Relazione Secondo Progetto - MapReduce, Hive e Spark

Giovanni Pio Grieco, Emilia Russo

June 13, 2025

1 Introduzione

Il progetto descritto in questa relazione è stato realizzato nell'ambito del corso di Big Data e ha come obiettivo l'analisi avanzata di un dataset di grandi dimensioni, sfruttando tecnologie distribuite per il processamento dati. Il dataset utilizzato è il seguente Dataset US Used Cars, disponibile su Kaggle, che contiene circa 3 milioni di record relativi ad automobili usate vendute negli Stati Uniti fino al 2020. Ogni record comprende 66 attributi, tra cui marca, modello, anno, prezzo, motorizzazione, chilometraggio e descrizione testuale dell'annuncio.

Gli obiettivi principali del progetto sono stati:

- la preparazione e pulizia del dataset, con l'obiettivo di ridurre il rumore o inconsistenti e selezionare i dati rilevanti per l'analisi;
- la progettazione e implementazione di due job di analisi dei dati, utilizzando tre tecnologie di big data processing viste a lezione: MapReduce, Hive e Spark Core;
- la misurazione delle performance dei job su dataset di dimensioni crescenti, eseguiti sia in locale sia su cluster.

L'analisi si concentra sui pattern relativi a marche, modelli e prezzi delle automobili, nonché sulle caratteristiche testuali delle descrizioni e sulle performance di vendita nel tempo. Le tecnologie sono state scelte per evidenziare differenze tra modelli di programmazione (imperativo vs dichiarativo vs funzionale), facilità di sviluppo, ottimizzazione e gestione delle risorse computazionali.

2 Ambiente di Esecuzione e Prestazioni Hardware

I test sono stati eseguiti sia su due postazioni di lavoro locali con configurazioni hardware differenti, sia su un cluster Elastic Map Reduce di AWS, al fine di valutare le prestazioni dei job implementati in ambienti con risorse computazionali variegate.

- **Laptop:** Lenovo IdeaPad 3 15IML05
  - Memoria RAM: 8,0 GB
  - Processore: Intel Core™ i5-10210U
  - Storage: Lenovo Micron 2210 NVMe SSD 512 GB
    - Sequential Write: 1.070 MB/s
    - Sequential Read: 2.200 MB/s
- **Desktop:** Workstation desktop
  - Memoria RAM: 32,0 GB
  - Processore: Intel Core™ i5-13500
  - Storage: Samsung SSD 870 QVO 1 TB
    - Sequential Write: 530 MB/s
    - Sequential Read: 560 MB/s

- **Cluster EMR:**
  - 4x instanze EC2 m5.xlarge
    - 4x vCPU
    - Memoria RAM: 16,0 GB
    - Storage: EBS 50,0 GB
      - IOPS: 5000
      - Throughput: 125,0 MiB/s

Cluster EMR su AWS Il cluster utilizzato per i test è composto da un nodo master (Namenode) e tre nodi worker (Datanode), dispiegati su 4 istanze EC2 m5.xlarge.

3 Preparazione dei dati

L'attività di preparazione del dataset è stata fondamentale per garantire l'efficacia e l'efficienza delle analisi successive. Considerata la presenza di oltre 3 milioni di record e 66 colonne, si è deciso di procedere con un processo di **pulizia, normalizzazione e partizionamento** del dataset originale.

Questa operazione ha permesso di ridurre sensibilmente le dimensioni del dataset, che sono passate da 9,29 GB a 6,96 GB, accelerando tutte le successive fasi di elaborazione.

3.1 Pulizia e Selezione delle Colonne

In una prima fase è stata effettuata la rimozione dei campi non significativi o ridondanti rispetto agli obiettivi del progetto. A tal fine, è stato fornito un elenco degli indici delle colonne da mantenere, permettendo così di eseguire una scansione completa del file CSV, estraendo esclusivamente le colonne di interesse e salvando il risultato in un nuovo file ridotto.

3.2 Normalizzazione e Pulizia Testuale

Successivamente, sono state eseguite operazioni di pulizia avanzata, tra cui:

- rimozione di caratteri speciali presenti nelle descrizioni dei veicoli;
- eliminazione di tag non informativi, come ad esempio [!@@Additional Info@@!];
- pulizia della colonna contenente le descrizioni, che ha previsto la rimozione della punteggiatura, l'eliminazione delle stopwords in lingua inglese e la conversione del testo in minuscolo.

3.3 Campionamento per test su scala

Infine, per valutare l'efficienza e la scalabilità dei job implementati, il dataset è stato suddiviso in più versioni di dimensioni crescenti: **1%, 5%, 20%, 100% e una versione duplicata al 200%**. La procedura ha previsto una lettura iniziale del file per determinarne il numero totale di righe, seguita dalla generazione dei sottoinsiemi corrispondenti, ciascuno salvato in un file separato. Tutte le partizioni sono state impiegate nei test di esecuzione, sia in ambiente locale che sul cluster cloud, al fine di osservare il comportamento delle soluzioni sviluppate al variare del volume di dati.

Va precisato che le tabelle riportate in seguito dei primi 10 risultati delle diverse esecuzioni si riferiscono specificamente al campionamento del 1%.

4 Job1: Statistiche per Marca e Modello

Il primo job ha l'obiettivo di generare le statistiche di ciascuna marca di automobile (make_name) presente nel dataset indicando, per ogni marca: (a) il nome della marca e (b) una lista di modelli (model_name) per quella marca indicando, per ciascun modello: (i) il numero di auto presenti nel dataset, (ii) il prezzo (price) minimo, massimo e medio di auto di quel modello nel dataset e (iv) l'elenco degli anni in cui il modello è presente nel dataset.

Nella Tabella 1 sono riportate le prime 10 ennuple ottenute dal Job1, che risultano identiche per le tre implementazioni: MapReduce, Hive e Spark Core.

Marca	Modello	Count	Prezzo Medio	Prezzo Min	Prezzo Max	Anni Presenti
acura	ilx	36	19424.61	12900.00	31205.00	2016, 2017, 2018, 2019, 2020, 2015
acura	mdx	107	27279.71	4495.00	59275.00	2005–2020
acura	mdx hybrid sport	2	46385.00	31995.00	60775.00	2018, 2020
acura	rdx	71	24106.49	7840.00	47195.00	2008–2015, 2016–2021
acura	rl	5	7818.60	5990.00	9458.00	1997, 2000, 2007, 2009, 2011
acura	rlx	5	36410.40	22999.00	52342.00	2015, 2016, 2020
acura	rlx hybrid sport	4	53592.50	38999.00	58457.00	2018, 2020
acura	rsx	3	3832.33	2499.00	4999.00	2002, 2003
acura	tl	15	8155.87	1899.00	14900.00	2003–2005, 2008–2010, 2012–2014
acura	tlx	96	23561.24	14200.00	44195.00	2015–2020

Table 1: Risultati del Job1 eseguito con MapReduce, Hive e Spark Core: statistiche per marca e modello con conteggio, prezzi e anni di presenza nel dataset.

4.1 MapReduce

L'elaborazione è stata realizzata tramite un job MapReduce suddiviso in due fasi:

- **Mapper**: legge ogni riga del dataset, estrae marca, modello, prezzo e anno, e genera come chiave la combinazione *(marca, modello)*, emettendo come valore prezzo e anno.
- **Reducer**: riceve in input i dati raggruppati per chiave *(marca e modello)*, calcola il numero totale di auto per quel modello, i prezzi minimo, massimo e medio, e raccoglie gli anni unici di presenza nel dataset.

I risultati di questa elaborazione sono riportati nella Tabella 2, la quale presenta il confronto delle prestazioni tra i due PC e il Cluster EMR, al variare della dimensione del dataset. Di seguito sono riportati gli pseudocodici relativi al mapper e al reducer utilizzati per questo job.

Algorithm 1 Pseudocodice Mapper per Job1

```
Input: Riga del dataset CSV
skip_header ← True
for ogni riga nel dataset do
    if skip_header è True then
        skip_header ← False
        continua alla prossima riga
    end if
    dividi riga in colonne usando la virgola come separatore
    estrai:
    make ← colonna indice 5
    model ← colonna indice 6
    price ← colonna indice 7
    year ← colonna indice 8
    stampa la stringa: ((make,model), price year)
end for
```

Dataset	Laptop	Desktop	Cluster EMR
1%	3.14 s	2.33 s	35.75 s
5%	5.05 s	3.35 s	30.05 s
20%	12.12 s	5.31 s	33.00 s
100%	40.55 s	19.37 s	106.61 s
200%	75.58 s	35.41 s	187.49 s

Table 2: Tempi di esecuzione del Job1 di MapReduce in locale su due PC, con differenti prestazioni hardware, e sul Cluster EMR

Algorithm 2 Pseudocodice Reducer per Job1

Input: Linee ordinate per chiave `(make,model)`
inizializza dizionario `stats` vuoto
for ogni linea in input **do**
    dividi la linea in chiave, price, year
    converti `price` a float, `year` a intero
    if `price` < 0 **then**
        ignora questa linea (errore prezzo negativo)
        continua
    end if
    if chiave non è in `stats` **then**
        crea nuovo record di statistiche per la chiave:
        conteggio $\leftarrow 0$
        sum prezzi $\leftarrow 0$
        prezzo min $\leftarrow +\infty$
        prezzo max $\leftarrow$ -∞
        insieme anni $\leftarrow$ vuoto
    end if
    aggiorna le statistiche:
      conteggio $\leftarrow$ conteggio + 1
      sum prezzi $\leftarrow$ sum prezzi + price
      prezzo min $\leftarrow$ min(prezzo min, price)
      prezzo max $\leftarrow$ max(prezzo max, price)
      aggiungi `year` all'insieme anni
end for
for ogni chiave in `stats` **do**
    calcola prezzo avg = sum prezzi / conteggio
    stampa `make, model, conteggio, prezzo avg, prezzo min, prezzo max, anni`
end for

4.2 Hive

L'elaborazione è stata realizzata utilizzando Hive, attraverso la definizione di una tabella e l'esecuzione di query SQL per l'analisi dei dati. Le fasi principali sono:

- **Creazione tabella**: viene definita la tabella `used_cars` con i campi rilevanti come marca (`make_name`), modello (`model_name`), prezzo (`price`) e anno (`year`), specificando il formato di caricamento e ignorando la prima riga di intestazione.
- **Caricamento dati**: i dati sono caricati all'interno della tabella da un percorso specificato.
- **Query di aggregazione**: viene eseguita una query che raggruppa i dati per marca e modello, calcolando il numero di occorrenze, il prezzo medio, minimo e massimo, e l'elenco degli anni in cui il modello è presente nel dataset.

I risultati di questa elaborazione sono riportati nella Tabella 3, la quale presenta confronto delle prestazioni tra i due PC e il Cluster EMR, al variare della dimensione del dataset.

Di seguito è riportato lo pseudocodice che sintetizza le operazioni eseguite.

Algorithm 3 Pseudocodice Hive per Job1

```
Step 1: Creare la tabella used_cars con i campi necessari e specificare il formato CSV con intestazione da saltare
CREATE TABLE used_cars (make_name, model_name, price, year)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE
TBLPROPERTIES ("skip.header.line.count"="1");
Step 2: Caricare i dati nella tabella da percorso specificato
LOAD DATA INPATH '<percorso_input>' OVERWRITE INTO TABLE used_cars;
Step 3: Eseguire query di aggregazione per calcolare statistiche per marca e modello
SELECT make_name AS make, model_name AS model, COUNT(*), AVG(price), MIN(price), MAX(price)
FROM used_cars
GROUP BY make_name, model_name
ORDER BY make_name, model_name;
```

Dataset	Laptop	Desktop	Cluster EMR
1%	13.71 s	8.97 s	28.30 s
5%	17.29 s	9.31 s	29.93 s
20%	22.58 s	11.38 s	32.59 s
100%	46.79 s	21.37 s	36.58 s
200%	81.77 s	36.72 s	42.34 s

Table 3: Tempi di esecuzione del Job1 di Hive in locale su due PC, con differenti prestazioni hardware, e sul Cluster EMR

4.3 SparkCore

L'elaborazione è stata realizzata utilizzando Spark Core, attraverso uno script PySpark che esegue il calcolo distribuito su un RDD. Le fasi principali sono:

- **Parsing dati**: ogni riga del dataset viene analizzata per estrarre marca, modello, prezzo e anno, restituendo una coppia chiave-valore con chiave il modello e valore una tupla contenente le informazioni rilevanti.
- **Aggregazione**: i dati vengono raggruppati per modello con l'operazione `reduceByKey`, che combina i valori calcolando il conteggio totale, il prezzo minimo, massimo, la somma dei prezzi e un insieme degli anni unici.

- **Formattazione output**: i dati aggregati vengono trasformati in una stringa formattata che riporta marca, modello, numero di occorrenze, prezzi minimo, massimo e medio, e l'insieme degli anni di presenza.
- **Salvataggio**: i risultati vengono salvati su file di output specificato.

I risultati di questa elaborazione sono riportati nella Tabella 4, la quale presenta il confronto delle prestazioni tra i due PC e il Cluster EMR, al variare della dimensione del dataset.

Di seguito è riportato lo pseudocodice che sintetizza le operazioni eseguite.

Algorithm 4 Pseudocodice Spark Core per Job1

```
Input: file CSV con dati delle auto usate
Output: file con risultati aggregati per modello
function parseLine(line)
    cols ← split(line, ",")
    if cols non sufficienti then
        return None
    end if
    make ← cols[5]
    model ← cols[6]
    price ← float(cols[7])
    year ← int(cols[8])
    return (model, (make, 1, price, price, price, {year}))
end function
function reducer(a, b)
    (make_a, count_a, min_price_a, max_price_a, sum_price_a, years_a) ← a
    (make_b, count_b, min_price_b, max_price_b, sum_price_b, years_b) ← b
    make ← make_a se make_a non vuoto, altrimenti make_b
    count ← count_a + count_b
    min_price ← minimo tra min_price_a e min_price_b
    max_price ← massimo tra max_price_a e max_price_b
    sum_price ← sum_price_a + sum_price_b
    years ← unione di years_a e years_b
    return (make, count, min_price, max_price, sum_price, years)
end function
function formatOutput(reducedData)
    (model, (make, count, min_price, max_price, sum_price, years)) ← reducedData
    avg_price ← sum_price / count se count > 0, altrimenti 0
    return stringa con make, model, count, min_price, max_price, avg_price, years
end function

Main:
inizializza SparkSession
dataset ← carica file di input come RDD di stringhe
parsed ← dataset.map(parseLine).filter(non None)
aggregated ← parsed.reduceByKey(reducer)
formatted ← aggregated.map(formatOutput)
salva formatted su file di output
```

5 Job2

Il secondo job ha come obiettivo generare un report contenente, per ciascuna città (city) e per ciascun anno (year): il numero di modelli di auto in vendita quell'anno appartenenti a tre fasce di prezzo (alto: sopra i 50K, medio: tra 20K e 50K, basso: inferiore a 20K) indicando, per ciascuna fascia, oltre al numero di auto in quella fascia, la media dei giorni di presenza delle auto sul mercato (daysonmarket) e le tre parole più frequenti che appaiono nella descrizione delle auto (description).

Dataset	Laptop	Desktop	Cluster EMR
1%	6.60 s	4.49 s	27.87 s
5%	6.62 s	5.67 s	27.23 s
20%	14.15 s	7.94 s	31.96 s
100%	32.44 s	18.65 s	38.41 s
200%	60.28 s	25.10 s	48.71 s

Table 4: Tempi di esecuzione del Job1 di Spark Core in locale su due PC, con differenti prestazioni hardware, e sul Cluster EMR

5.1 MapReduce

L'elaborazione è stata realizzata tramite un job `MapReduce` suddiviso in due fasi:

- **Mapper**: legge ogni riga del dataset, estrae *città*, *anno*, *prezzo*, *giorni sul mercato* e *descrizione*, e genera come chiave la combinazione `(città, anno)`, emettendo come valore il prezzo, i giorni sul mercato e la descrizione.
- **Reducer**: riceve in input i dati raggruppati per chiave `(città, anno)` e li suddivide in tre fasce di prezzo:
  - **low**: prezzo $< 20\,000$
  - **middle**: $20\,000 \leq$ prezzo $< 50\,000$
  - **high**: prezzo $\geq 50\,000$

  Per ciascuna fascia di prezzo, calcola:
  - il numero totale di annunci
  - la media dei giorni sul mercato
  - le tre parole più frequenti nelle descrizioni degli annunci

  Infine, il Reducer emette una riga per ogni combinazione `(città, anno, fascia_di_prezzo)`, con il numero di annunci, la media dei giorni sul mercato e le parole chiave più ricorrenti.

L'utilizzo di MapReduce in questo contesto ha reso possibile l'elaborazione di dataset di grandi dimensioni in modo distribuito, anche se l'aumento della complessità logica e la gestione esplicita dello stato e dei conteggi ha comportato un incremento significativo dei tempi di elaborazione rispetto ad altre tecnologie più ottimizzate per operazioni miste su dati strutturati e testuali.

I risultati di questa elaborazione sono riportati nella Tabella 5 e nella Tabella 6, le quali rispettivamente rappresentano i primi 10 risultati dell'elaborazione del Job2 con l'approccio MapReduce e il confronto delle prestazioni tra i due PC e il Cluster EMR, al variare della dimensione del dataset.

Di seguito sono riportati gli pseudocodici relativi al mapper e al reducer utilizzati per questo job.

Algorithm 5 Pseudocodice Mapper per Job2

```
Input: Linee CSV del dataset (con intestazione)
salta la prima riga (header)
for ogni linea nel file do
    rimuovi spazi iniziali/finali e dividi per virgola
    city ← colonna[0]
    daysonmarket ← colonna[1]
    description ← colonna[2]
    price ← colonna[7]
    year ← colonna[8]
    stampa city, year, price, daysonmarket, description separati da tab
end for
```

Algorithm 6 Pseudocodice Reducer per Job2

Input: Linee raggruppate per chiave `(city, year)`
Inizializza 3 insiemi di dizionari: `low`, `middle`, `high` per ogni metrica
for ogni linea in input **do**
    estrai `city, year, price, daysonmarket, description`
    converti `price` a float, `daysonmarket` a intero
    if `price` < 20000 **then**
        aggiorna contatore, somma giorni e parole per fascia `low`
    else if $20000 \leq$ `price` < 50000 **then**
        aggiorna contatore, somma giorni e parole per fascia `middle`
    else
        aggiorna contatore, somma giorni e parole per fascia `high`
    end if
end for
for ogni chiave `(city, year)` **do**
    for ogni fascia di prezzo: `low, middle, high` **do**
        if presente almeno un elemento **then**
            calcola media giorni = somma / conteggio
            estrai top 3 parole più frequenti
            stampa: `city, year, fascia, conteggio, media, top words`
        end if
    end for
end for

City	Year	Price Range	Num Models	Avg Days on Market	Top 3 Words
Hasbrouck Heights	2012	Low	8	37.12	rear, power, front
Hasbrouck Heights	2012	Middle	2	42.50	power, control, steering
Hasbrouck Heights	2012	High	0	0.00	N/A
Springfield	2019	Low	8	60.25	front, rear, power
Springfield	2019	Middle	1	74.00	front, power, rear
Springfield	2019	High	0	0.00	N/A
East Hartford	2015	Low	8	17.12	rear, power, front
East Hartford	2015	Middle	1	13.00	rear, power, steering
East Hartford	2015	High	0	0.00	N/A
Totowa	2005	Low	3	23.33	fees, power, vehicle

Table 5: Primi 10 risultati della computazione del Job2 con l'utilizzo di MapReduce

Dataset	Laptop	Desktop	Cluster EMR
1%	8.02 s	4.34 s	30.84 s
5%	26.10 s	13.36 s	35.98 s
20%	100.57 s	46.38 s	56.89 s
100%	516.97 s	243.68 s	228.63 s
200%	1009.01 s	506.67 s	425.87 s

Table 6: Tempi di esecuzione del Job2 di MapReduce in locale su due PC, con differenti prestazioni hardware, e sul Cluster EMR

5.2 Hive

L'elaborazione è stata realizzata utilizzando Hive, attraverso la definizione di tabelle e l'esecuzione di query SQL per l'analisi dei dati. Le principali fasi del processo sono state:

- **Creazione della tabella** `cars` per importare i dati grezzi dal file CSV, specificando la struttura e il formato dei dati, e gestendo l'intestazione del file.
- **Caricamento dei dati** dal percorso di input nella tabella `cars`.
- **Aggiunta di una nuova colonna**, deniminata `fascia_prezzo`, che classifica le auto in tre

categorie di prezzo: alta, media e bassa.

- **Calcolo di statistiche aggregate**, come il numero di veicoli e la media dei giorni sul mercato, raggruppate per città, anno e fascia di prezzo.
- **Analisi della** `descrizione` per estrarre e contare la frequenza delle parole utilizzate nelle descrizioni delle auto.
- **Identificazione delle tre parole più ricorrenti** per ciascun gruppo di città, anno e fascia di prezzo.
- **Creazione di una tabella dei risultati** che combina le statistiche quantitative con le parole più frequenti, fornendo un report completo sull'andamento del mercato auto usate.

I risultati di questa elaborazione sono riportati nella Tabella 7 e nella Tabella 8, le quali rispettivamente rappresentano i primi 10 risultati dell'elaborazione del Job2 con l'approccio Hive e il confronto delle prestazioni tra i due PC e il Cluster EMR, al variare della dimensione del dataset.

Di seguito è riportato lo pseudocodice relativo all'utilizzo di Hive per questo job.

Algorithm 7 Pseudocodice Hive Job2

```
Step 1: Creare la tabella cars con i campi necessari e specificare formato CSV saltando l'intestazione
DROP TABLE IF EXISTS cars;
CREATE TABLE cars (city STRING, daysonmarket INT, description STRING,
  engine_displacement FLOAT, horsepower FLOAT, make_name STRING,
  model_name STRING, price FLOAT, year INT)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE
TBLPROPERTIES ("skip.header.line.count"="1");
Step 2: Caricare i dati dal percorso di input nella tabella cars
LOAD DATA INPATH '<input_path>' OVERWRITE INTO TABLE cars;
Step 3: Creare tabella cars_with_fascia aggiungendo colonna fascia prezzo in base al price
  fascia_prezzo =
    "alta" se price > 50000
    "media" se 20000 <= price <= 50000
    "bassa" altrimenti
Step 4: Calcolare statistiche per ogni gruppo (city, year, fascia_prezzo)
  num_cars = COUNT(*)
  avg_daysonmarket = AVG(daysonmarket)
Step 5: Calcolare frequenza parole dalla colonna description
  Estrarre parole da description con SPLIT e EXPLODE
  Raggruppare per city, year, fascia_prezzo e parola, contare occorrenze
Step 6: Selezionare le prime 3 parole più frequenti per gruppo
Step 7: Creare tabella finale report_cars unendo statistiche e parole top 3
Step 8: Visualizzare i risultati
  SELECT * FROM report_cars;
```

5.3 SparkCore

Il processo di elaborazione si articola in quattro fasi principali:

- **Mapper**: seleziona i campi utili, determina la fascia di prezzo dell'auto e analizza la descrizione testuale contando le parole significative.
- **Reducer**: combina i dati aggregando il numero di auto, i giorni complessivi sul mercato e le frequenze delle parole.
- **Formatter**: calcola la media dei giorni sul mercato e identifica le tre parole più ricorrenti nella descrizione.

City	Year	Price Range	Num Models	Avg Days on Market	Top 3 Words
acton	2010	bassa	2	469.5	power, air, bag
acton	2015	bassa	2	12.5	power, rear, bag
acton	2016	bassa	2	25.5	bag, air, rear
acton	2016	media	1	4.0	air, bag, power
acton	2017	bassa	6	51.67	2017, bag, air
acton	2017	media	9	46.11	2017, rear, system
acton	2018	bassa	1	6.0	rear, power, control
acton	2018	media	8	47.75	rear, bag, air
acton	2019	media	12	20.42	rear, power, mirror
acton	2020	bassa	5	38.4	rear, system, eyesight

Table 7: Primi 10 risultati dell'esecuzione del Job2 utilizzando Hive

Dataset	Laptop	Desktop	Cluster EMR
1%	10.18 s	7.15 s	75.83 s
5%	11.51 s	7.39 s	16.51 s
20%	12.71 s	7.03 s	17.13 s
100%	12.68 s	6.94 s	15.59 s
200%	12.67 s	7.12 s	17.76 s

Table 8: Tempi di esecuzione del Job2 di Hive in locale su due PC, con differenti prestazioni hardware, e sul Cluster EMR

I risultati di questa elaborazione sono riportati nella Tabella 9 e nella Tabella 10, le quali rispettivamente rappresentano i primi 10 risultati dell'elaborazione del Job2 con l'approccio Spark Core e il confronto delle prestazioni tra i due PC e il Cluster EMR, al variare della dimensione del dataset. Di seguito è riportato lo pseudocodice relativo all'utilizzo di Spark Core per questo job.

Algorithm 8 Pseudocodice Spark Core Job2

Input: file CSV con dati auto usate
Output: aggregazioni per città, anno, fascia di prezzo
function MAPPER(line)
    Estrai `city`, `year`, `price`, `days`, `description`
    price_code ← codifica fascia prezzo
    word_count ← conteggio parole in `description`
    return ((city, year, price_code), (1, days, word_count, 1))
end function
function REDUCER(a, b)
    Somma conteggi e giorni
    Unisci dizionari delle parole
    return (tot_count, tot_days, merged_words, tot_items)
end function
function FORMATOUTPUT((key, values))
    Calcola media giorni
    Estrai top-3 parole più frequenti
    return stringa formattata
end function

Main:
Inizializza SparkSession
dataset ← carica input CSV
parsed ← dataset.map(mapper).filter(non None)
reduced ← parsed.reduceByKey(reducer)
output ← reduced.map(formatOutput)
Salva output su file

City	Year	Price Range	Num Models	Avg Days on Market	Top 3 Words
Guaynabo	2016	middle	1	1233.00	
San Juan	2020	high	67	167.94	pack(233), rear(138), wheel(121)
Bay Shore	2018	middle	38	31.45	front(437), rear(409), power(305)
Bronx	2017	low	79	48.11	power(727), rear(657), front(602)
Bay Shore	2017	middle	71	34.25	rear(933), front(920), power(749)
Bronx	2018	low	14	39.36	vehicle(110), power(95), rear(90)
Bronx	2020	low	3	42.00	vehicle(26), power(19), air(17)
Bay Shore	2019	middle	31	49.45	front(410), rear(349), power(296)
Bronx	2013	low	6	108.00	front(73), rear(65), power(50)
Bay Shore	2016	low	8	31.88	front(101), rear(86), power(58)

Table 9: Primi 10 risultati dell'esecuzione del Job2 utilizzando Spark Core

Dataset	Laptop	Desktop	Cluster EMR
1%	8.89 s	4.61 s	26.09 s
5%	18.06 s	7.06 s	38.12 s
20%	49.82 s	10.25 s	67.36s
100%	204.76 s	44.84 s	127.29 s
200%	417.10 s	89.97 s	225.02 s

Table 10: Tempi di esecuzione del Job2 di SparkCore in locale su due PC, con differenti prestazioni hardware, e sul Cluster EMR

6 Valutazione complessiva delle esecuzioni

L'analisi delle esecuzioni condotte ha permesso di valutare in modo approfondito l'impatto congiunto di cinque dimensioni fondamentali: **tempo di esecuzione, dimensione del dataset, ambiente computazionale, tecnologia utilizzata e tipologia di task (Job1 e Job2)**. L'interazione tra questi fattori ha evidenziato pattern prestazionali ricorrenti, utili a orientare la scelta di strumenti e configurazioni in contesti reali.

6.1 Tempo e Dimensione del Dataset

Il tempo di esecuzione cresce in maniera lineare all'aumentare della dimensione del dataset, ma con inclinazioni diverse a seconda della tecnologia impiegata.

6.2 Differenze tra i Task

Il **Job1**, focalizzato su aggregazioni per marca e modello, ha comportato un carico computazionale contenuto e una pressione ridotta sul sistema di I/O. Le operazioni erano principalmente strutturate e numeriche, il che ha reso i tempi di esecuzione relativamente stabili e coerenti tra le tecnologie. I grafici 1, 2 e 3 mostrano l'evoluzione dei tempi di elaborazione del Job1 al crescere della dimensione del dataset rispettivamente per MapReduce, Hive e Spark Core. Da tali grafici, in particolar modo quelli relativi a Spark e Hive, si può evincere il vantaggio di eseguire computazioni su cluster EMR.

Infatti, è possibile notare come in entrambi i casi, la curva relativa al cluster EMR, seppur partendo da una quota più elevata, al crescere del dataset, finisca per intersecare le curve delle esecuzioni in locale, mostrando quindi una maggiore scalabilità.

Il **Job2**, invece, ha introdotto una maggiore complessità computazionale. Le operazioni richieste comprendevano elaborazioni testuali, classificazione in fasce di prezzo e aggregazioni più articolate. Questa maggiore complessità ha amplificato le differenze tra le tecnologie, come illustrato nei grafici 4, 5 e 6.

Questo ha reso più evidenti le differenze tra le tecnologie:

- **MapReduce** ha subito rallentamenti significativi, evidenziando il limite di efficienza nel gestire elaborazioni miste (numeriche e testuali).
- **Hive** ha avuto un comportamento inaspettato in quanto risulta mantenere un andamento costante

all'aumentare del dataset, dovuto probabilmente a operazioni di caching effettuate dalla tecnologia.

- **Spark Core** ha mostrato generalmente performance migliori rispetto a MapReduce. L'esecuzione su Cluster EMR performa meglio di quella del 'Laptop' al crescere del dataset. Diverso invece è il confronto tra Cluster EMR e 'Desktop', in quanto non è presente un'intersezione tra i due tracciati. Bisogna ricordare che il range in cui sono state effettuate le misurazioni è stato limitato al 200% e quindi non si può assumere che prima o poi anche questi ultimi due non si incontrino.

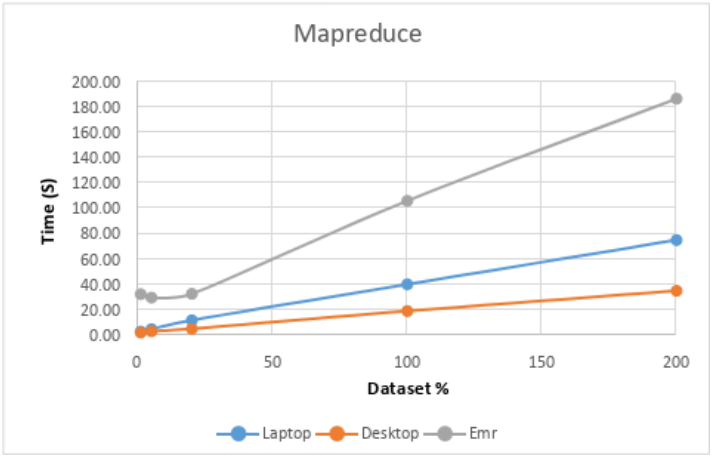

Figure 1: Tempi di esecuzione del Job1 con MapReduce al variare della dimensione del dataset

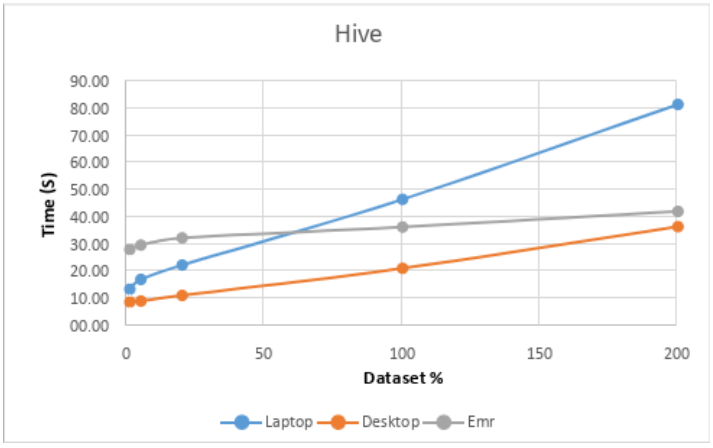

Figure 2: Tempi di esecuzione del Job1 con Hive al variare della dimensione del dataset

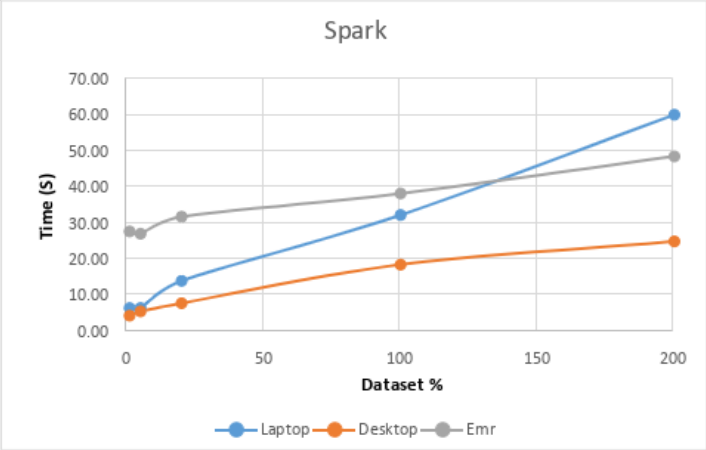

Figure 3: Tempi di esecuzione del Job1 con Spark Core al variare della dimensione del dataset

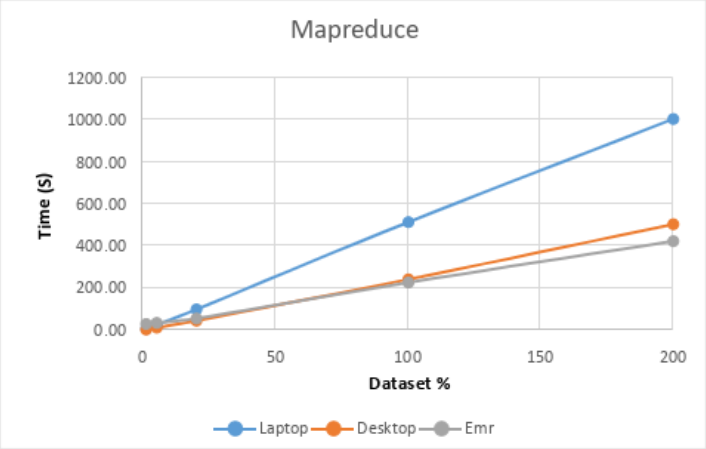

Figure 4: Tempi di esecuzione del Job2 con MapReduce al variare della dimensione del dataset

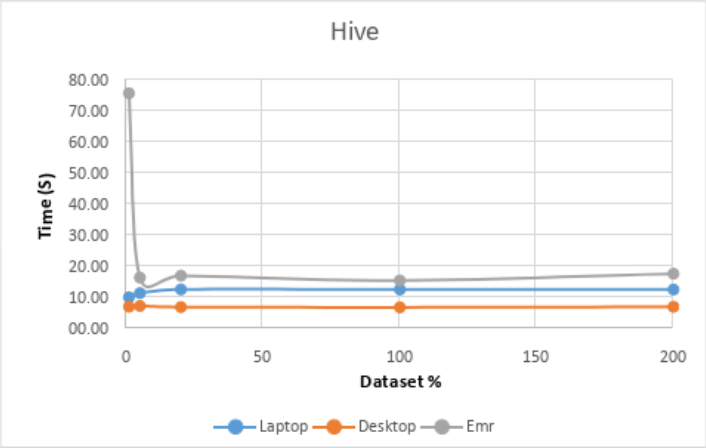

Figure 5: Tempi di esecuzione del Job2 con Hive al variare della dimensione del dataset

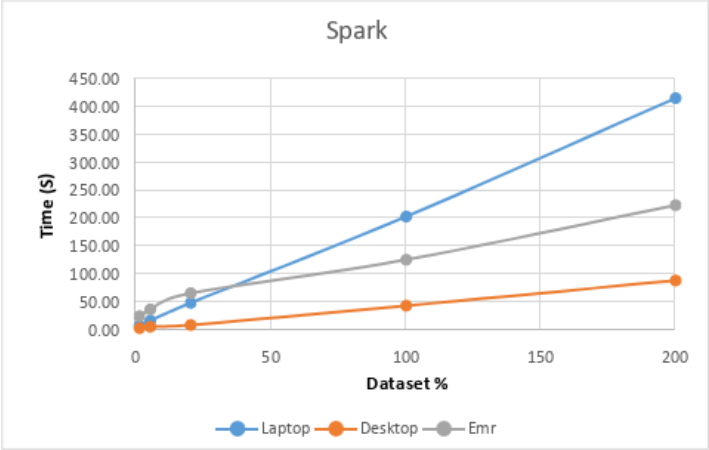

Figure 6: Tempi di esecuzione del Job2 con Spark Core al variare della dimensione del dataset

6.3 Ambiente di esecuzione e Scalabilità

Le tecnologie valutate si comportano in modo differente a seconda dell'ambiente in cui sono eseguite:

- In **locale**, il calcolatore 'Desktop' (più performante) ha consentito di ottenere risultati notevolmente migliori rispetto al calcolatore 'Laptop', in particolare per Spark Core, dove la maggiore RAM ha avuto un impatto decisivo.
- Il **cluster** EMR, pur introducendo un certo overhead iniziale su dataset piccoli, ha garantito maggiore scalabilità, ovvero tempi che crescono ad un tasso minore rispetto al tasso di crescita dei dati coinvolti. Per Spark e Hive in particolare, il cluster ha offerto prestazioni migliori rispetto all'ambiente locale.

Tale capacità del cluster EMR diventa ancora più evidente utilizzando la misura dell'elasticità.

Sia d la dimensione del dataset, i la tecnologia, j l'ambiente e k il task. Sia $L_{i,j,k}(d)$ la funzione che, data la dimensione del dataset d, la tecnologia i, l'ambiente j e il Job k, restituisce il tempo di esecuzione T.

L'elasticità è definita come:

$$E = \frac{L_{i,j,k}(d_1) - L_{i,j,k}(d_0)}{d_1 - d_0} \cdot \frac{d_1}{L_{i,j,k}(d_1)}$$

Se l'elasticità è pari a 1, ciò implica che il tempo di esecuzione del job è sensibile al cambiamento di dimensione del dataset. Se otteniamo un valore minore di 1, significa che abbiamo un sistema che è anaelastico nel tempo rispetto alla dimensione del dataset. Minore è l'elasticità, meglio è. Questa caratteristica è positiva nel contesto *big data* e viene evidenziata nei risultati riportati nella Tabella 11.

7 Conclusioni

L'ambiente distribuito del Cluster EMR ha dimostrato di essere una buona scelta per elaborazioni eseguite su grandi volumi di dati.

Per quanto riguarda le tecnologie, Spark e Hive hanno performato meglio rispetto a MapReduce, su tutte le piattaforme e rispetto ai corrispettivi job.

References

Per accedere al repository del progetto seguire il seguente link Progetto di Big Data.

Job	Tecnologia	Dataset %	Elasticità Laptop	Elasticità Desktop	Elasticità EMR
Job 1	MapReduce	5%-1%	0.47	0.38	**-0.11**
		20%-5%	0.78	0.49	**0.12**
		100%-20%	0.88	0.91	**0.86**
		200%-100%	0.93	0.91	**0.86**
	Hive	5%-1%	0.26	**0.05**	0.07
		20%-5%	0.31	0.24	**0.11**
		100%-20%	0.65	0.58	**0.14**
		200%-100%	0.86	0.84	**0.27**
	Spark-Core	5%-1%	0.00	0.26	**-0.03**
		20%-5%	0.71	0.38	**0.20**
		100%-20%	0.70	0.72	**0.21**
		200%-100%	0.92	0.51	**0.42**
Job 2	MapReduce	5%-1%	0.87	0.84	**0.18**
		20%-5%	0.99	0.95	**0.49**
		100%-20%	1.01	1.01	**0.94**
		200%-100%	0.98	1.04	**0.93**
	Hive	5%-1%	0.14	0.04	**-4.49**
		20%-5%	0.13	**-0.07**	0.05
		100%-20%	0.00	-0.02	**-0.12**
		200%-100%	**0.00**	0.05	0.24
	Spark-Core	5%-1%	0.63	0.43	**0.39**
		20%-5%	0.85	**0.41**	0.58
		100%-20%	0.95	0.96	**0.59**
		200%-100%	1.02	1.00	**0.87**

Table 11: Elasticità per i diversi ambienti e tecnologie nei Job 1 e Job 2