Primo Progetto - Big Data 2021

Gruppo “The Hadoopers”
Emanuele Spezia mat. 486819
Giuseppe Brescia mat. 550718

Specifiche progetto

Si consideri il dataset Daily Historical Stock Prices che contiene l’andamento giornaliero di un’ampia selezione di azioni sulla borsa di New York (NYSE) e sul NASDAQ dal 1970 al 2018. Il dataset è formato da due file CSV.

Ogni riga del primo (historical_stock_prices) ha i seguenti campi:

- ticker: simbolo univoco dell’azione (https://en.wikipedia.org/wiki/Ticker_symbol)
- open: prezzo di apertura
- close: prezzo di chiusura
- adj_close: prezzo di chiusura “modificato” (potete trascurarlo)
- lowThe: prezzo minimo
- highThe: prezzo massimo
- volume: numero di transazioni
- date: data nel formato aaaa-mm-gg

Il secondo (historical_stocks) ha invece questi campi:

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

Dopo avere eventualmente eliminato dal dataset dati errati o non significativi, progettare e realizzare in: (a) MapReduce, (b) Hive e (c) Spark core:

1. Un job che sia in grado di generare un report contenente, per ciascuna azione: (a) la data della prima quotazione, (b) la data dell’ultima quotazione, (c) la variazione percentuale della quotazione (differenza percentuale tra il primo e l’ultimo prezzo di chiusura presente nell’archivio), (d) il prezzo massimo e quello minimo e (e) (facoltativo) il massimo numero di giorni consecutivi in cui l’azione è cresciuta (chiusura maggiore dell’apertura) con indicazione dell’anno

in cui questo è avvenuto. Il report deve essere ordinato per valori decrescenti del punto b.

2. Un job che sia in grado di generare un report contenente, per ciascun settore e per ciascun anno del periodo 2009-2018: (a) la variazione percentuale della quotazione del settore1 nell'anno, (b) l'azione del settore che ha avuto il maggior incremento percentuale nell'anno (con indicazione dell'incremento) e (c) l'azione del settore che ha avuto il maggior volume di transazioni nell'anno (con indicazione del volume). Il report deve essere ordinato per nome del settore.
3. Un job in grado di generare le coppie di aziende che si somigliano (sulla base di una soglia scelta a piacere) in termini di variazione percentuale mensile nell'anno 2017 mostrando l'andamento mensile delle due aziende (es. Soglia=1%, coppie: 1:{Apple, Intel}: GEN: Apple +2%, Intel +2,5%, FEB: Apple +3%, Intel +2,7%, MAR: Apple +0,5%, Intel +1,2%, ...; 2:{Amazon, IBM}: GEN: Amazon +1%, IBM +0,5%, FEB: Amazon +0,7%, IBM +0,5%, MAR: Amazon +1,4%, IBM +0,7%, ..)

Specifiche Tecniche

Sistema Locale

- Hardware 1:
  OS: Linux 64 bit (Deepin OS 20.2)
  CPU: 1,70 GHz Intel Core i5-4210U
  RAM: 8 GB

- Hardware 2:
  OS: MacOS BigSur 11.2.3
  CPU: 3,3 GHz Intel Core i7 dual-core
  RAM: 16 GB 2133 MHz LPDDR3

- Software:
  Hadoop 3.3.2
  Python 3.9
  Hive 2.3.7
  Spark core 3.1.1

Sistema Cluster

- Hardware (x3):
  OS: AMI (Amazon Machine Images)
  CPU: 4 vCPU 2.5 GHz Custom Intel Xeon Platinum 8175M
  RAM: 16 GB

- Software:
  Hadoop Amazon 2.10.1
  Hive 2.3.7
  Spark 2.4.7

Implementazione Jobs

Di seguito viene fornita una possibile implementazione dei tre Job nei tre differenti strumenti sopracitati.

MapReduce (Pseudocodice)

❏ Linguaggio di programmazione: Python 3.9

Verrà ora presentato lo pseudocodice dei tre job implementati in MapReduce.

Job 1 - Mapper

Nel primo Job il mapper compie la maggior parte del lavoro, si occupa, infatti, della creazione di differenti dizionari che serviranno ad immagazzinare i risultati dei punti richiesti per il primo Job. Nel codice completo, in allegato al documento, è presente la specifica implementazione di tutti i passaggi per generare i risultati richiesti, che qui nello pseudocodice verranno riassunti in modalità semplificata.
Anche il punto facoltativo è stato implementato correttamente, prima calcolando per ogni Ticker, le date in cui il valore cresceva (prezzo di chiusura maggiore del prezzo di apertura), che sono state aggiunte ad una lista successivamente ordinata; infine si è calcolato per ogni data il giorno della settimana corrispondente e si è confrontato con quella successiva, effettuando una sottrazione fra data successiva a quella di partenza e quella di partenza, iterando per tutta la lista e spostando ogni volta la data di partenza a quella immediatamente successiva: se il valore era pari a 1 o -4 (le date sono degli interi con Lunedì=0, quindi il giorno successivo poteva essere o uno successivo all'altro, come Mercoledì e Giovedì, quindi 3-2=1, oppure anche il Venerdì e il Lunedì perchè la borsa chiude il weekend, quindi 0-4=-4) allora veniva incrementato un contatore, altrimenti si fermava il contatore e si verificava se fosse maggiore del massimo numero di giorni consecutivi calcolato in altre iterazioni.

```
<MAPPER>
ticker_list = {}
first_listing = {}
first_close = {}
last_listing = {}
last_close = {}
max_price = {}
min_price = {}
date_growing = {}
for line in input:
    ticker, open_price_str, close_price_str, adj_close, low_str, high_str, volume,
    date_str = line.split(",")

    if ticker not in ticker_list = insert in ticker_list(key = ticker)
    if date is first: insert in first_listing(key = ticker) and first_close(key = ticker)
    if this ticker is last: insert in last_listing(key = ticker) and last_close(key = ticker)
    if close_price is max: insert in max_price(key = ticker)
    if close_price is min: insert in min_price(key = ticker)
    if close_price - open_price > 0: insert date in date_growing(key = ticker)

for el in date_growing:
    max_days = cosecutive days ticker grows
    max_year = year in witch growth occurs

for el in first_close:
    variation = (last_close - first_close / first_close)*100

for ticker in ticker_list:
    print(ticker, first_listing, last_listing, variation, max_price, min_price,
          max_days, max_year)
```

Job 1 - Reducer

Il Reducer del primo Job si occupa di catturare i risultati del Mapper e di riordinarli secondo il giusto output finale. Un altro compito è quello di evitare errori nella lettura o nel passaggio di dati, attraverso un “error handler” che ha come obiettivo proprio quello di assicurarsi che tutto ciò che verrà stampato sarà corretto e conforme al formato prestabilito.

```
<REDUCER>
results = {}
for line in input:
    ticker, first_quot, last_quot, var, max_price, min_price, days_growth,
    year_growth = line.split("\t")

    results = insert(ticker, first_quot, last_quot, var, max_price, min_price,
                     days_growth, year_growth)

results.sort(last_listing(decreasing))
print(results)
```

Job 2 Join - Mapper

Nel secondo Job sono stati implementati 2 Mapper e 2 Reducer, a causa della necessità di unire due dataset appartenenti a file csv differenti. Quindi vi saranno i primi due Mapper e Reducer che avranno il compito proprio di creare un nuovo dataset, join dei due sopracitati; i secondi due avranno, invece, il compito di rispondere alle richieste di progetto per il Job 2.
Il mapper del Join ha quindi il compito di generare, a partire dai due file di input differenti, una stringa diversa per ogni dataset, che abbia, però, lo stesso numero di elementi. Gli elementi non comuni, saranno etichettati con “-” e verranno poi riassemblati dal Reducer.

```
<MAPPER_JOIN>
for line in input:
    words = line.split(",")

    if len(words) == 8:
        # Row of historical_stock_prices.csv
        ticker, open_price_str, close_price_str, adj_close, low_str, high_str, volume_str, date_str = words
        print(ticker, close_price_str, volume_str, date_str)

    else:
        # Row of historical_stocks.csv
        if len(words) == 5:
            ticker, exchange, name, sector, industry = words
            print(ticker, sector)
```

Job 2 Join - Reducer

Il reducer del Join si occupa di prendere in input le stringhe del Mapper e costruire il nuovo dataset sostituendo, ogni qual volta appaia il simbolo “-”, il ticker e il settore di appartenenza, così da generare la stringa definitiva che comprenda anche il settore, dato che abbiamo dovuto prelevare dal secondo dataset. Infine stampa tutte le stringhe, definendo il nuovo dataset di partenza, unione dei primi due, limitata ai campi a noi necessari per completare il secondo Job.

```
<REDUCER_JOIN>
for line in input:
    ticker, close_price_str, volume_str, date_str, sector = line.split('\t')
    if sector != '-':
        ticker_result = ticker
        sector_result = sector
        continue
    print(ticker_result, close_price, volume, date, sector_result)
```

Job 2 - Mapper

Ora che i dati sono corretti e completi, ci apprestiamo a rispondere alle richieste del Job 2. Con il Mapper, eliminiamo da dataset, le date che non sono comprese nel range 2009-2018 e stampiamo, quindi passiamo al Reducer, tutte quelle in questo intervallo.

```
<MAPPER>
for line in input:
    ticker, close, volume, date, sector = line.split('\t')
    start_period = "2009-01-01"
    end_period = "2018-12-31"

    if (date >= start_period) and (date <= end_period) and sector != 'None' and sector != 'N/A':
        print(ticker, date, close, volume, sector)
```

Job 2 - Reducer

Una volta che l'intervallo è stato definito, il Reducer si occuperà della gran parte del lavoro e attraverso un sistema di dizionari e metodi per calcolare i dati richiesti, stamperà nel formato corretto, l'output definitivo del Job 2.

```
<REDUCER>
sector_ticker_list = {}
volume_year = {}
beginning_year = {}
ending_year = {}
results = {}
for line in input:
    ticker, date, close, volume, sector = line.split('\t')

    if sector not in sector_ticker_list = insert ticker in sector_ticker_list(key = sector)
    else insert ticker in sector_ticker_list(key = sector)
    sum volume and insert in volume_year(key = ticker)
    for year in range(2009, 2019, 1):
        if date is first in year: insert date and close in beginning_year(key = ticker)
        if date is last in year: insert date and close in ending_year(key = ticker)
for year in range(2009, 2019, 1):
    sum close of beginning_year and insert in total_begin
    sum close of ending_year and insert in total_end
for ticker in sector_ticker_list:
    if (ending_year - beginning_year / beginning_year)*100 is the max:
        best_ticker = ticker
        var_max = (ending_year - beginning_year / beginning_year)*100
    if volume in volume_year is max:
        best_ticker_volume = ticker
        volume_max = volume

for sector in sector_ticker_list:
    results = insert(sector, year, (total_end - total_begin / total_begin)*100, best_ticker,
                     var_max, best_ticker_volume, volume_max)

results.sort(sector)
print(results)
```

Job 3 - Mapper

Per il Job 3 il Mapper si occupa di definire e delimitare lo studio, all'anno 2017, selezionando solo i dati facenti parte di quest'ultimo.

```
<MAPPER>
for line in input:
    ticker, open, close, adj_close, low, high, volume, date = line.split(",")

    if date_dt.year == 2017:
        print(ticker, date, close)
```

Job 3 - Reducer

Anche in questo caso, come nel Job 2, il Reducer si occupa della gran parte del lavoro, calcolando prima la variazione percentuale di ogni settore per ogni mese dell'anno 2017 e poi mettendole a confronto mese per mese, stampando solo le coppie di Ticker che, per ogni mese di tutto l'anno, hanno la crescita, o decrescita, percentuale simile, con soglia di similarità pari a 3%.

```
<REDUCER>
first_close = {}
last_close = {}
ticker_month_var = {}

for line in input:
    ticker, date, close = line.split('\t')

    for month in range(1, 13):
        if date is first in month: insert date and close in first_close(key = ticker)
        if date is last in month: insert date and close in last_close(key = ticker)
for ticker in first_close:
    for month in range(1, 13):
        ticker_month_var = (last_close - first_close / first_close)*100

couple_number = 0
for ticker1 in ticker_month_var:
    for ticker2 in ticker_month_var:
        count = 0
        if (ticker_compare != ticker) and (ticker_compare not in ignore_ticker.keys()):
            for month in range(1, 13):
                if ticker_month_var(ticker1) and ticker_month_var(ticker2) is similar(threshold = 3%)
                count += 1

                if count == 12
                couple_number +=1
                results(output_format)
print(results)
```

Hive

- ❏ Linguaggio di programmazione: Hql, Python 3.9

Verrà ora presentato lo pseudocodice dei tre job implementati in Hive.

Job 1

Fase di caricamento del dataset da formato csv in forma tabellare.

```
CREATE TABLE  historical_stock_prices (ticker STRING,
                                       open DOUBLE,
                                       close DOUBLE,
                                       adj_close DOUBLE,
                                       lowThe DOUBLE,
                                       highThe DOUBLE,
                                       volume DOUBLE,
                                       data DATE)

ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';

LOAD DATA LOCAL INPATH 'historical_stocks_prices.csv' OVERWRITE INTO TABLE historical_stock_prices;
```

Il punto A e B calcolano la data della prima e ultima chiusura tramite rank function.

```
/*  Punto A */

CREATE TABLE firstClose AS
select * from (
select ticker,close, data,
rank() over ( partition by ticker order by data asc) as rank
FROM historical_stock_prices) t
WHERE rank = 1;

/*  Punto B */

CREATE TABLE lastClose AS
select * from (
select ticker,close,data,
rank() over ( partition by ticker order by data desc) as rank
FROM historical_stock_prices) t
WHERE rank = 1;
```

Il punto C calcola l'incremento percentuale di ogni ticker all'interno dell'archivio.

```
/*  Punto C */

CREATE TABLE percentageIncrease AS
SELECT DISTINCT f.ticker, (100*(l.close-f.close)/f.close) as percentage
FROM firstClose as f JOIN lastClose as l ON f.ticker=l.ticker;
```

Il punto D calcola il prezzo massimo e minimo assunto da ogni ticker all'interno dell'archivio attraverso due rank function.

```
/*  Punto D */

CREATE TABLE minLowThe AS
select * from (
select ticker,lowThe,
rank() over ( partition by ticker order by lowThe asc) as rank
FROM historical_stock_prices) t
WHERE rank = 1;

CREATE TABLE maxHighThe AS
select * from (
select ticker,highThe,
rank() over ( partition by ticker order by highThe desc) as rank
FROM historical_stock_prices) t
WHERE rank = 1;
```

Il punto E viene implementato tramite la funzione job1facoltativo.py in codice Python.

```
/*  Punto E */

ADD FILE /Users/giuseppebrescia/Desktop/StockPricesAnalytics-master/Job1/Hive/job1facoltativo.py;

CREATE TABLE first_query AS
SELECT TRANSFORM (ticker,open,close,adj_close,lowThe,highThe,volume,data)
            USING 'job1facoltativo.py'
            AS (ticker STRING,days_growing INT, year_growing STRING)
FROM historical_stock_prices;
```

La funzione job1facoltativo.py riceve il dataset tramite standard input, lo elabora e fornisce per ogni ticker il massimo numero di giorni consecutivi in cui l'azione è cresciuta.

```
date_growing = {}
days_growing = {}
year_growing = {}

input_file = sys.stdin

# Not read the first line of input file
next(input_file)

# Read lines from input_file
for line in input_file:

    # Removing leading/trailing whitespaces
    line = line.strip("\n ")

    # Parse the input elements
    ticker, open_price_str, close_price_str, adj_close, low_str, high_str, volume, date_str = line.split("\t")

    # Convert String to Date
    date_dt = dt.strptime(date_str, '%Y-%m-%d')
    # Convert String to float
    open_price = float(open_price_str)
    close_price = float(close_price_str)

    if close_price - open_price > 0:
        if ticker not in date_growing:
            date_growing[ticker] = []
            date_growing[ticker].append(date_dt)
        else:
            date_growing[ticker].append(date_dt)
```

```
# Final calculation of consecutive days in which the ticker has grown with the associated year
for ticker in date_growing.keys():
    sorted_date = sorted(date_growing[ticker])
    max_year = 0
    # Initialization
    days_count = 1
    max_days = 1

    for i in range(1, len(sorted_date)):
        if (sorted_date[i].weekday() - sorted_date[i - 1].weekday()) == 1 or (
                sorted_date[i].weekday() - sorted_date[i - 1].weekday()) == -4:
            days_count += 1
        else:
            if days_count > max_days:
                max_days = days_count
                max_year = sorted_date[i].year
                days_count = 1
            else:
                days_count = 1
    #
    days_growing[ticker] = max_days
    year_growing[ticker] = max_year

for ticker in days_growing.keys():
    print('%s\t%i\t%s' % (ticker, days_growing[ticker], year_growing[ticker]))
```

Infine attraverso una serie di join viene elaborato il report finale che lega i campi richiesti in output ad ogni azione e ne stampa i soli primi 10 risultati.

```
CREATE TABLE reportFinal AS
SELECT a.ticker,a.firstData,a.lastData,a.lowThe,a.highThe,a.percentage, p.days_growing, p.year_growing
FROM firstJoin as a JOIN first_query as p ON a.ticker=p.ticker
ORDER BY ticker ASC LIMIT 10;
```

Job 2

Fase di caricamento dei due dataset. Questa volta il caricamento e l'elaborazione del secondo dataset viene affidata ad OpenCSVSerde. Successivamente i due dataset vengono uniti tramite operazione di JOIN e filtrati secondo le date richieste.

```
CREATE TABLE historical_stock_prices (ticker STRING,
                                      open DOUBLE,
                                      close DOUBLE,
                                      adj_close DOUBLE,
                                      lowThe DOUBLE,
                                      highThe DOUBLE,
                                      volume DOUBLE,
                                      data STRING)


ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
LOAD DATA LOCAL INPATH 'historical_stock_prices.csv' OVERWRITE INTO TABLE historical_stock_prices;

CREATE TABLE historical_stocks (ticker STRING,market STRING,name STRING,sector STRING,industry STRING)

ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES (
   "separatorChar" = ",",
   "quoteChar"     = "\");
LOAD DATA LOCAL INPATH 'historical_stocks.csv' OVERWRITE INTO TABLE historical_stocks;

CREATE TABLE joined AS
SELECT f.ticker as ticker, f.close as close, f.volume as volume, f.data as data, YEAR(data) as anno, l.sector as sector
FROM
    historical_stock_prices as f JOIN historical_stocks as l ON f.ticker=l.ticker
WHERE l.sector != 'N/A' AND data > '2009-01-01' AND data < '2018-12-31';
```

Nel punto A viene calcolata la variazione della quotazione del settore per ogni anno.

```sql
/*  Punto A */

CREATE TABLE temp1 AS
SELECT ticker,close,data,anno,sector,
RANK() OVER (PARTITION BY sector,anno ORDER BY data ASC) as rn
FROM joined;

CREATE TABLE firstCloseForYear AS
SELECT ticker,close,data,anno,sector
FROM temp1
WHERE rn = 1;

CREATE TABLE sumFirstQuote AS
SELECT SUM(close) as sumClose,anno,sector
FROM firstCloseForYear
GROUP BY anno,sector;

CREATE TABLE temp2 AS
SELECT ticker,close,data,anno,sector,
RANK() over (PARTITION BY anno,sector ORDER BY data DESC) as rn
FROM joined;

CREATE TABLE lastCloseForYear AS
SELECT ticker,close,data,anno,sector
FROM temp2
WHERE rn = 1;

CREATE TABLE sumLastQuote AS
SELECT SUM(close) as sumClose,anno,sector
FROM lastCloseForYear
GROUP BY anno,sector;

CREATE TABLE sectorVarByYear AS
SELECT f.anno as anno, f.sector as sector, (100*(l.sumClose-f.sumClose)/f.sumClose) as percentageSectorVar
FROM
    sumFirstQuote as f JOIN sumLastQuote as l ON f.sector=l.sector AND f.anno=l.anno;
```

Nel punto B vengono calcolati in una prima fase la prima e l'ultima chiusura di ogni azione nell'anno. Successivamente ne viene calcolato l'incremento percentuale e infine vengono selezionate solo le azioni con l'incremento percentuale massimo nell'anno.

```sql
/*  Punto B */

CREATE TABLE temp3 AS
SELECT ticker,close,data,anno,sector,
ROW_NUMBER() OVER (PARTITION BY ticker,sector,anno ORDER BY data ASC) as rn
FROM joined;

CREATE TABLE firstCloseForYear2 AS
select ticker,close,data,anno,sector
FROM temp3
WHERE rn = 1;

CREATE TABLE temp4 AS
SELECT ticker,close,data,anno,sector,
ROW_NUMBER() over (PARTITION by ticker,sector,anno ORDER BY data DESC) as rn
FROM joined;

CREATE TABLE lastCloseForYear2 AS
SELECT ticker,close,data,anno,sector
FROM temp4
WHERE rn = 1;

CREATE TABLE maxIncrement AS
SELECT f.ticker as ticker,f.anno as anno, f.sector as sector, (100*(l.close-f.close)/f.close) as percentageIncrease
FROM
    firstCloseForYear2 as f JOIN lastCloseForYear2 as l ON f.ticker=l.ticker AND f.anno=l.anno;

CREATE TABLE maxIncrementByYear AS
SELECT ticker,percentageIncrease,anno, sector, ROW_NUMBER() over (PARTITION by anno,sector ORDER BY percentageIncrease DESC) as rn
FROM maxIncrement;

CREATE TABLE maxIncrementByYearB AS
SELECT ticker,percentageIncrease,anno,sector
FROM maxIncrementByYear
WHERE rn=1
ORDER BY anno ASC;
```

Nel punto C viene calcolato il volume di ogni azione nell'anno per selezionarne quella con volume maggiore.

```
CREATE TABLE first AS
SELECT sector, anno, ticker, SUM(volume) as volumeTot
FROM joined
GROUP BY sector, anno, ticker;

CREATE TABLE semifinal AS
select sector, anno, ticker, volumeTot,
ROW_NUMBER() over ( partition by sector,anno order by volumeTot DESC) as rn
FROM first;

CREATE TABLE final AS
SELECT sector, anno, ticker, volumeTot
FROM semifinal
WHERE rn = 1;
```

Infine una serie di JOIN per presentare il report finale.

```
/* Report  */

CREATE TABLE report AS
SELECT a.sector as sector, a.anno as anno, a.percentageSectorVar as percentageSectorVar,
b.ticker as ticker,b.percentageIncrease as percentageIncrease
FROM sectorVarByYear as a JOIN maxIncrementByYearB as b
ON a.sector=b.sector AND a.anno=b.anno;

CREATE TABLE reportFinal AS
SELECT a.sector as sector, a.anno as anno, a.percentageSectorVar as percentageSectorVar,
a.ticker as ticker, a.percentageIncrease as percentageIncrease, b.ticker as ticker2, b.volumeTot as volumeTot
FROM report as a JOIN final as b
ON a.sector=b.sector AND a.anno=b.anno
ORDER BY anno ASC;
```

Job 3

Prima fase di caricamento del dataset e di filtraggio nell'anno 2017.

```
CREATE TABLE IF NOT EXISTS historical_stock_prices (ticker STRING,
                                                    open DOUBLE,
                                                    close DOUBLE,
                                                    adj_close DOUBLE,
                                                    lowThe DOUBLE,
                                                    highThe DOUBLE,
                                                    volume DOUBLE,
                                                    data STRING)


ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
LOAD DATA LOCAL INPATH 'historical_stock_prices.csv' OVERWRITE INTO TABLE historical_stock_prices;

CREATE TABLE IF NOT EXISTS filter AS
SELECT ticker, close, data,MONTH(data) as mese
FROM historical_stock_prices
WHERE data >= '2017-01-01' AND data <= '2017-12-31';
```

Vengono calcolate le prime e le ultime chiusure delle azioni per ogni mese e l'incremento percentuale nel mese.

```
CREATE TABLE temp5 AS
SELECT ticker,close,data,mese,
ROW_NUMBER() OVER (PARTITION BY ticker,mese ORDER BY data ASC) as rn
FROM filter;

CREATE TABLE firstCloseForMonth AS
select ticker,close,data,mese
FROM temp5
WHERE rn = 1;

CREATE TABLE temp6 AS
SELECT ticker,close,data,mese,
ROW_NUMBER() over (PARTITION by ticker,mese ORDER BY data DESC) as rn
FROM filter;

CREATE TABLE lastCloseForMonth AS
SELECT ticker,close,mese
FROM temp6
WHERE rn = 1;

CREATE TABLE maxIncrement AS
SELECT f.ticker as ticker,f.mese as mese, (100*((l.close-f.close)/f.close)) as percentageIncrease
FROM
    firstCloseForMonth as f JOIN lastCloseForMonth as l ON f.ticker=l.ticker AND f.mese=l.mese
ORDER BY ticker, mese ASC;
```

Nella tabella maxIncrement si trovano i ticker con l'incremento percentuale ordinati per mese ascendente. Quindi tramite la LEAD function viene costruita la tabella final dalla quale viene selezionata solo la prima riga (in extrafinal) per ogni ticker contenente questa volta il ticker e dodici campi contenenti l'andamento percentuale del ticker in ogni mese.

```
CREATE TABLE final AS
SELECT mese, ticker, percentageIncrease as gen,
LEAD ( percentageIncrease ,1 , 0) OVER (PARTITION BY ticker ORDER BY mese ASC) as feb,
LEAD ( percentageIncrease ,2 , 0) OVER (PARTITION BY ticker ORDER BY mese ASC) as mar,
LEAD ( percentageIncrease ,3 , 0) OVER (PARTITION BY ticker ORDER BY mese ASC) as apr,
LEAD ( percentageIncrease ,4 , 0) OVER (PARTITION BY ticker ORDER BY mese ASC) as mag,
LEAD ( percentageIncrease ,5 , 0) OVER (PARTITION BY ticker ORDER BY mese ASC) as giu,
LEAD ( percentageIncrease ,6 , 0) OVER (PARTITION BY ticker ORDER BY mese ASC) as lug,
LEAD ( percentageIncrease ,7 , 0) OVER (PARTITION BY ticker ORDER BY mese ASC) as ago,
LEAD ( percentageIncrease ,8 , 0) OVER (PARTITION BY ticker ORDER BY mese ASC) as sett,
LEAD ( percentageIncrease ,9 , 0) OVER (PARTITION BY ticker ORDER BY mese ASC) as ott,
LEAD ( percentageIncrease ,10 , 0) OVER (PARTITION BY ticker ORDER BY mese ASC) as nov,
LEAD ( percentageIncrease ,11 , 0) OVER (PARTITION BY ticker ORDER BY mese ASC) as dic
FROM maxIncrement;

CREATE TABLE extrafinal AS
SELECT ticker, gen, feb, mar, apr, mag ,giu, lug, ago ,sett, ott, nov, dic
FROM final
WHERE mese = 1;
```

Infine dentro report vengono selezionate le coppie di ticker che in tutto l'anno hanno avuto un incremento percentuale che si discosta di una soglia pari a 3.

```
CREATE TABLE report AS
    select distinct
        p1.ticker,p2.ticker as ticker2,p1.gen,p2.gen as gen2,p1.feb,p2.feb as feb2,p1.mar,p2.mar as mar2,
        p1.apr,p2.apr as apr2,p1.mag,p2.mag as mag2,p1.giu,p2.giu as giu2,p1.lug,p2.lug as lug2,p1.ago,
        p2.ago as ago2,p1.sett,p2.sett as sett2,p1.ott,p2.ott as ott2,p1.nov,p2.nov as nov2,p1.dic,p2.dic as dic2
    from extrafinal p1
        inner join extrafinal p2
            on ((p1.gen between p2.gen -3 and p2.gen + 3) and
            (p1.feb between p2.feb -3 and p2.feb + 3) and
            (p1.mar between p2.mar -3 and p2.mar + 3) and
            (p1.apr between p2.apr -3 and p2.apr + 3) and
            (p1.mag between p2.mag -3 and p2.mag + 3) and
            (p1.giu between p2.giu -3 and p2.giu + 3) and
            (p1.lug between p2.lug -3 and p2.lug + 3) and
            (p1.ago between p2.ago -3 and p2.ago + 3) and
            (p1.sett between p2.sett -3 and p2.sett + 3) and
            (p1.ott between p2.ott -3 and p2.ott + 3) and
            (p1.nov between p2.nov -3 and p2.nov + 3) and
            (p1.dic between p2.dic -3 and p2.dic + 3)
            and p1.ticker < p2.ticker);
```

Spark core (Pseudocodice)

- ❏ Linguaggio di programmazione: Python 3.9

Verrà ora presentato lo pseudocodice dei tre job implementati in Spark core.

Job 1

Nel primo Job con Spark, dopo l'acquisizione dei dati, si utilizza una funzione di reduceByKey per raccogliere le prime e le ultime date in cui un'azione è apparsa nel dataset. Si è poi calcolata la variazione percentuale per ogni azione, con un join delle prime e ultime date delle quotazioni sopra citate e infine è stato calcolato attraverso delle funzioni di supporto (maxDaysGrowing e yearMaxGrowing) i massimi giorni di crescita consecutivi e l'anno in cui ciò è avvenuto. I risultati sono poi stati ordinati per essere consoni al formato output richiesto.

```
# Support functions:
# Calculates the consecutive days on which the stock grows
def maxDaysGrowing(date_list)
# Calculates the year in which there are days the stock grows
def yearMaxGrowing(date_list)
# Initialize SparkSession with the proper configuration
spark = SparkSession(configuration)
# Read the input file and obtain an RDD with a record for each line
input_data = spark.sparkContext.textFile(input_filepath).cache()
# First close_price and associated date
first_quot_RDD = input_data.reduceByKey(line => date)
# Last close_price and associated date
last_quot_RDD = input_data.reduceByKey(line => date)
# General percentage variation
percentage_variation_RDD = last_quot_RDD.join(first_quot_RDD) \
    .map(line => close)
# Min close_price
min_price_RDD = input_data.reduceByKey(line => min_close)
# Max close_price
max_price_RDD = input_data.reduceByKey(line => max_close)
# Dates when ticker grew
ticker_grew_RDD = input_data.filter(line => close - open > 0).groupByKey() \
    .map(line => ticker, maxDaysGrowing(), yearMaxGrowing())
# Construct the result RDD
results = first_quot_RDD.join(last_quot_RDD).join(percentage_variation_RDD) \
    .join(min_price_RDD).join(max_price_RDD).join(ticker_grew_RDD)
# Format with the final view
results_formatted = results.map(line => output_format) \
    .sortBy(line => last_quot, ascending=False)
# Print the results in output path
results_formatted.coalesce(1).saveAsTextFile(output_filepath)
```

Job 2

Nel secondo Job sono stati inizialmente filtrati i dati, raccogliendo solo le date comprese dal 2009 al 2018. Poi sono stati uniti i due differenti input provenienti da due dataset distinti attraverso un join, facendo così corrispondere, ad ogni Ticker, il suo settore di appartenenza. Si è successivamente provveduto al solito calcolo della percentuale di variazione, stavolta divisa per anno, attraverso una chiave sia sul settore che sull'anno e con il calcolo successivo del volume e della variazione percentuale di ogni ticker per ogni anno. I risultati sono poi stati ordinati attraverso un map per corrispondere al formato output previsto.

```
# Support functions
# Calculate the percentage variation of all tickers each year
def percentageVarTot(closes)
# Calculate the percentage variation of a ticker
def percentageVar(closes)
# initialize SparkSession with the proper configuration
spark = SparkSession(configuration)
# Read the input file and obtain an RDD with a record for each line
input_data = spark.sparkContext.textFile(input_filepath).cache()
input_data_filtered = input_data.filter(line => (2009 <= date.year <= 2018))
input_data_stocks = spark.sparkContext.textFile(input_filepath_stocks).cache()
joined_RDD = input_data_filtered.join(input_data_stocks).map(line => input_format)
# Calculate the first and last close for every ticker in every year
first_price_ticker_RDD = joined_RDD.reduceByKey(line => date)
last_price_ticker_RDD = joined_RDD.reduceByKey(line => date)
# Percentage change in the price of the sector in the year
total_first_close = first_price_ticker_RDD.reduceByKey(line => sum close)
total_last_close = last_price_ticker_RDD.reduceByKey(line => sum close)
percentage_variation_RDD = total_last_close.join(total_first_close) \
    .map(line => percentageVarTot(closes))
# Ticker of the sector that had the greater percentage increase in the year
percentage_variation_ticker_RDD = last_price_ticker_RDD.join(first_price_ticker_RDD) \
    .map(line => percentageVar(closes)) \
    .reduceByKey(line => date)
# Ticker of the sector that had the highest volume of transactions in the year
total_volume_ticker_RDD = joined_RDD.reduceByKey(line => volume) \
    .reduceByKey(line => maxVolume)
# Construct the result RDD
results = percentage_variation_RDD.join(percentage_variation_ticker_RDD) \
    .join(total_volume_ticker_RDD)
# Format with the final view
results_formatted = results.map(line => output_format)\
    .sortBy(line => sector, ascending=False)
# Print the results in output path
results_formatted.coalesce(1).saveAsTextFile(output_filepath)
```

Job 3

Nel terzo Job di Spark, si è prima reso necessario filtrare il dataset prendendo in considerazione soltanto l'anno 2017. Poi si è passati al calcolo della variazione percentuale di ogni ticker per ogni mese dell'anno sopra citato e si è creato un dizionario per ogni ticker, che contenesse tutti i mesi e tutte le variazioni percentuali corrispondenti. Infine attraverso la scelta di una soglia (3%) si è verificato, coppia per coppia, che i due Ticker avessero una percentuale di crescita, o decrescita, che fosse compresa in questa soglia, per ogni mese dell'anno. In caso affermativo le due azioni verranno aggiunte all'output finale.

```
# Support functions
# Calculate the percentage variation of a ticker
def percentageVar(closes)
# Initialize SparkSession with the proper configuration
spark = SparkSession(configuration)
# Read the input file and obtain an RDD with a record for each line
input_data = spark.sparkContext.textFile(input_filepath).cache()
values_filtered_RDD = input_data.filter(line => date.year == 2017)
# Calculate the first and last close for every ticker in every month
first_price_ticker_RDD = values_filtered_RDD.reduceByKey(line => date)
last_price_ticker_RDD = values_filtered_RDD.reduceByKey(line => date)
# Structure: (ticker, month), %var
percentage_variation_ticker_RDD = last_price_ticker_RDD.join(first_price_ticker_RDD) \
    .map(line => percentageVar(closes))
# Structure: january, (ticker, %var)
ticker_var_every_month_RDD = percentage_variation_ticker_RDD.filter(line => date.month)
# Create a map of previous RDD
ticker_var_every_month = ticker_var_every_month_RDD.collectAsMap()
# Create and populate a dict of variation and month
ticker_variation_dict = {}
for el in ticker_var_every_month:
    if el not in ticker_variation_dict:
        ticker_variation_dict[el] = {}
        ticker_variation_dict[el][1] = ticker_var_every_month[el]
    else:
        ticker_variation_dict[el][1] = ticker_var_every_month[el]
# Calculate the ticker witch percentage variation is similar for all months
ignore_ticker = {}
final_results = {}
couple_number = 0
for ticker in ticker_variation_dict:
    ignore_ticker[ticker] = ticker
    for ticker_compare in ticker_variation_dict:
        count = 0
        if (ticker_compare != ticker) and (ticker_compare not in ignore_ticker.keys()):
            for m in range(1, 13):
                if (m in ticker_variation_dict[ticker].keys()) and (m in ticker_variation_dict[ticker_compare].keys()):
                    if (0 <= (ticker_variation_dict[ticker][m] - ticker_variation_dict[ticker_compare][m]) <= 3) or (
                            -3 <= (ticker_variation_dict[ticker][m] - ticker_variation_dict[ticker_compare][m]) <= 0):
                        count += 1
                    if count == 12:
                        couple_number += 1
                        final_results(output_format)
# Final result output
results_RDD = spark.sparkContext.parallelize(list(final_results.items()))
results_final_RDD = results_RDD.map(lambda x: (dict([x])))
results_final_RDD.coalesce(1).saveAsTextFile(output_filepath)
```

Risultati Job (prime 10 righe)

Job 1:

Ticker	First quot	Last quot	Percentage	Max price	Min price	Days growth	Year growth
ZYNE	2015-08-05	2018-08-24	-61.23 %	37.98	4.75	5	2015
ZYME	2017-04-28	2018-08-24	+6.08 %	22.13	6.43	5	2018
ZUO	2018-04-12	2018-08-24	+66.9 %	36.06	19.23	5	2018
ZUMZ	2005-05-06	2018-08-24	+139.24 %	51.8	5.79	8	2009
ZTS	2013-02-01	2018-08-24	+188.94 %	93.37	28.48	8	2014
ZTR	1988-09-26	2018-08-24	-70.15 %	44.0	10.52	10	2013
ZTO	2016-10-27	2018-08-24	+17.8 %	22.48	11.35	9	2018
ZSAN	2015-01-27	2018-08-24	-98.17 %	243.2	3.86	5	2015
ZS	2018-03-16	2018-08-24	+34.64 %	43.1	25.0	8	2018
ZOES	2014-04-11	2018-08-24	-45.91 %	45.6	8.73	9	2015

Job 2:

Sector	Year	Percentage	Best percentage ticker		Best volume ticker	
BASIC INDUSTRIES	2018	-3.08 %	XRM	+213.82 %	VALE	3710091900
CAPITAL GOODS	2018	-1.46 %	RFIL	+341.51 %	F	6925457600
CONSUMER DURABLES	2018	+7.59 %	HEAR	+1394.38 %	GPK	551091500
CONSUMER NON-DURABLES	2018	+7.47 %	DFBG	+367.0 %	ABEV	4241614900
CONSUMER SERVICES	2018	-63.07 %	BHR	+16823.08 %	CMCSA	4253643400
ENERGY	2018	+8.26 %	LGCY	+235.44 %	GE	12264436100
FINANCE	2018	+4.14 %	DHCP	+609.88 %	BAC	11026441500
HEALTH CARE	2018	+15.12 %	TNDM	+1410.63 %	NVCN	7582018100
MISCELLANEOUS	2018	+10.4 %	TRHC	+184.92 %	BABA	3262795800
PUBLIC UTILITIES	2018	+100.5 %	ELC	+350309.83 %	T	5700827400

Job 3:

```
Soglia: 3%  Coppia 1:{AAXJ,ADRE}:   GEN: AAXJ 6.35%, ADRE 7.41% FEB: AAXJ 2.59%, ADRE 1.64% MAR: AAXJ 2.84%, ADRE 1.21%
APR: AAXJ 1.28%, ADRE 0.35% MAG: AAXJ 3.63%, ADRE 2.64% GIU: AAXJ 0.10%,     ADRE 0.55%  LUG: AAXJ 4.75%, ADRE 7.49%
AGO: AAXJ 1.15%, ADRE 2.35% SET: AAXJ -0.29%, ADRE -1.75%   OTT: AAXJ 3.90%, ADRE 1.78% NOV: AAXJ -0.43%, ADRE -1.62%
DEC: AAXJ 1.53%, ADRE 2.35%

Soglia: 3%  Coppia 2:{AAXJ,AIA}:    GEN: AAXJ 6.35%, AIA 6.45%  FEB: AAXJ 2.59%, AIA 1.93%  MAR: AAXJ 2.84%, AIA 2.62%
APR: AAXJ 1.28%, AIA 1.21%  MAG: AAXJ 3.63%, AIA 3.93%  GIU: AAXJ 0.10%, AIA 0.81%  LUG: AAXJ 4.75%, AIA 5.21%
AGO: AAXJ 1.15%, AIA 0.54%  SET: AAXJ -0.29%, AIA 0.39% OTT: AAXJ 3.90%, AIA 4.90%  NOV: AAXJ -0.43%, AIA 0.23%
DEC: AAXJ 1.53%, AIA 1.41%

Soglia: 3%  Coppia 3:{AAXJ,APF}:    GEN: AAXJ 6.35%, APF 4.62%  FEB: AAXJ 2.59%, APF 3.49%  MAR: AAXJ 2.84%, APF 2.54%
APR: AAXJ 1.28%, APF 2.19%  MAG: AAXJ 3.63%, APF 2.07%  GIU: AAXJ 0.10%, APF 1.59%  LUG: AAXJ 4.75%, APF 4.11%
AGO: AAXJ 1.15%, APF -0.23% SET: AAXJ -0.29%, APF -0.75%    OTT: AAXJ 3.90%, APF 3.22%  NOV: AAXJ -0.43%, APF -0.06%
DEC: AAXJ 1.53%, APF 2.62%

Soglia: 3%  Coppia 4:{AAXJ,CEZ}:    GEN: AAXJ 6.35%, CEZ 4.66%  FEB: AAXJ 2.59%, CEZ 2.14%  MAR: AAXJ 2.84%, CEZ 2.05%
APR: AAXJ 1.28%, CEZ 1.66%  MAG: AAXJ 3.63%, CEZ 1.55%  GIU: AAXJ 0.10%, CEZ -0.43% LUG: AAXJ 4.75%, CEZ 2.87%
AGO: AAXJ 1.15%, CEZ 1.58%  SET: AAXJ -0.29%, CEZ -2.74%    OTT: AAXJ 3.90%, CEZ 1.86%  NOV: AAXJ -0.43%, CEZ -0.18%
DEC: AAXJ 1.53%, CEZ 1.52%

Soglia: 3%  Coppia 5:{AAXJ,EDBI}:   GEN: AAXJ 6.35%, EDBI 5.85% FEB: AAXJ 2.59%, EDBI 1.53% MAR: AAXJ 2.84%, EDBI 2.87%
APR: AAXJ 1.28%, EDBI 1.73% MAG: AAXJ 3.63%, EDBI 1.37% GIU: AAXJ 0.10%, EDBI -0.39%    LUG: AAXJ 4.75%, EDBI 4.65%
AGO: AAXJ 1.15%, EDBI 2.26% SET: AAXJ -0.29%, EDBI -1.01%   OTT: AAXJ 3.90%, EDBI 1.56% NOV: AAXJ -0.43%, EDBI -0.61%
DEC: AAXJ 1.53%, EDBI 2.05%

Soglia: 3%  Coppia 6:{AAXJ,EEMA}:   GEN: AAXJ 6.35%, EEMA 6.46% FEB: AAXJ 2.59%, EEMA 2.33% MAR: AAXJ 2.84%, EEMA 2.63%
APR: AAXJ 1.28%, EEMA 1.46% MAG: AAXJ 3.63%, EEMA 4.09% GIU: AAXJ 0.10%, EEMA 0.75% LUG: AAXJ 4.75%, EEMA 4.90%
AGO: AAXJ 1.15%, EEMA 1.12% SET: AAXJ -0.29%, EEMA -0.17%   OTT: AAXJ 3.90%, EEMA 4.52% NOV: AAXJ -0.43%, EEMA -0.54%
DEC: AAXJ 1.53%, EEMA 1.43%

Soglia: 3%  Coppia 7:{AAXJ,ESGE}:   GEN: AAXJ 6.35%, ESGE 4.02% FEB: AAXJ 2.59%, ESGE 2.41% MAR: AAXJ 2.84%, ESGE 2.18%
APR: AAXJ 1.28%, ESGE 1.58% MAG: AAXJ 3.63%, ESGE 2.27% GIU: AAXJ 0.10%, ESGE -0.22%    LUG: AAXJ 4.75%, ESGE 5.38%
AGO: AAXJ 1.15%, ESGE 1.88% SET: AAXJ -0.29%, ESGE -0.23%   OTT: AAXJ 3.90%, ESGE 2.94% NOV: AAXJ -0.43%, ESGE 0.19%
DEC: AAXJ 1.53%, ESGE 2.93%

Soglia: 3%  Coppia 8:{AAXJ,FDT}:    GEN: AAXJ 6.35%, FDT 3.87%  FEB: AAXJ 2.59%, FDT 1.08%  MAR: AAXJ 2.84%, FDT 1.78%
APR: AAXJ 1.28%, FDT 2.55%  MAG: AAXJ 3.63%, FDT 2.84%  GIU: AAXJ 0.10%, FDT -0.88% LUG: AAXJ 4.75%, FDT 4.15%
AGO: AAXJ 1.15%, FDT 0.54%  SET: AAXJ -0.29%, FDT 1.82% OTT: AAXJ 3.90%, FDT 2.45%  NOV: AAXJ -0.43%, FDT 0.31%
DEC: AAXJ 1.53%, FDT 2.25%

Soglia: 3%  Coppia 9:{AAXJ,FPA}:    GEN: AAXJ 6.35%, FPA 5.19%  FEB: AAXJ 2.59%, FPA 3.47%  MAR: AAXJ 2.84%, FPA 1.93%
APR: AAXJ 1.28%, FPA -0.84% MAG: AAXJ 3.63%, FPA 5.89%  GIU: AAXJ 0.10%, FPA -0.85% LUG: AAXJ 4.75%, FPA 4.02%
AGO: AAXJ 1.15%, FPA -0.03% SET: AAXJ -0.29%, FPA -1.90%    OTT: AAXJ 3.90%, FPA 2.94%  NOV: AAXJ -0.43%, FPA -0.57%
DEC: AAXJ 1.53%, FPA 2.41%

Soglia: 3%  Coppia 10:{AAXJ,FPXI}:  GEN: AAXJ 6.35%, FPXI 5.17% FEB: AAXJ 2.59%, FPXI 1.22% MAR: AAXJ 2.84%, FPXI 4.20%
APR: AAXJ 1.28%, FPXI 3.12% MAG: AAXJ 3.63%, FPXI 4.40% GIU: AAXJ 0.10%, FPXI 0.10% LUG: AAXJ 4.75%, FPXI 6.19%
AGO: AAXJ 1.15%, FPXI 2.13% SET: AAXJ -0.29%, FPXI 0.35%    OTT: AAXJ 3.90%, FPXI 1.08% NOV: AAXJ -0.43%, FPXI -1.22%
DEC: AAXJ 1.53%, FPXI 1.70%
```

Tabelle e grafici tempi di esecuzione

Qui di seguito verranno presentati i grafici e le rispettive tabelle legate ai tempi di esecuzione dei vari Job in Locale e in Cluster, nonché con dimensioni crescenti dell'input che sono state etichettate con X1, X2 ed X3 (con X1 = database di partenza). I tempi forniti sono tutti calcolati in secondi e il Job 2 di MapReduce è ottenuto sommando i tempi di esecuzione dei Mapper e Reducer di Join con quelli specifici del Job 2.

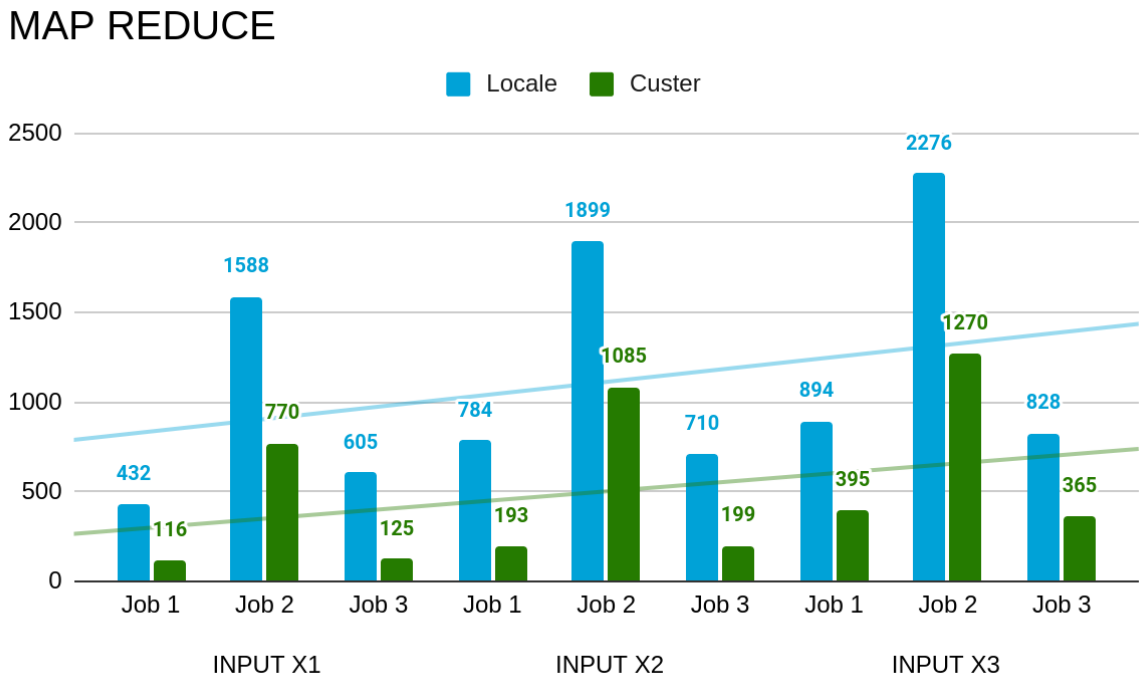
Tabelle:

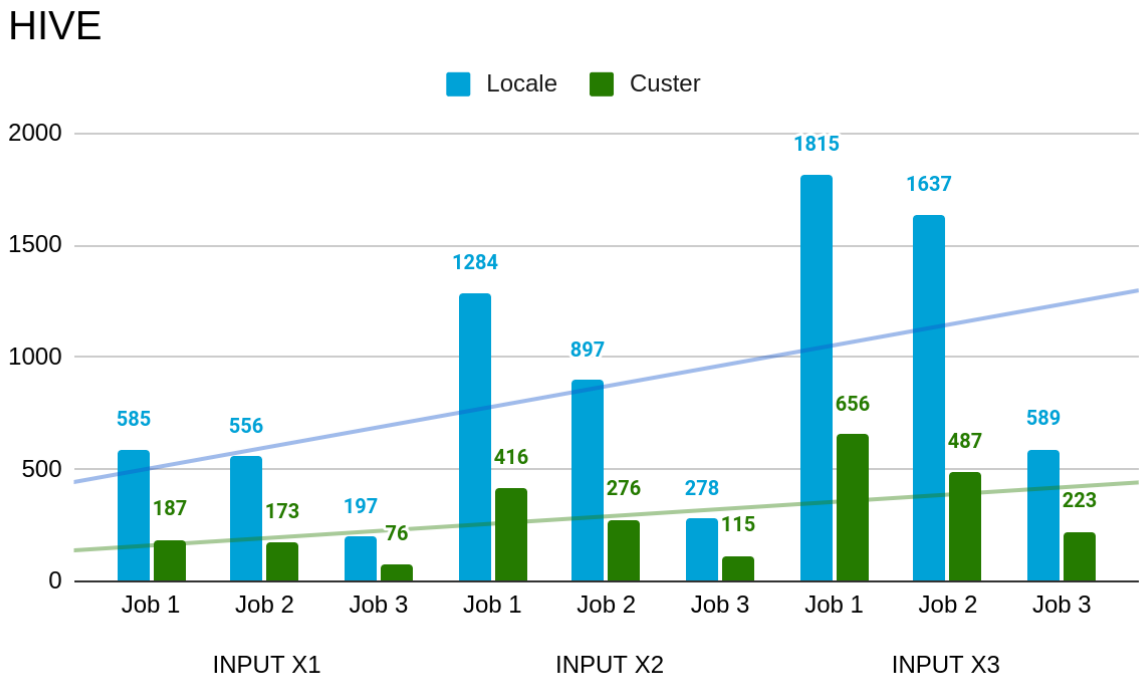
MAPREDUCE	*INPUT X1*		*INPUT X2*		*INPUT X3*	
	Locale	Custer	Locale	Custer	Locale	Custer
Job 1	432 sec	116 sec	784 sec	193 sec	894 sec	395 sec
Job 2	1588 sec	770 sec	1899 sec	1085 sec	2276 sec	1270 sec
Job 3	605 sec	125 sec	710 sec	199 sec	828 sec	365 sec

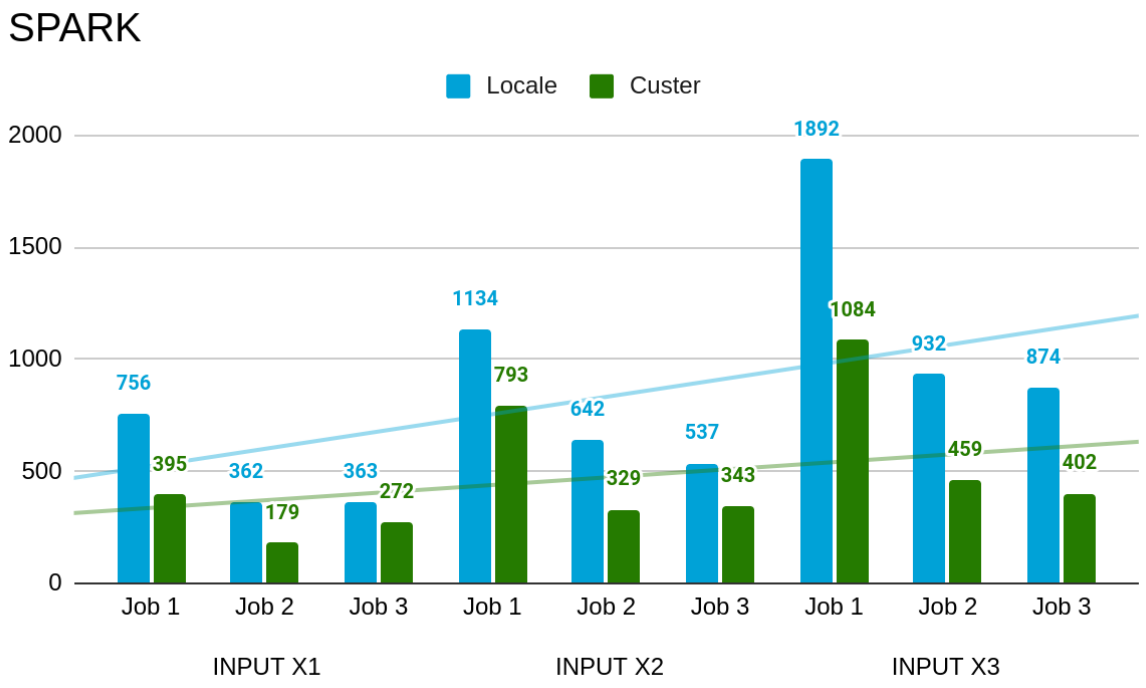
HIVE	*INPUT X1*		*INPUT X2*		*INPUT X3*	
	Locale	Custer	Locale	Custer	Locale	Custer
Job 1	585 sec	187 sec	1284 sec	416 sec	1815 sec	656 sec
Job 2	556 sec	173 sec	897 sec	276 sec	1637 sec	487 sec
Job 3	197 sec	76 sec	278 sec	115 sec	589 sec	223 sec

SPARK	*INPUT X1*		*INPUT X2*		*INPUT X3*	
	Locale	Custer	Locale	Custer	Locale	Custer
Job 1	756 sec	395 sec	1134 sec	793 sec	1892 sec	1084 sec
Job 2	362 sec	179 sec	642 sec	329 sec	932 sec	459 sec
Job 3	363 sec	272 sec	537 sec	343 sec	874 sec	402 sec

Grafici:


MAP REDUCE
Locale
Custer
2500
2000
1500
1000
500
0
432
116
1588
770
605
125
784
193
1899
1085
710
199
894
395
2276
1270
828
365
Job 1
Job 2
Job 3
Job 1
Job 2
Job 3
Job 1
Job 2
Job 3
INPUT X1
INPUT X2
INPUT X3


HIVE
Locale
Custer
2000
1500
1000
500
0
585
187
556
173
197
76
1284
416
897
276
278
115
1815
656
1637
487
589
223
Job 1
Job 2
Job 3
Job 1
Job 2
Job 3
Job 1
Job 2
Job 3
INPUT X1
INPUT X2
INPUT X3



Codice completo MapReduce e Spark

Il codice completo di MapReduce e Spark sviluppato in python è in allegato a questo documento o raggiungibile attraverso la repository di GitHub:

https://github.com/EmanueleSpezia/PrimoProgetto_BigData2021