BigData

PRIMO PROGETTO
MAP REDUCE, SPARK, HIVE

—

Di Gianmaria Del Monte

INTRODUZIONE

Il primo homework del corso di Big Data consiste nel generare alcune utili statistiche dal dataset *Daily Historical Stock Prices*, contenente l'andamento giornaliero delle azioni sulla borsa di New York (*NYSE*) e sul *NASDAQ*, dal 1970 al 2018.

Struttura del dataset

Il dataset è formato da due file CSV. Ogni riga del primo, denominato **historical stock prices** (*hsp*) è formato dai seguenti campi:

- *ticker*: simbolo univoco dell'azione
- *open*: prezzo di apertura
- *close*: presso di chiusura
- *adj_close*: prezzo di chiusura "modificato"
- *lowThe*: prezzo minimo
- *highThe*: prezzo massimo
- *volume*: numero di transazioni
- *date*: data nel formato *aaaa-mm-gg*

Il secondo, denominato **historical stocks** (*hp*), ha i seguenti campi:

- *ticker*: simbolo dell'azione
- *exchange*: NYSE o NASDAQ
- *name*: nome dell'azienda
- *sector*: settore dell'azienda
- *industry*: industria di riferimento per l'azienda

Descrizione dei job

A partire dal dataset, si devono progettare e realizzare in *MapReduce*, *Spark* ed *Hive* i seguenti 3 job:

1. Generare le statistiche di ciascuna azione tra il 2008 e il 2018 indicando per ogni azione:
    a. il simbolo,
    b. la variazione della quotazione,
    c. il prezzo minimo,
    d. il prezzo massimo,
    e. volume medio nell'intervallo,

    ordinando l'elenco in ordine decrescente di variazione della quotazione.
2. Generare per ciascun settore il relativo trend nel periodo 2008-2018, ovvero un elenco contenente per ciascun anno nell'intervallo:

   a. il volume annuale medio delle azioni del settore
   b. la variazione annuale media delle aziende del settore
   c. la quotazione giornaliera media delle aziende del settore

3. Generare i gruppi di aziende le cui azioni hanno avuto lo stesso trend in termini di variazione annuale nell'ultimo triennio disponibile, indicando le aziende ed il trend comune.

PSEUDOCODICE

In questo paragrafo è riportata l'implementazione MapReduce e Spark in pseudocodice dei 3 job.

Map Reduce

Job1

Il primo job è formato a sua volta da 2 job MapReduce eseguiti uno dopo l'altro: il primo si occupa di calcolare le statistiche richieste, il secondo di ordinare in ordine decrescente di variazione della quotazione.

Il primo job MapReduce è composto da una funzione *map*, che si occupa di filtrare le azioni il cui anno è compreso nell'intervallo richiesto, selezionando inoltre solo i valori utili alla generazione delle statistiche richieste, ovvero il simbolo, il prezzo di chiusura (per il calcolo della variazione, del prezzo minimo e del prezzo massimo), il volume (per il calcolo del volume medio) e la data (usato in combinazione del prezzo di chiusura per il calcolo della variazione della quotazione).

La funzione *reduce*, prende in input coppie simbolo, lista di tuple (prezzo di chiusura, volume, data) e calcola per ogni lista il prezzo minimo, il prezzo massimo, il volume medio, il prezzo di chiusura all'inizio e alla fine del periodo per il calcolo della variazione.

```
Map(_, line):
      line_splitted = split(line, ",")
      select ticker, close_price, volume date from line_splitted

      emit(ticker, (close_price, volume, date)) if 2008 <= year <= 2018

Reduce(ticker, tuples):
```

```
      select min_price from tuples list
      select max_price from tuples list
      compute mean_volume # sum volumes in tuples / len(tuples)
      select price_at_min_date, i.e. the price at the min date in the
interval considered in tuples list
      select price_at_max_date, i.e. the price at the max date in the
interval considered in tuples list

      variation = (price_at_max_date - price_at_min_date) /
price_at_min_date * 100

      emit(ticker, (variation, min_price, max_price, mean_volume))
```

Il secondo job MapReduce consiste in una funzione *map* che sostanzialmente restituisce gli stessi valori di input, incapsulando la variazione in un oggetto, denominato *InverseDouble*, su cui è definito un comparatore *inverso* rispetto a quello dell'oggetto Double. Anche la funzione *filter* non fa altro che emettere le tuple che ricevere in input. Il sort è quindi lasciato al framework durante la fase di shuffle and sort.

```
Map(ticker, stat):
      select variation, min_price, max_price, mean_volume from stat
      emit((InverseDouble) variation, (ticker, variation, min_price,
max_price, mean_volume))

Reduce(variation, stats):
      for (ticker, variation, min_price, max_price, mean_volume) in stats:
      emit(ticker, (variation, min_price, max_price, mean_volume))
```

Job 2

Il secondo job è formato da 3 job MapReduce. Il primo si occupa di calcolare per ogni azione, anno per anno il volume annuale, ovvero la somma dei volumi dell'azione in un anno, la quotazione giornaliera, calcolato come media del prezzo di chiusura dell'azione in un anno, e la variazione della quotazione.

```
Map(_, line):
      line_splitted = split(line, ",")

      if 2008 <= year <= 2018:
```

```
      select ticker, volume, close_price, date

      emit((ticker, year), (close_price, volume, date))

Reduce(ticker_year, stats):
      compute annual_volume # sum volumes in stats
      select price_at_min_date, i.e. the price at the min date in the
interval considered in stats list
      select price_at_max_date, i.e. the price at the max date in the
interval considered in stats list
      compute sum_prices # sum prices in stats list

      variation = (price_at_max_date - price_at_min_date) /
price_at_min_date * 100

      emit(ticker_year, (variation, annual_volume, sum_prices, len(stats) as
count))
```

Il secondo job MapReduce si occupa del join tra il risultato del precedente job MapReduce e il file *hp*, ottenendo una lista di coppie settore => (simbolo, anno, variatione, volume annuale, somma prezzi, contatore).

Infine il terzo job si occupa di di generare per ogni anno e per ogni settore il volume medio, la variazione media e la quotazione giornaliera media.

```
Map(_, line):
      select ticker, year, volume, variation, sum_price, count from line

      emit((ticker, year), (variation, volume, sum_price, count))

Reduce(ticker_year, stats):
      for (variation, volume, sum_price, count) in stats:
      c++;
      sum_volume += volume
      sum_variation += variation
      sum_sum_price += sum_price
      counts += count

      mean_volume = sum_volume / c
      mean_variation = sum_variation / c
      mean_prices = sum_sum_price / counts

      emit(ticker_year, (mean_volume, mean_variation, mean_prices))
```

Job 3

Il terzo job è formato da 3 job MapReduce. Il primo effettua un join tra *hp* e il risultato del primo reduce del job 2, generando coppie azienda => (anno, variazione).

Il secondo calcola il trend nell'ultimo triennio disponibile per ogni azienda.

```
Map(_, line):
      select company, year, variation from line

      emit(company, (year, variation))

Reduce(company, tuples):
      trend = get_last_three_continuous_years(tuples)

      if (last_three_years)
          emit(company, trend)
```

Il terzo job MapReduce, sfruttando la fase di shuffle and sort, si occupa di unire le aziende che hanno lo stesso trend.

```
Map(company, trend):
      emit(trend, company)

Reduce(trend, companies):
      emit(companies, trend)
```

Spark

Di seguito sono riportate le implementazioni in pseudocodice di Spark.

Job 1

```
      hsp.map(line -> (ticker, (close_price, close_price, volume, 1, date,
close_price, date, close_price)))
      .filter(t -> 2008 <= date <= 2018)
      .reduceByKey(t1, t2 ->
```

```
            (min(t1[0], t2[0]), max(t1[2], t2[2]), t1[3] + t2[3], t1[4] +
t2[4], min(t1[5], t2[5]), price_min_date, max(t1[7], t2[7]), price_max_date)
      )
      .map(k, v -> ((v[7] - v[5])/v[5] * 100 as variation,
            (k, v[0], v[1], v[2], v[3], variation))
      )
      .sortByKey(false)
      .map(k, v -> v)
```

Job 2

```
      hs_mapped = hs.map(l -> ticker, sector)

      hsp.map(l -> (ticker, year), volume, close_price, close_price, date,
date, close_price, 1)
      .fiter(2008 <= year 2018)
      .reduceByKey(t1, t2 ->
            (t1[0] + t2[0], close_price_min_date, close_price_max_date,
min(t1[3], t2[3]),
            max(t1[4], t2[4], t1[5] + t2[5], t1[6] + t2[6])
            )
      )
      .map(t -> ticker, (year, volume, variation, mean_price))
      .join(hs_mapped)
      .reduceByKey(t1, t2 -> t1[0]+t2[0],
                  t1[1]+t2[1], t1[2]+t2[2], t1[3]+t2[3])
```

Job 3

```
      hsp.map(l -> (ticker, year), (close, date, close, date))
      .reduceByKey(t1, t2 -> (price_max_date, max(t1[1], t2[1]),
price_min_date, min(t1[3], t2[3])))
      .map((ticker, year), prices -> ticker, (year, variation))
      .join(hs)
      .map(k, v -> company, (year, variation))
      .groupByKey()
      .map(company, list[(year, variation)] -> company,
laste_three_years(list))
      .map(company, last_three_years -> last_three_years, company)
      .groupByKey()
```

RISULTATI

Di seguito sono riportate le prime 10 righe del risultato di ogni job.

Job 1

Ogni riga è formata dai seguenti campi: simbolo dell'azione, prezzo di chiusura minimo, prezzo di chiusura massimo, volume medio, variazione della quotazione.

```
EAF,0.00,22.37,1245139.04,267757.13
ORGS,0.00,19.80,8570.96,217415.93
PUB,0.01,135.00,34449.40,179900.00
RMP,0.03,78.55,120487.05,121081.60
CTZ,0.00,26.73,52050.27,120300.00
CCD,0.01,25.55,105416.89,111250.00
SAB,1.40,318800.00,1695378.02,110428.77
KE,0.01,22.20,73249.85,99400.00
LN,0.01,49.63,394344.93,96700.00
GHG,0.00,24.31,607659.29,64850.00
```

Job 2

Ogni riga è formata dai seguenti campi: settore, anno, volume annuale medio, variazione annuale media, quotazione giornaliera media.

```
BASIC INDUSTRIES,2008,560950485.80,-4.94,604.10
BASIC INDUSTRIES,2009,589384165.14,-1.71,224.26
BASIC INDUSTRIES,2010,476571424.23,-2.50,32.13
BASIC INDUSTRIES,2011,448450099.40,0.01,443.94
BASIC INDUSTRIES,2012,377483105.25,-2.16,124.53
BASIC INDUSTRIES,2013,367271657.58,-0.77,238.53
BASIC INDUSTRIES,2014,350472233.62,0.38,233.44
BASIC INDUSTRIES,2015,388588042.27,0.09,41.42
BASIC INDUSTRIES,2016,470968318.09,1.19,30.90
BASIC INDUSTRIES,2017,383340553.00,0.32,35.16
```

Job 3

Ogni riga è formata dai seguenti campi: lista delle aziende con stesso trend, trend.

```
{SPECTRUM BRANDS HOLDINGS, INC.,HERSHEY COMPANY
(THE)}:2018:-11,2017:9,2016:17
```

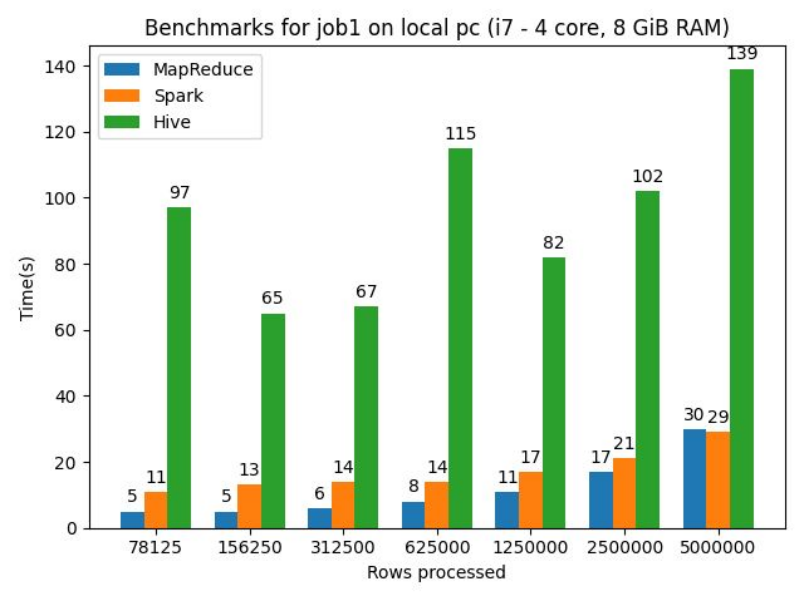
```
{SPDR DORSEY WRIGHT FIXED INCOME ALLOCATION ETF,INVESCO TRUST FOR INVESTMENT
GRADE MUNICIPALS}:2018:-7,2017:3,2016:-4
{SPECIAL OPPORTUNITIES FUND INC.,BLACKROCK MULTI-SECTOR INCOME
TRUST}:2018:-2,2017:8,2016:4
{SNAP-ON INCORPORATED,WALT DISNEY COMPANY (THE)}:2018:0,2017:1,2016:1
{BLACK KNIGHT, INC.,POWERSHARES S&P SMALLCAP CONSUMER DISCRETIONARY
PORTFOLIO}:2018:16,2017:16,2016:16
{ISHARES SHORT TREASURY BOND ETF,ISHARES 1-3 YEAR TREASURY BOND ETF,ISHARES
0-5 YEAR INVESTMENT GRADE CORPORATE BOND ETF,IVY NEXTSHARES,VANGUARD
SHORT-TERM TREASURY ETF,WISDOMTREE BARCLAYS INTEREST RATE HEDGED U.S.
AGGREGATE BOND F,POWERSHARES VARIABLE RATE INVESTMENT GRADE PORTFOLIO,FIRST
TRUST ENHANCED SHORT MATURITY ETF,ISHARES 1-3 YEAR CREDIT BOND
ETF}:2018:0,2017:0,2016:0
{PIMCO MUNICIPAL INCOME FUND II,MAINSTAY MACKAY DEFINEDTERM MUNICIPAL
OPPORTUNITIE}:2018:0,2017:7,2016:-3
{ENTERGY CORPORATION,OIL-DRI CORPORATION OF AMERICA}:2018:3,2017:11,2016:7
{PIMCO NEW YORK MUNICIPAL INCOME FUND II,NUVEEN QUALITY MUNICIPAL INCOME
FUND}:2018:-5,2017:0,2016:-3
{ISHARES CORE 1-5 YEAR USD BOND ETF,POWERSHARES LADDERRITE 0-5 YEAR
CORPORATE BOND PORTFOLIO,ISHARES 3-7 YEAR TREASURY BOND ETF,VANGUARD
SHORT-TERM CORPORATE BOND ETF,VANGUARD INTERMEDIATE-TERM TREASURY
ETF,VANGUARD MORTGAGE-BACKED SECURITIES ETF,ISHARES GNMA BOND
ETF}:2018:-1,2017:0,2016:0
```

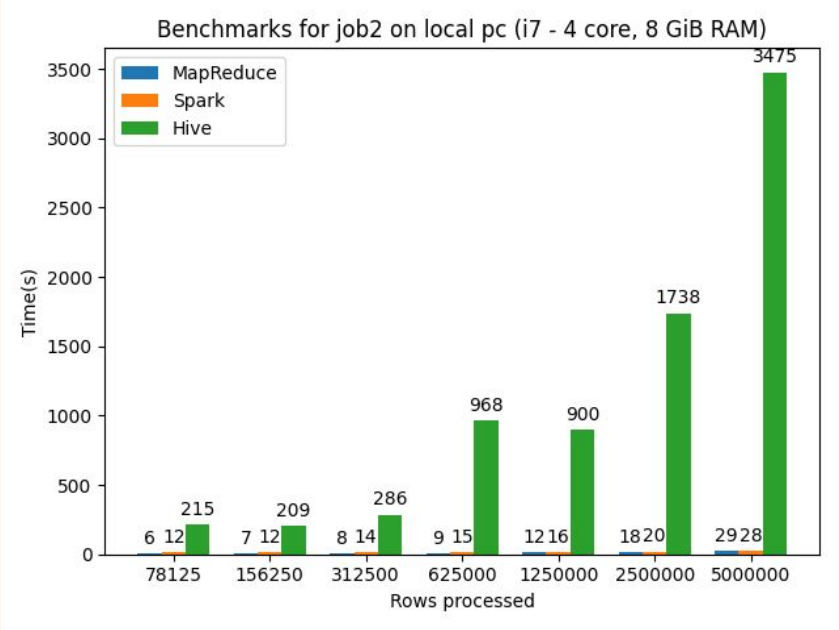
TEMPI DI ESECUZIONE

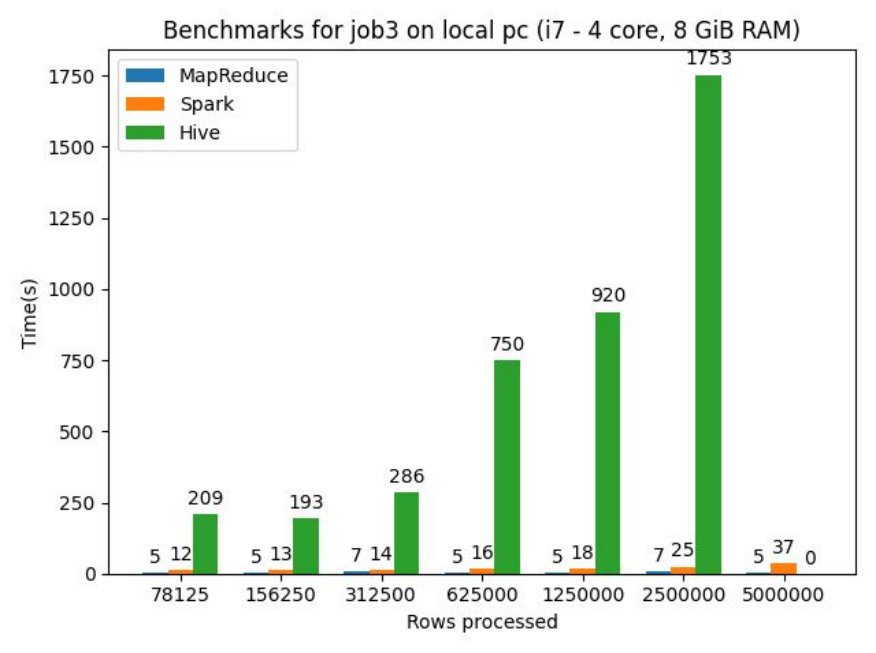
Sono riportati in seguito i grafici relativi ai tempi di esecuzione dei tre job eseguiti con MapReduce, Spark ed Hive, sia in locale che su un cluster EMR su Amazon AWS.

Esecuzione locale

I tre job sono stati eseguiti su una macchina avente un i7 a 2 core, 4 thread, 8 GiB di RAM, sul file hsp di input crescente rispetto al numero di linee, ma con dimensione massima circa un quarto più piccolo rispetto all'originale.

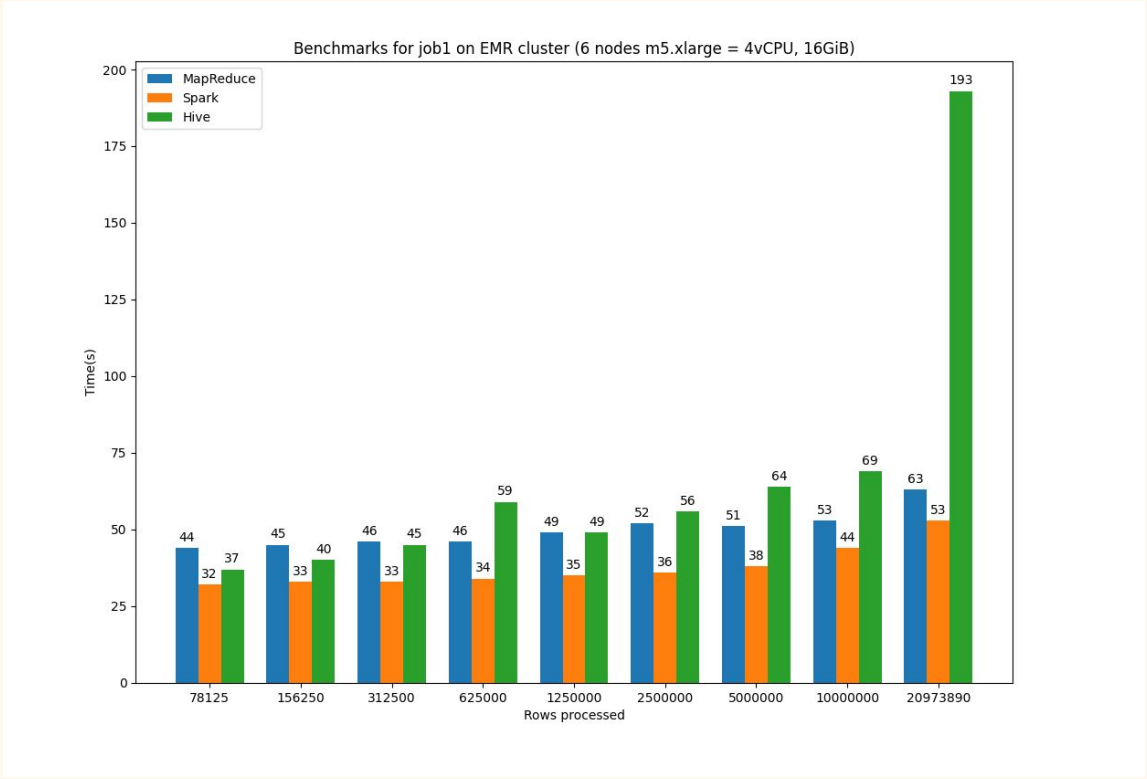

Benchmarks for job1 on local pc (i7 - 4 core, 8 GiB RAM)
MapReduce
Spark
Hive
Time(s)
Rows processed
78125
156250
312500
625000
1250000
2500000
5000000
5 11 97
5 13 65
6 14 67
8 14 115
11 17 82
17 21 102
30 29 139

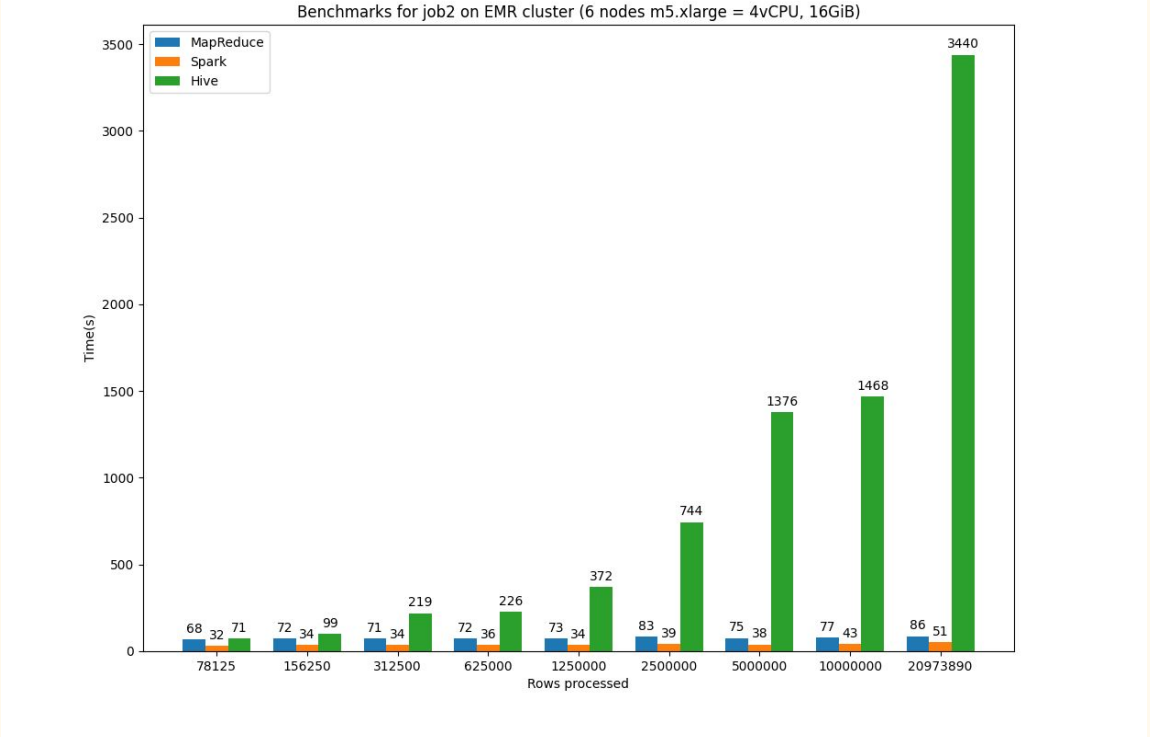

Benchmarks for job2 on local pc (i7 - 4 core, 8 GiB RAM)
MapReduce
Spark
Hive
Time(s)
Rows processed
78125
156250
312500
625000
1250000
2500000
5000000
6 12 215
7 12 209
8 14 286
9 15 968
12 16 900
18 20 1738
29 28 3475

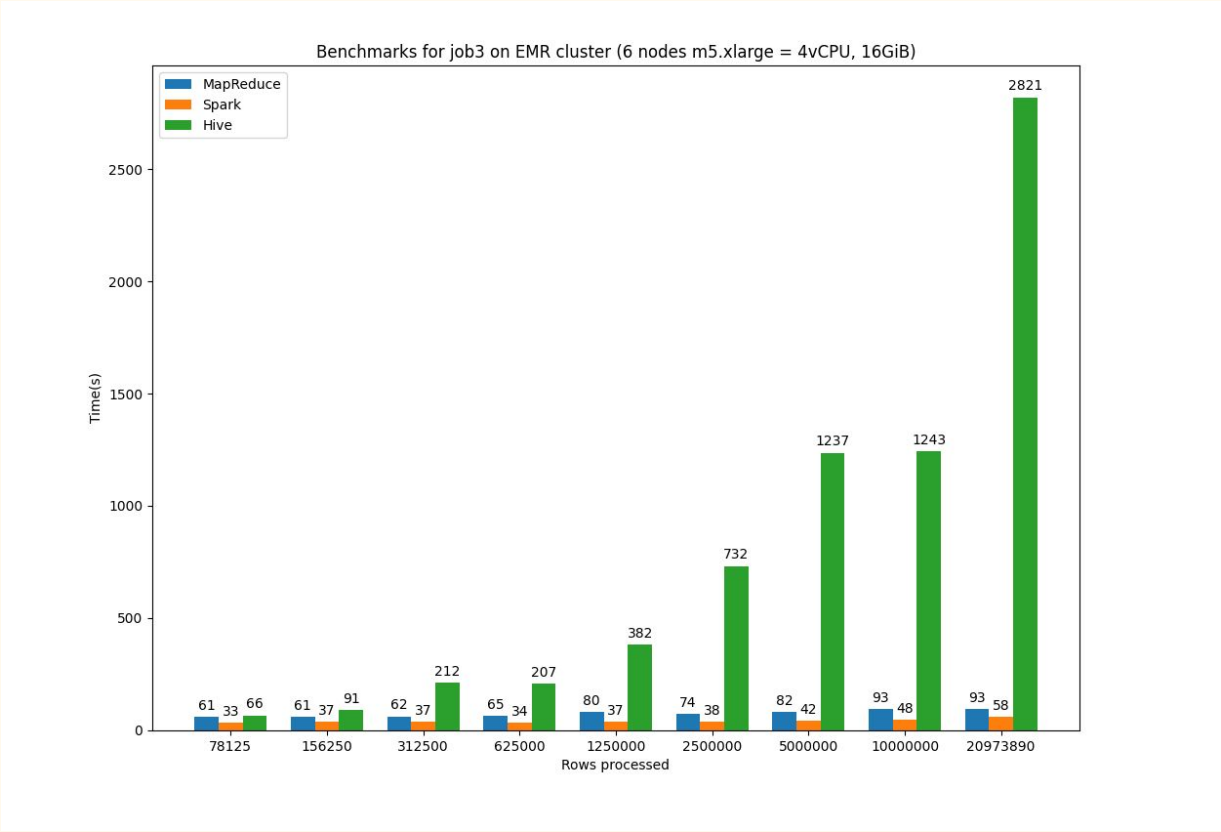


Esecuzione su cluster

I tre job sono stati eseguiti su un cluster EMR di Amazon AWS, composto da 6 nodi *m5.xlarge* ognuno avente 4 vCPU e 16 GiB di RAM, con input di dimensione crescente fino a raggiungere il file *hsp* originale.


Benchmarks for job1 on EMR cluster (6 nodes m5.xlarge = 4vCPU, 16GiB)
MapReduce
Spark
Hive
Time(s)
Rows processed
78125
156250
312500
625000
1250000
2500000
5000000
10000000
20973890
44
32
37
45
33
40
46
33
45
46
34
59
49
35
49
52
36
56
51
38
64
53
44
69
63
53
193


Benchmarks for job2 on EMR cluster (6 nodes m5.xlarge = 4vCPU, 16GiB)
MapReduce
Spark
Hive
Time(s)
Rows processed
78125
156250
312500
625000
1250000
2500000
5000000
10000000
20973890
68
32
71
72
34
99
71
34
219
72
36
226
73
34
372
83
39
744
75
38
1376
77
43
1468
86
51
3440


Benchmarks for job3 on EMR cluster (6 nodes m5.xlarge = 4vCPU, 16GiB)
MapReduce
Spark
Hive
Time(s)
0
500
1000
1500
2000
2500
61 33 66
61 37 91
62 37 212
65 34 207
80 37 382
74 38 732
82 42 1237
93 48 1243
93 58 2821
78125
156250
312500
625000
1250000
2500000
5000000
10000000
20973890
Rows processed