluzione più concisa e performante, focalizzata sull'individuazione del modello top per fascia temporale.

In sintesi, Hadoop si presta ad analisi complesse a step multipli, mentre Spark risulta più immediato e veloce per operazioni iterative e di aggregazione.

LIST OF TABLES