Rapporto finale Big Data Primo Progetto 2021

Omar Moh'd(510448) - Jomin Neelikatt(507937)

May 21, 2021

1 JOB 1

L'obiettivo del job n°1 è quello di generare un report che per ogni azione, restituisca la data della prima quotazione, la data dell'ultima quotazione, la variazione percentuale della quotazione, il prezzo massimo e quello minimo.

1.1 MAP REDUCE

Nel file mapper si stampano solamente le righe. Successivamente, nel reducer si crea un dizionario annidato che per ogni azione contiene date e valori. Ogni riga va, in caso, aggiunta oopure usata come aggiornamento nel dizionario.
Come ultima operazione si calcola la variazione percentuale del valore dell'azione fra data iniziale e data finale, per quell'azione.

```
class mapper:

    for line in csv:
        print(line)

class reducer:

    stocks={}

    for line in lines:
        if line in stocks:
            stocks.update(line)
        else:
            stocks.add(line)

    for stock in stocks:
        stocks.calculate_variation(stock)

    return stocks
```

1.2 HIVE

Inizialmente viene caricato il file sorgente in una tabella. Successivamente vengono create due tabelle('stock_min_date', 'stock_max_date') con lo scopo di immagazzinare per ogni azione, data massima e minima, con relativi valori.

Dopo di che, tramite un'operazione di join fra le ultime due tabelle create si ottiene, per ogni azione, la variazione percentuale di valore fra le date minime e massime.
Viene creata un'altra tabella con l'obiettivo di racchiudere, per ogni azione, le restanti informazioni richieste(data minima, data massima, valore minimo, valore massimo).
Infine, si crea un join fra le ultime due tabelle per completare i requisiti, ossia, per ogni azione si ottengono data minima, data massima, valore minimo, valore massimo e variazione fra data minima e massima.

```
DROP TABLE hsp;
DROP TABLE stock_min_date;
DROP TABLE stock_max_date;
DROP TABLE stock_min_max_variation;
DROP TABLE stocks_min_max;

-- tabella iniziale
CREATE TABLE hsp (nome STRING, apertura FLOAT, chiusura FLOAT,
chiusura_2 FLOAT, minimo FLOAT, massimo FLOAT, volume INTEGER,
data DATE)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';

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

-- data massima per ogni azione
CREATE TABLE stock_max_date AS
SELECT DISTINCT a.nome, a.data, a.chiusura
FROM hsp a
INNER JOIN (
    SELECT nome, max(data) as data
    FROM hsp
    GROUP BY nome
) b
ON (a.nome = b.nome
```

```
    AND a.data = b.data);

-- data minima per ogni azione
CREATE TABLE stock_min_date AS
SELECT DISTINCT a.nome, a.data, a.chiusura
FROM hsp a
INNER JOIN (
    SELECT nome, min(data) as data
    FROM hsp
    GROUP BY nome
) b
ON (a.nome = b.nome
    AND a.data = b.data);

-- variazione fra le date massime/minime
CREATE TABLE stock_min_max_variation AS
SELECT a.nome,100*((b.chiusura - a.chiusura) / a.chiusura) as variation
FROM stock_min_date a, stock_max_date b
WHERE a.nome=b.nome;

-- valori a date estreme
CREATE TABLE stocks_min_max as
SELECT a.nome, max(a.data) as data_max, min(a.data) as data_min,
max(a.massimo) as max_value, min(a.minimo) as min_value
FROM hsp a
GROUP BY a.nome
ORDER BY a.nome;

--output
SELECT a.nome, a.data_max, a.data_min, b.variation,
a.max_value, a.min_value
FROM stocks_min_max a, stock_min_max_variation b
WHERE a.nome=b.nome
GROUP BY a.nome, data_max, data_min, variation,
a.max_value, a.min_value
LIMIT 10;
```

1.3 SPARK

Viene caricato l'input historical_stock_prices.csv in un RDD chiamato hsp. Per ogni ticker vengono calcolate(in due RDD separati) la data minima e massima e il rispettivo valore di close con due reduceByKey. Il valore di close minimo e massimo è poi utilizzato per il calcolo della variazione percentuale. Il massimo e minimo totali per ogni ticker sono ricavati sempre grazie all'utilizzo di reduceByKey. Infine i quattro RDD usati per contenere i quattro output vengono joinati e ordinati.

```
hsp = RDD che contiene il dataset historical_stock_prices.csv

min_date_close = hsp \
        .map(ticker, (date, close))) \
        .reduceByKey(get_min(date))

max_date_close = hsp \
        .map(ticker, (date, close)) \
        .reduceByKey(get_max(date))

variazione_percentuale = min_date_close.join(max_date_close) \
        .map(ticker, (close_max-close_min)/close_min * 100)

max_high = hsp \
        .map(ticker, (high))\
        .reduceByKey(get_max(high)) \

min_low = hsp \
        .map(ticker, (low))\
        .reduceByKey(get_min(low)) \

output = min_date_close.join(max_date_close) \
        .join(variazione_percentuale) \
        .join(max_high) \
        .join(min_low) \
        .map([ticker, min_date, max_date, percent_change, max_high, min_low]) \
        .sortBy(max_date, Descending)
```

2 JOB 2

L'obiettivo del job n°2 è quello di generare un report che per ciascun settore e per ciascun anno del periodo dal 2009 al 2018 restituisca la variazione percentuale della quotazione del settore nell'anno, l'azione del settore che ha avuto il maggior incremento percentuale nell'anno(con indicazione dell'incremento) e l'azione del settore che ha avuto il maggior volume di transazioni nell'anno(con indicazione del volume).

2.1 MAP REDUCE

Nel mapper viene letto il file historical_stocks.csv dalla Distributed Cache da joinare con il file historical_stock_prices.csv(letto da input) per poter risalire ai settori associati alle azioni. Questa associazione viene effetuata con il dizionario tickerToSectorMap. La riga viene stampata solo se l'anno è compreso tra il 2009 e il 2018.

```
class mapper:
    setup():
        read historical_stocks.csv from the Distributed Cache
        tickerToSectorMap = map which associate a (non null) sector
                            for each ticker in historical_stocks.csv
    map(key, line):
        ticker,_,close,_,_,_,volume,date = line
        year = getYear(date)
        if year in range from 2009 to 2018 and
        ticker has a corresponding sector in tickerToSectorMap:
            sector = ticker's corresponding sector
            key = sector, ticker, date
            value = close, volume
            print(key, value)
```

Nel reducer vengono inizializzati tre dizionari(uno per richiesta). Per ogni riga di input vengono confrontati i settori(corrente e precedente) e i ticker(corrente e precedente), e vengono di conseguenza aggiornati i dizionari con i valori di chiusura iniziale o finale, o con il valore di volume. Prima di stampare la riga vengono calcolati i relativi valori di output a partire dai dizionari.

```
class reducer:
    reduce(sector, lines):
```

```
yearToChange = empty nested map
yearToTickerVolume = empty nested map
yearToTickerChange = empty double nested map
for line in lines:
    year = get year value from the day variable
    if the current ticker has changed from the previous line:
        yearToChange[prev_year]['close_final'] += prev_close
        yearToTickerChange[prev_year][prev_ticker]['close_final'] += prev_close
        yearToChange[curr_year]['close_start'] += curr_close
        yearToTickerChange[curr_year][curr_ticker]['close_start'] += curr_close
        yearToTickerVolume[curr_year][curr_ticker] += curr_volume
    for each year in yearToChange:
        totDifference = yearToChange[year]['close_final']-
                        yearToChange[year]['close_start']
        totPercentChange = totDifference / yearToChange[year]['close_start'] * 100
        tickerChangeMap = yearToTickerChange[year]
        maxTickerForChange, maxChange = getMaxTickerChange(tickerChangeMap)
        tickerTrend = yearToTickerVolume[year]
        maxTickerForVolume, maxVolume = getMaxTickerVolume(tickerTrend)
        print(sector, year, totPercentChange, maxTickerForChange,
        maxChange, maxTickerForVolume, maxVolume)
```

2.2 HIVE

Inizialmente si creano le tabelle che conterranno gli input caricati dal path locale. Viene effettuato tra loro il join filtrando l'anno e il settore. Si calcola il peso dalla tabella di join come somma dei volumi relativi agli stessi ticker, viene, poi per ogni settore e anno, preso quello maggiore con un'altra tabella. Le variazioni percentuali(totale e azione maggiore) sono calcolate specularmente a partire dalla tabella di data minima e massima, prendendo poi con altre tabelle il valore di chiusura di partenza e quello finale per il calcolo dei due output.

```
--creo le due tabelle di input
CREATE TABLE prices(ticker STRING,open DOUBLE, close DOUBLE,
adj DOUBLE, low DOUBLE, high DOUBLE, volume DOUBLE, data DATE)
row format delimited fields terminated by ',';
```

```
CREATE TABLE stocks(ticker STRING, exch STRING, name STRING,
sector STRING, industry STRING)ROW FORMAT SERDE
'org.apache.hadoop.hive.serde2.OpenCSVSerde';

-- load the CSV (original and modified)
LOAD DATA LOCAL INPATH 'historical_stock_prices.csv' OVERWRITE INTO TABLE prices;
LOAD DATA LOCAL INPATH 'historical_stocks.csv' OVERWRITE INTO TABLE stocks;

CREATE TABLE joinTable as
SELECT
    prices.ticker,
    prices.close,
    prices.volume,
    prices.data,
    stocks.sector
FROM prices JOIN stocks
ON prices.ticker = stocks.ticker
WHERE YEAR(prices.data)>='2009' AND YEAR(prices.data)<='2018'
AND stocks.sector!='N/A';

--per ogni settore, per ogni anno, per ogni ticker ho il loro peso totale
CREATE TABLE sumOfTickerVolume AS
SELECT
    sector,
    YEAR(data) AS anno,
    ticker,
    SUM(volume) AS totTickerVolume
FROM joinTable
GROUP BY sector, YEAR(data), ticker;

--prendo solo il ticker con il peso maggiore
CREATE TABLE maxTickerVolume AS
SELECT m.*
FROM sumOfTickerVolume m
    LEFT JOIN sumOfTickerVolume b
        ON m.sector = b.sector AND m.anno = b.anno
        AND m.totTickerVolume < b.totTickerVolume
```

```
WHERE b.totTickerVolume IS NULL;

--per ogni anno, per ogni settore, per ogni ticker ho la data minima e massima
CREATE TABLE dateMinMax AS
SELECT
    sector,
    ticker,
    min(TO_DATE(data)) AS min_data,
    max(TO_DATE(data)) AS max_data
FROM joinTable
GROUP BY sector, ticker, YEAR(data);

--per ogni ogni anno, per ogni settore ho prezzo di chiusura minimo
CREATE TABLE minClose AS
SELECT
    b.sector,
    YEAR(b.min_data) AS anno,
    SUM(a.close) AS min_close
FROM joinTable AS a, dateMinMax AS b
WHERE a.sector=b.sector AND a.data=b.min_data AND b.ticker=a.ticker
GROUP BY b.sector, YEAR(b.min_data);

--per ogni ogni anno, per ogni settore, per ogni ticker ho prezzo di chiusura minimo
CREATE TABLE minCloseTicker AS
SELECT
    b.sector,
    YEAR(b.min_data) AS anno,
    b.ticker,
    SUM(a.close) AS min_close
FROM joinTable AS a, dateMinMax AS b
WHERE a.sector=b.sector AND a.data=b.min_data AND b.ticker=a.ticker
GROUP BY b.sector, YEAR(b.min_data), b.ticker;

--per ogni ogni anno, per ogni settore ho prezzo di chiusura massimo
CREATE TABLE maxClose AS
SELECT
    b.sector,
```

```
    YEAR(b.max_data) AS anno,
    SUM(a.close) AS max_close
FROM joinTable AS a, dateMinMax AS b
WHERE a.sector=b.sector AND a.data=b.max_data AND b.ticker=a.ticker
GROUP BY b.sector, YEAR(b.max_data);

--per ogni ogni anno, per ogni settore, per ogni ticker ho prezzo di chiusura massimo
CREATE TABLE maxCloseTicker AS
SELECT
    b.sector,
    YEAR(b.max_data) AS anno,
    b.ticker,
    SUM(a.close) AS max_close
FROM joinTable AS a, dateMinMax AS b
WHERE a.sector=b.sector AND a.data=b.max_data AND b.ticker=a.ticker
GROUP BY b.sector, YEAR(b.max_data), b.ticker;

CREATE TABLE percentChange AS
SELECT
    min.sector,
    min.anno,
    ROUND(((max.max_close-min.min_close)/min.min_close) * 100, 2) AS variazioneAnnuale
FROM minClose AS min, maxClose AS max
WHERE min.sector=max.sector AND min.anno=max.anno
ORDER BY sector, anno;

--per ogni anno, per ogni settore, per ogni ticker ho la variazione percentuale
CREATE TABLE percentChangeTicker AS
SELECT
    min.sector,
    min.anno,
    min.ticker,
    ROUND(((max.max_close-min.min_close)/min.min_close) * 100, 2) AS variazioneAnnualeTicker
FROM minCloseTicker AS min, maxCloseTicker AS max
WHERE min.sector=max.sector AND min.anno=max.anno AND min.ticker=max.ticker
ORDER BY sector, anno;
```

```sql
--prendo solo il ticker con la variazione maggiore
CREATE TABLE maxTickerPercentChange AS
SELECT m.*
FROM percentChangeTicker m
    LEFT JOIN percentChangeTicker b
        ON m.sector = b.sector AND m.anno = b.anno
        AND m.variazioneAnnualeTicker < b.variazioneAnnualeTicker
WHERE b.variazioneAnnualeTicker IS NULL;

--query finale
SELECT
    a.sector,
    a.anno,
    a.variazioneAnnuale,
    b.ticker AS tickerVariazioneAnnualeMax,
    b.variazioneAnnualeTicker,
    c.ticker AS tickerVolumeMax,
    c.totTickerVolume
FROM percentChange AS a, maxTickerPercentChange AS b, maxTickerVolume AS c
WHERE a.sector=b.sector AND b.sector=c.sector AND a.anno=b.anno AND c.anno=b.anno
ORDER BY sector, anno;
```

2.3 SPARK

Vengono caricati i due input, filtrandoli per anno e settore, nei due RDD hsp e hs. Dopo il join si calcola il volume massimo e il relativo ticker con l'utilizzo di due reduceByKey uno per sommare i volumi e uno per prendere il maggiore. Per le variazioni percentuali(totale e ticker) viene usato un procedimento simile, calcolando prima le date minime e massime, poi i valori di close minimi e massimi, fino ad arrivare al calcolo delle varizioni. Infine i tre RDD usati per contenere i tre output vengono joinati.

```
hsp = RDD che contiene il dataset historical_stock_prices.csv
    .filter(2009 <= anno <= 2018)

hs = RDD che contiene il dataset historical_stocks.csv
    .filter(settore not null)
```

```
join_object = hsp.join(hs) \
    .map(ticker, close, volume, date, settore) \
    .persist(StorageLevel.MEMORY_AND_DISK)

volume = join_object \
        .map((settore, anno ticker), volume) \
        .reduceByKey(volume_x + volume_y) \
        .reduceByKey(get_max(volume_x, volume_y))

data_close_min_ticker = join_object\
    .map((ticker, settore, anno), (close, date))\
    .reduceByKey(get_min(close_x, close_y))

data_close_min_tot = data_close_min_ticker \
        .map((sector, anno), close) \
        .reduceByKey(close_x + close_y)

data_close_max_ticker = join_object\
    .map((ticker, settore, anno), (close, date))\
    .reduceByKey(get_max(close_x, close_y))

data_close_max_tot = data_close_max_ticker \
        .map((sector, anno), close) \
        .reduceByKey(close_x + close_y)

variazione_percentuale_ticker = data_close_min_ticker.join(data_close_max_ticker)
.map(lambda linea: ((settore, anno), (ticker, variazione_percentuale))
variazione_percentuale_tot = data_close_min_tot.join(data_close_max_tot)
.map(settore, anno, variazione_percentuale_tot)

output = variazione_percentuale_tot \
                .join(volume) \
                .join(variazione_percentuale_ticker
                .reduceByKey(get_max(variazione_x, variazione_y)))
                .map(settore, anno, variazione_percentuale_tot,
                          (ticker, variazione_max), (ticker, volume_max)) \
                .sortBy(settore)
```

3 JOB 3

L'obiettivo del Job n° 3 era quello di estrapolare coppie di aziende che, nell'anno 2017, hanno avuto una variazione percentuale del valore mensile simile, nel nostro caso è stata applicata una soglia del 0,5%.

3.1 MAP REDUCE

Per ogni riga del file, opportunamente pulito e filtrato vengono estratte, per tutte le righe relative al 2017, nome dell'azione, data e valore di chiusura.
Ogni riga estratta viene data ad una funzione che ne controlla i valori e in caso, aggiorna data e valore per quell'azione all'interno di un dizionario annidato(per ogni azione corrispondono dodici dizionari che si riferiscono al mese, ognuno dei quali racchiude date iniziali e finali per quel mese con relativi valori).
Dopo aver completato la lettura del file sorgente, si crea una nuova struttura dati che contiene azione, mese e relativa variazione percentuale.
Questa struttura dati viene filtrata, eliminando tutte quelle azioni che non contengono dodici variazioni, ossia che non hanno quindi dati completi per almeno un mese. Per quanto riguarda il reducer, si confrontano le varie azioni (evitando il confronto fra la stessa azione) e mese per mese, si osserva se la differenza fra le variazioni delle due azioni rispetta la soglia prestabilita.
Infine, vengono stampate tutte le coppie(con relativo mese e variazioni) che per ogni mese dell'anno 2017 hanno avuto una variazione percentuale del loro valore simile.
Di seguito sono allegati gli pseudo codici delle parti Mapper e Reducer - nello zip allegato, sono presenti i codici per esteso.

```
class mapper:

    all_stocks = {}
    for line in csv:
        if line.date.year == 2017:
            if line in all_stocks:
                all_stocks.update(line)
            else
                all_stocks.add(line)
```

```
    monthly_variations = {}

    for stock in all_stocks:
        monthly_variations.add(calculate_monthly_variation(stock))

    delete stock in monthly_variations with len(stock) < 12


class reducer:

    stock_couples={}
    months=['1', '2', '3', .., '11', '12']

    for stock in monthly_variations:
        for stock2 in monthly_variations:
            if stock != stock2:
                for month in months:
                    if (-0.5 < stock[month] - stock2[month] < 0.5):
                        stock_couples.add(stock, stock2,
                                          month, stock1[variation],
                                          stock2[variation])

    return stock_couples
```

3.2 HIVE

Per quanto riguarda Hive, viene caricato il file in una tabella apposita, successivamente si crea una tabella che contiene solamente le ennuple riguardanti l'anno 2017.
Vengono create due tabelle('stock_2017_min_date' e 'stock_2017_max_date') che racchiudono per ogni azione, per ogni mese, rispettivamente il giorno massimo e minimo documentati, con relativi valori.
A partire da queste ultime due tabelle viene creata una tabella con la quale si ricavano, per ogni azione, le variazioni mensili in termini di percentuale. Si crea una tabella di coppie fra azioni che per ogni mese hanno avuto una variazione di 0,5%. Successivamente la tabella viene filtrata per coppie che presentano variazioni per tutti e 12 i mesi dell'anno 2017. Tramite una SELECT vengono

presentati i risultati.

```
DROP TABLE hsp;
DROP TABLE stock_2017;
DROP TABLE stock_2017_max_date;
DROP TABLE stock_2017_min_date;
DROP TABLE stock_2017_monthly_variations;
DROP TABLE stock_couples;
DROP TABLE stock_couples_filtered;

-- tabella iniziale
CREATE TABLE hsp (nome STRING, apertura FLOAT, chiusura FLOAT,
chiusura_2 FLOAT, minimo FLOAT, massimo FLOAT, volume INTEGER, data DATE)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';

LOAD DATA INPATH '/input/historical_stock_prices_triple.csv'
OVERWRITE INTO TABLE hsp;

-- prendo solamente colonne d'interesse
CREATE TABLE stock_2017 AS
SELECT hsp.nome, hsp.chiusura,
MONTH(hsp.data) as mese, DAY(hsp.data) as giorno
FROM hsp
WHERE YEAR(hsp.data) = 2017;

-- data massima per ogni azione, per ogni mese
CREATE TABLE stock_2017_max_date AS
SELECT DISTINCT a.nome, a.mese, a.giorno, a.chiusura
FROM stock_2017 a
INNER JOIN (
    SELECT max(giorno) as giorno, mese,
        nome
    FROM stock_2017
    GROUP BY nome, mese
) b
ON (a.nome = b.nome
```

```
    AND a.giorno = b.giorno
AND b.mese = a.mese);

-- data minima per ogni azione, per ogni mese
CREATE TABLE stock_2017_min_date AS
SELECT DISTINCT a.nome, a.mese, a.giorno, a.chiusura
FROM stock_2017 a
INNER JOIN (
    SELECT min(giorno) as giorno, mese,
        nome
    FROM stock_2017
    GROUP BY nome, mese
) b
ON (a.nome = b.nome
    AND a.giorno = b.giorno
AND b.mese = a.mese);

-- variazioni mensili per ogni azione
CREATE TABLE stock_2017_monthly_variations AS
SELECT a.nome, a.mese,
((b.chiusura-a.chiusura)*100)/a.chiusura as variation
FROM stock_2017_min_date a, stock_2017_max_date b
WHERE a.nome = b.nome AND a.mese=b.mese;

-- coppie d'azioni
CREATE TABLE stock_couples as
SELECT a.nome, b.nome as nome2, a.mese,
a.variation as var, b.variation as var2
FROM stock_2017_monthly_variations a inner join
     stock_2017_monthly_variations b
     WHERE a.mese=b.mese
     AND a.nome<>b.nome
     AND (a.variation-b.variation)
     BETWEEN -0.5 AND 0.5
     ORDER by a.nome, nome2, a.mese;

-- filtro per azioni che hanno 12 variazioni
```

```sql
CREATE TABLE stock_couples_filtered as
SELECT a.nome, a.nome2, count(a.mese) as tot_mesi
FROM stock_couples a
GROUP BY a.nome, a.nome2
HAVING count(a.mese)=12;

--output finale
SELECT A.*
FROM stock_couples A
    JOIN stock_couples_filtered B ON
        ((A.nome = B.nome) AND (A.nome2=B.nome2))
ORDER BY a.nome, a.nome2, a.mese
LIMIT 120;
```

3.3 SPARK

Inizialmente viene caricato il file escludendo la prima riga. Si applica un filtro sull'anno 2017 e si salvano solo le possibili date iniziali e finali per ogni mese.
Si effettua un map creando una struttura (azione, mese) → (giorno, valore). Partendo da questa struttura si raggruppa per campi azione/mese e si calcola la variazione percentuale per ogni azione tramite le apposite date, ottenendo una struttura dati del tipo (azione) → (mese, variazione).
Si raggruppa per mese così da ottenere, per ogni azione le sue dodici variazioni mensili. Effettuo un filtro tenendo solamente le azioni con dodici variazioni, ed infine tramite un join(di python, non SQL) ottengo le coppie di azioni con relative variazioni.
Infine applico un filter per ottenere solamente le coppie che hanno variazioni che non superino 0,5%.
Di seguito è allegato lo pseudocodice - nello zip allegato, è presente il codice per esteso.

```
data = csv.map((stock, year), (month, day, value)) \
    .filter(year == '2017' and (day == '01' or day == '28' or day == '29' \
                                            or day == '30' or day == '31')) \
    .map(((stock, mese), (giorno,valore)) \
    .groupByKey(stock, mese) \
    .map((azione), (mese, variazione)) \
    .groupByKey(azione) \
```

```
    .filter(len(mese) == 12) \
    .map((azione), (sorted(mese, variazione))) \
    .map((1), (azione, (mese,variazione)) \
    .filter(lambda x: print(x))

result = data.join(data)
         .filter(-0.6 < variazione < 0.6)
```

4 Output, Tempistiche e Considerazioni

In questo paragrafo verranno descritti i risultati dei job, considerazioni, possibili miglioramenti e grafici sulle prestazioni.
I vari script sono stati eseguiti su quattro tipi di file, a partire dal file originale('historical_stock_prices.csv'):

1/20	Un ventesimo del file originale	100MB
1/2	Metà del file originale	1GB
Normale	File originale	2GB
x3	File originale triplicato	6GB

Per quanto riguarda le esecuzioni in locale, il computer utilizzato ha una CPU intel i3-7100 2.4Ghz con SSD da 256 GB, su una partizione Linux da circa 50GB.

4.1 Job 1

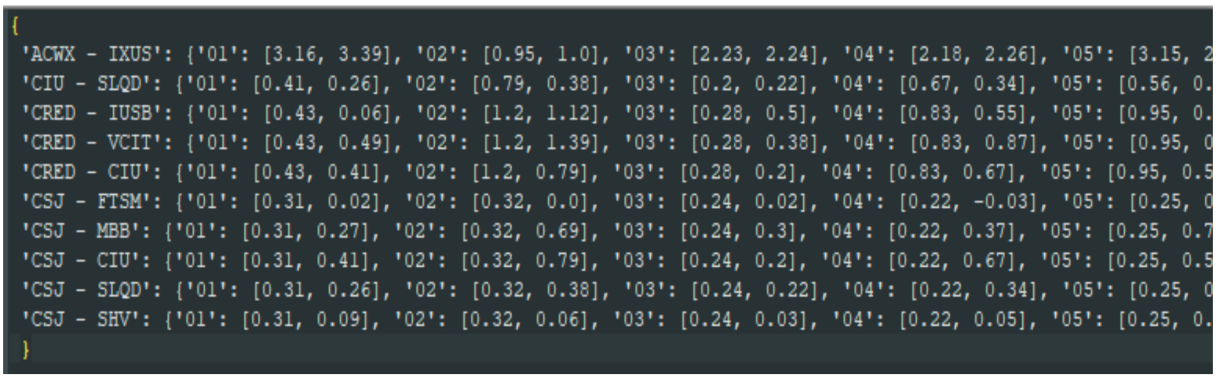
Di seguito i risultati del primo job:

```
A       2018-08-24      1999-11-18      109.63646513864813      115.87983       7.51073
AA      2018-08-24      1970-01-02      508.32539151728577      117.19431       3.6045
AABA    2018-08-24      1996-04-12      4910.909201882102       125.03125       0.6458333
AAC     2018-08-24      2018-01-16      4.8565072848188215      12.96   7.79
AAL     2018-08-24      2005-09-27      101.13990274300461      63.27   1.45
AAME    2018-08-24      1980-03-17      -29.87012689442144      15.8    0.375
AAN     2018-08-24      1987-01-20      4683.263038754136       51.53   0.4814815
AAOI    2018-08-24      2013-09-26      330.42168394946026      103.41  8.08
AAON    2018-08-24      1992-12-16      41348.204641503355      43.3    0.089770794
AAP     2018-08-24      2001-11-29      1084.149816187522       201.24  12.33
```

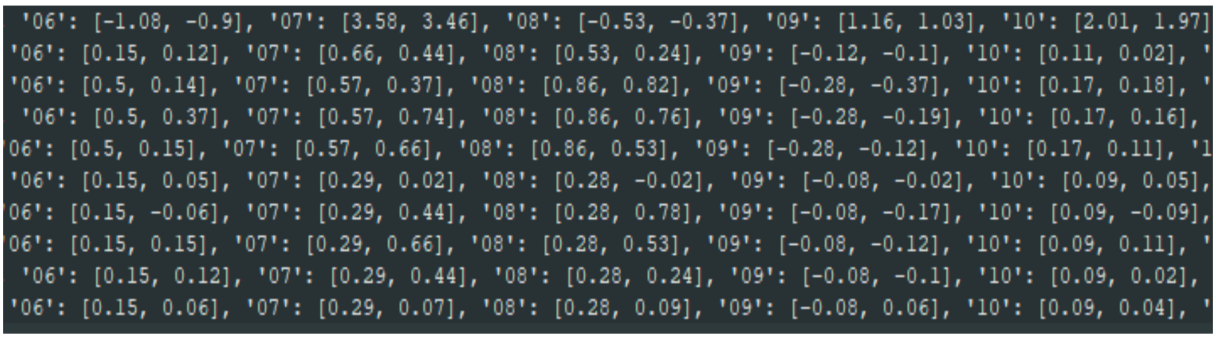
Di seguito i risultati del secondo job:

```
['BASIC INDUSTRIES', 2011, -58.6, ('ROAD', 188.7), ('FCX', 5150807800.0)]
['BASIC INDUSTRIES', 2012, -68.79, ('PATK', 261.86), ('VALE', 4659766700.0)]
['BASIC INDUSTRIES', 2009, 3.48, ('GURE', 709.72), ('FCX', 9141685400.0)]
['BASIC INDUSTRIES', 2010, 21.79, ('BLD', 519.8), ('FCX', 6891808600.0)]
['BASIC INDUSTRIES', 2015, -48.1, ('SUM', 35191.63), ('FCX', 7286761300.0)]
['BASIC INDUSTRIES', 2017, 15.28, ('OPNT', 310.18), ('VALE', 7023267600.0)]
['BASIC INDUSTRIES', 2018, -3.08, ('XRM', 213.82), ('VALE', 3710091900.0)]
['BASIC INDUSTRIES', 2013, 10.32, ('XRM', 416.93), ('VALE', 4428233700.0)]
['BASIC INDUSTRIES', 2014, -71.9, ('BLD', 884.6), ('VALE', 5660183200.0)]
['BASIC INDUSTRIES', 2016, 13.83, ('TECK', 451.79), ('FCX', 10464699500.0)]
```

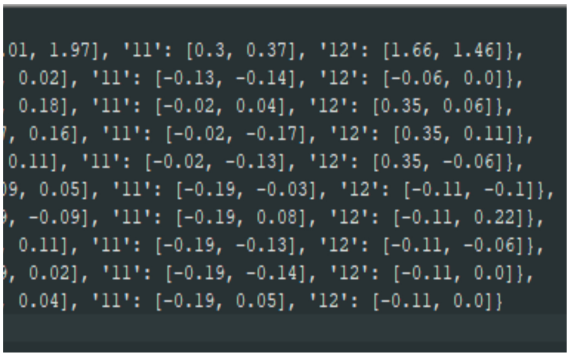
Di seguito i risultati del terzo job:



```
{
 'ACWX - IXUS': {'01': [3.16, 3.39], '02': [0.95, 1.0], '03': [2.23, 2.24], '04': [2.18, 2.26], '05': [3.15, 2
 'CIU - SLQD': {'01': [0.41, 0.26], '02': [0.79, 0.38], '03': [0.2, 0.22], '04': [0.67, 0.34], '05': [0.56, 0.
 'CRED - IUSB': {'01': [0.43, 0.06], '02': [1.2, 1.12], '03': [0.28, 0.5], '04': [0.83, 0.55], '05': [0.95, 0.
 'CRED - VCIT': {'01': [0.43, 0.49], '02': [1.2, 1.39], '03': [0.28, 0.38], '04': [0.83, 0.87], '05': [0.95, 0
 'CRED - CIU': {'01': [0.43, 0.41], '02': [1.2, 0.79], '03': [0.28, 0.2], '04': [0.83, 0.67], '05': [0.95, 0.5
 'CSJ - FTSM': {'01': [0.31, 0.02], '02': [0.32, 0.0], '03': [0.24, 0.02], '04': [0.22, -0.03], '05': [0.25, 0
 'CSJ - MBB': {'01': [0.31, 0.27], '02': [0.32, 0.69], '03': [0.24, 0.3], '04': [0.22, 0.37], '05': [0.25, 0.7
 'CSJ - CIU': {'01': [0.31, 0.41], '02': [0.32, 0.79], '03': [0.24, 0.2], '04': [0.22, 0.67], '05': [0.25, 0.5
 'CSJ - SLQD': {'01': [0.31, 0.26], '02': [0.32, 0.38], '03': [0.24, 0.22], '04': [0.22, 0.34], '05': [0.25, 0
 'CSJ - SHV': {'01': [0.31, 0.09], '02': [0.32, 0.06], '03': [0.24, 0.03], '04': [0.22, 0.05], '05': [0.25, 0.
}
```

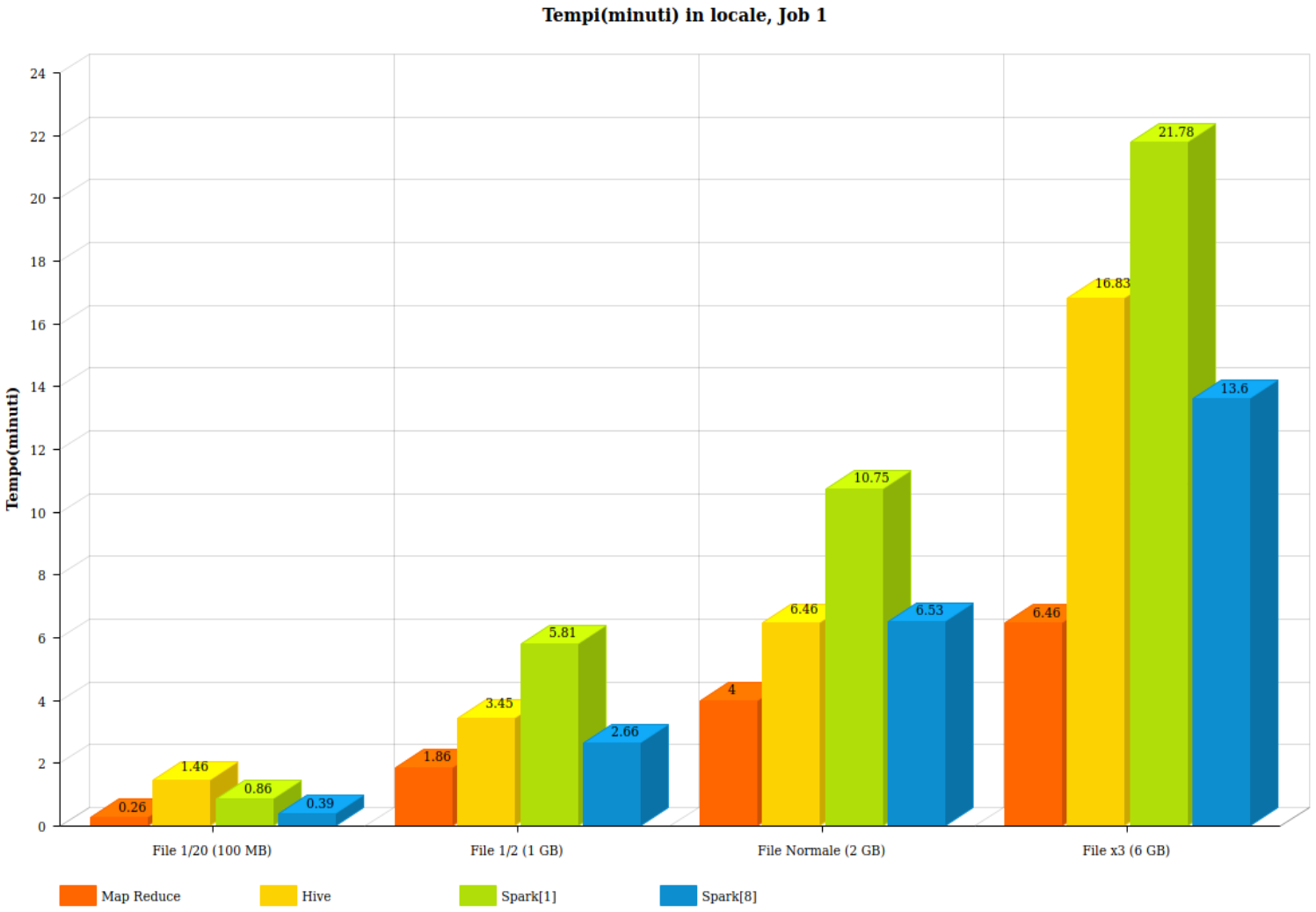



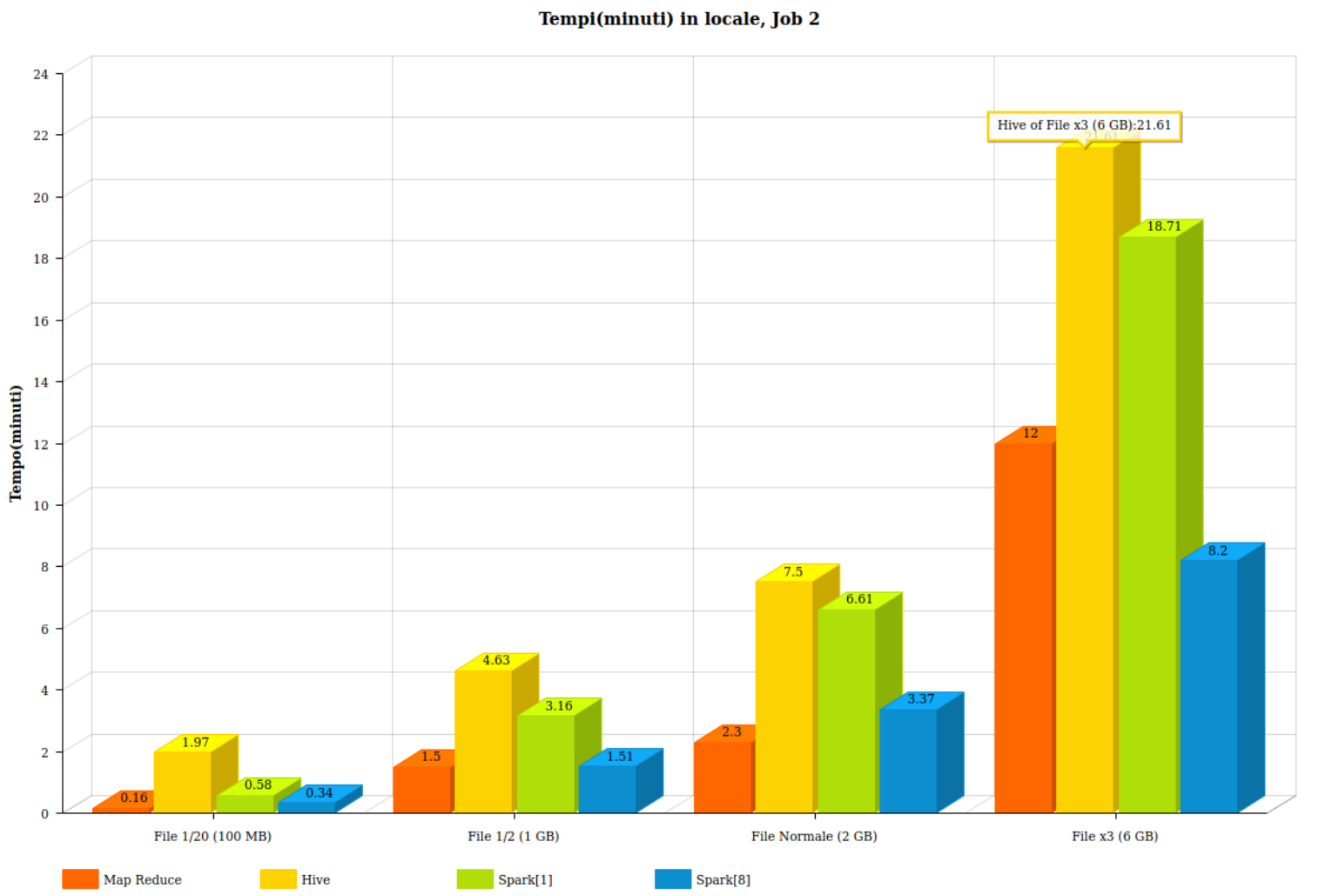
```
 '06': [-1.08, -0.9], '07': [3.58, 3.46], '08': [-0.53, -0.37], '09': [1.16, 1.03], '10': [2.01, 1.97]
'06': [0.15, 0.12], '07': [0.66, 0.44], '08': [0.53, 0.24], '09': [-0.12, -0.1], '10': [0.11, 0.02], '
'06': [0.5, 0.14], '07': [0.57, 0.37], '08': [0.86, 0.82], '09': [-0.28, -0.37], '10': [0.17, 0.18], '
 '06': [0.5, 0.37], '07': [0.57, 0.74], '08': [0.86, 0.76], '09': [-0.28, -0.19], '10': [0.17, 0.16],
06': [0.5, 0.15], '07': [0.57, 0.66], '08': [0.86, 0.53], '09': [-0.28, -0.12], '10': [0.17, 0.11], '1
'06': [0.15, 0.05], '07': [0.29, 0.02], '08': [0.28, -0.02], '09': [-0.08, -0.02], '10': [0.09, 0.05],
06': [0.15, -0.06], '07': [0.29, 0.44], '08': [0.28, 0.78], '09': [-0.08, -0.17], '10': [0.09, -0.09],
06': [0.15, 0.15], '07': [0.29, 0.66], '08': [0.28, 0.53], '09': [-0.08, -0.12], '10': [0.09, 0.11], '
 '06': [0.15, 0.12], '07': [0.29, 0.44], '08': [0.28, 0.24], '09': [-0.08, -0.1], '10': [0.09, 0.02],
'06': [0.15, 0.06], '07': [0.29, 0.07], '08': [0.28, 0.09], '09': [-0.08, 0.06], '10': [0.09, 0.04], '
```

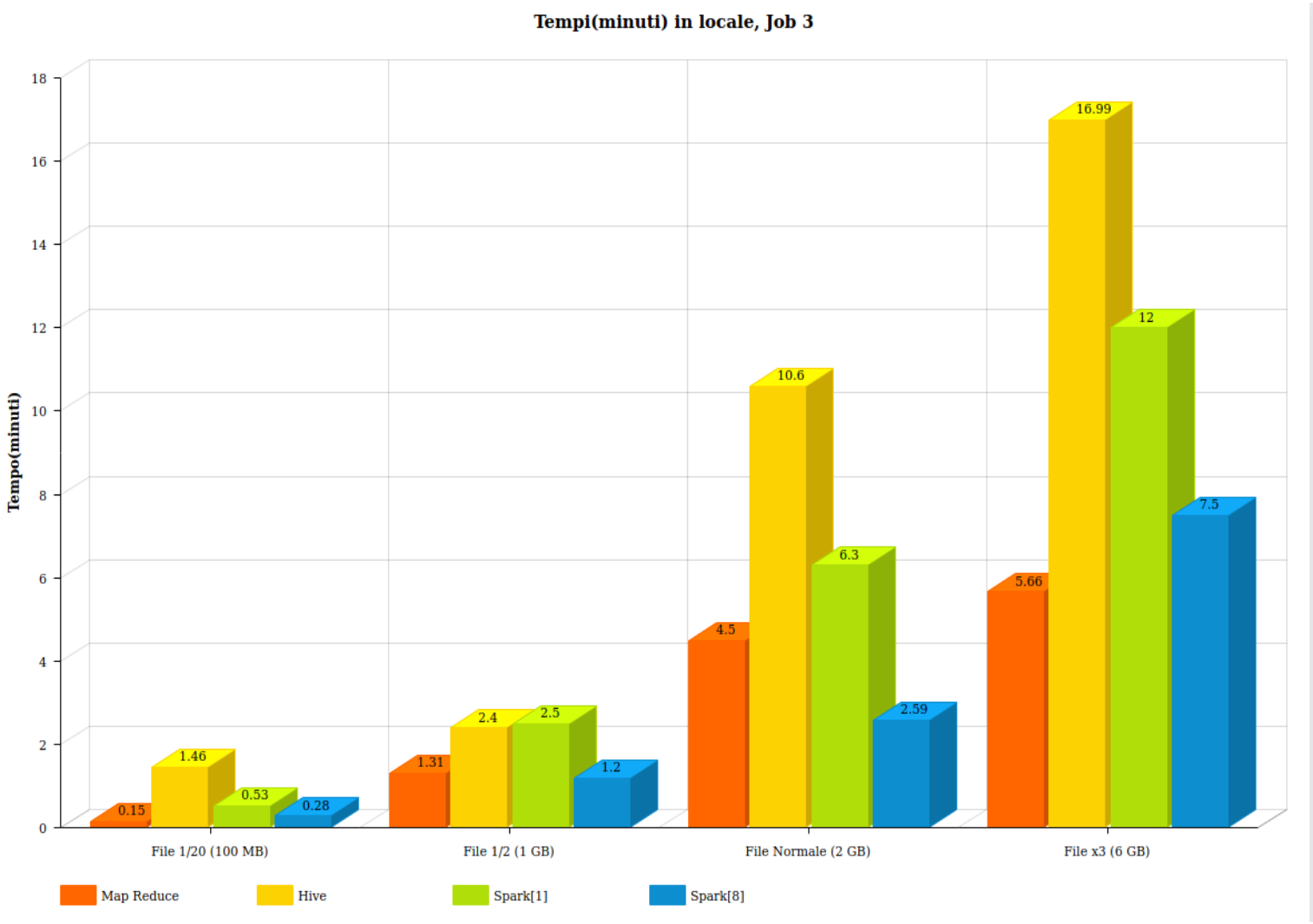



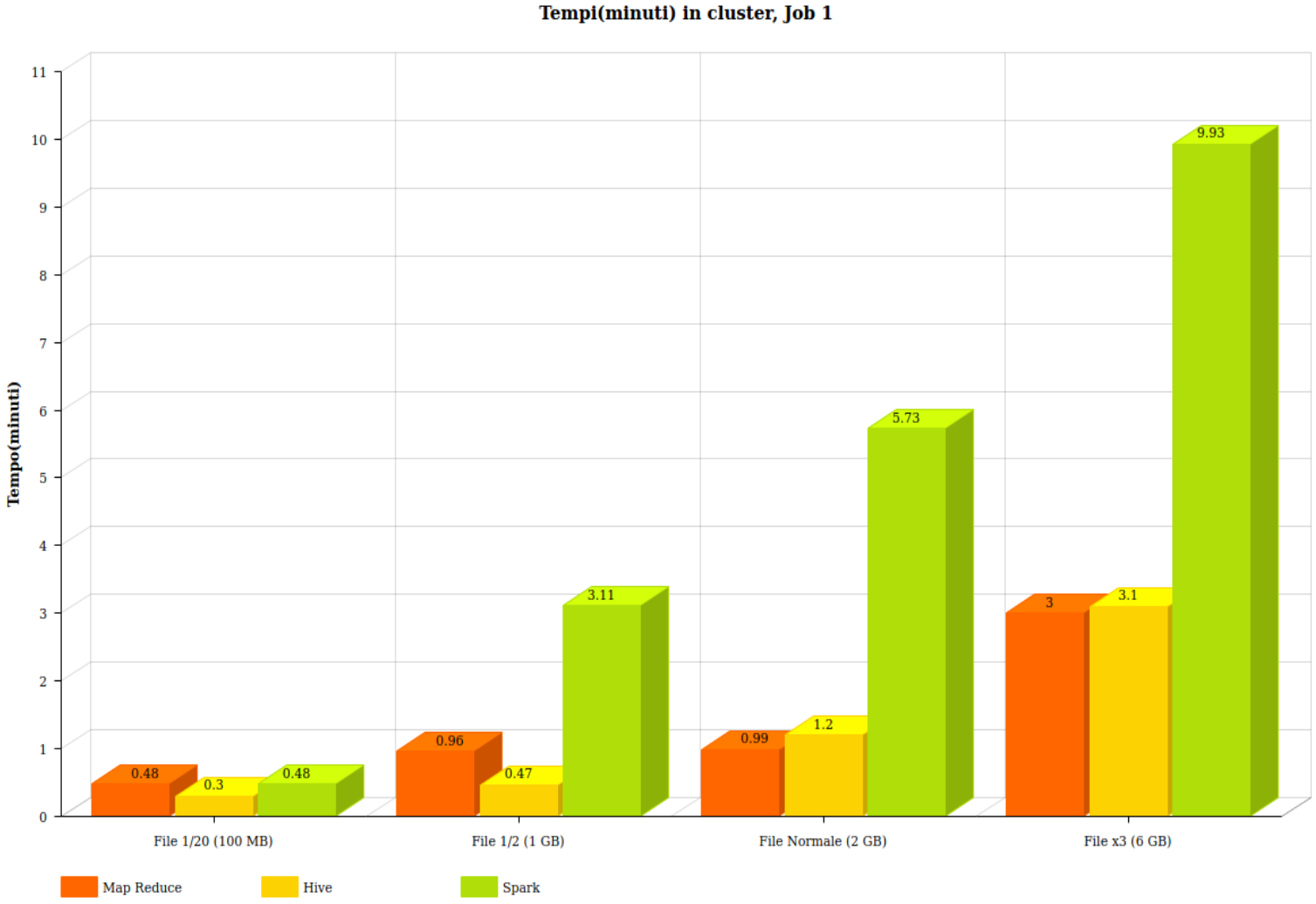
```
01, 1.97], '11': [0.3, 0.37], '12': [1.66, 1.46]},
 0.02], '11': [-0.13, -0.14], '12': [-0.06, 0.0]},
 0.18], '11': [-0.02, 0.04], '12': [0.35, 0.06]},
, 0.16], '11': [-0.02, -0.17], '12': [0.35, 0.11]},
0.11], '11': [-0.02, -0.13], '12': [0.35, -0.06]},
09, 0.05], '11': [-0.19, -0.03], '12': [-0.11, -0.1]},
, -0.09], '11': [-0.19, 0.08], '12': [-0.11, 0.22]},
 0.11], '11': [-0.19, -0.13], '12': [-0.11, -0.06]},
, 0.02], '11': [-0.19, -0.14], '12': [-0.11, 0.0]},
 0.04], '11': [-0.19, 0.05], '12': [-0.11, 0.0]}
```

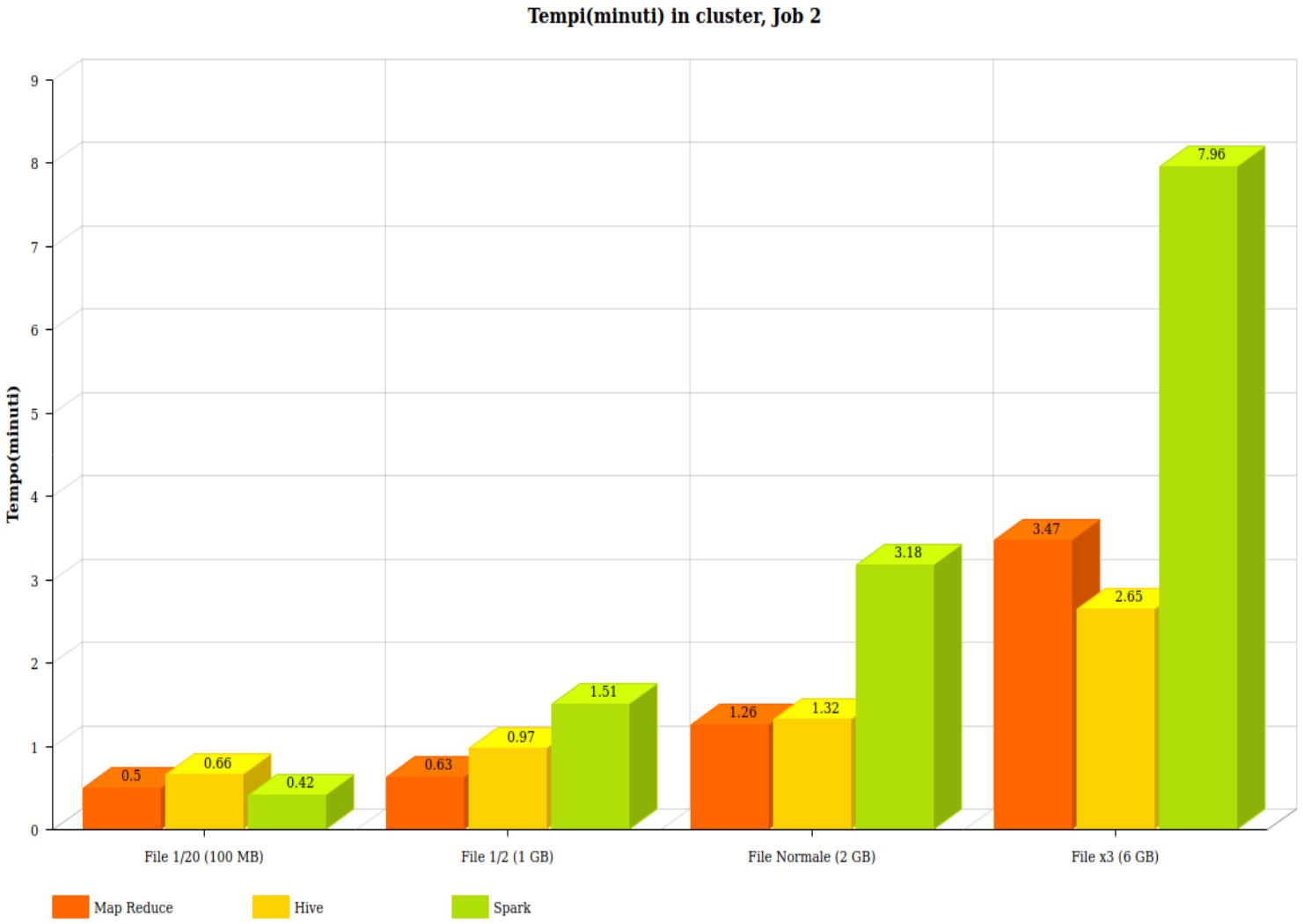
Di seguito grafici sulle prestazioni, per input variabile e per esecuzioni effettuate in locale e tramite cluster.

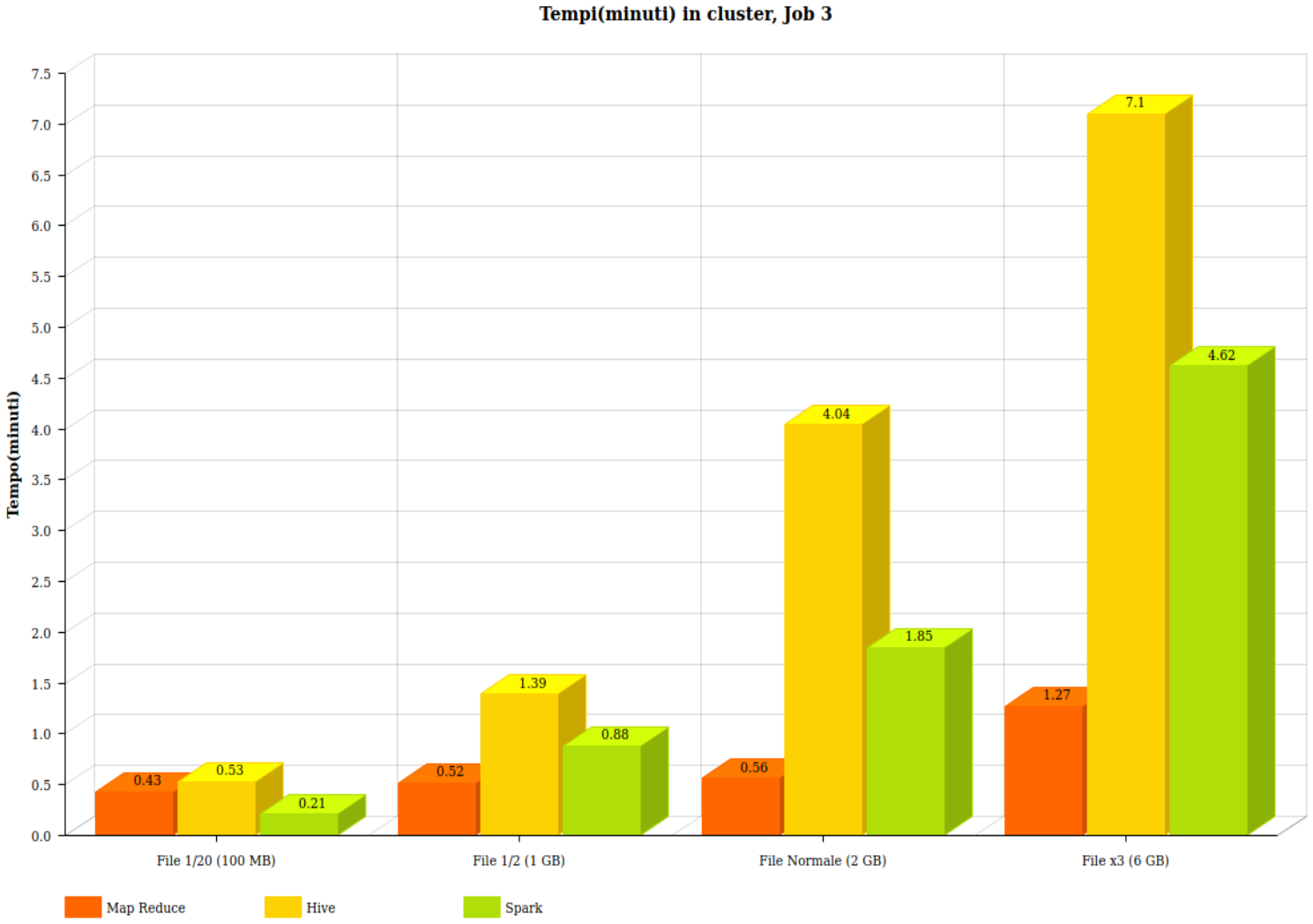



Tempi(minuti) in locale, Job 2
Tempo(minuti)
24
22
20
18
16
14
12
10
8
6
4
2
0
Hive of File x3 (6 GB):21.61
0.16
1.97
0.58
0.34
1.5
4.63
3.16
1.51
2.3
7.5
6.61
3.37
12
21.61
18.71
8.2
File 1/20 (100 MB)
File 1/2 (1 GB)
File Normale (2 GB)
File x3 (6 GB)
Map Reduce
Hive
Spark[1]
Spark[8]


Tempi(minuti) in locale, Job 3
Tempo(minuti)
0
2
4
6
8
10
12
14
16
18
0.15
1.46
0.53
0.28
1.31
2.4
2.5
1.2
4.5
10.6
6.3
2.59
5.66
16.99
12
7.5
File 1/20 (100 MB)
File 1/2 (1 GB)
File Normale (2 GB)
File x3 (6 GB)
Map Reduce
Hive
Spark[1]
Spark[8]


Tempi(minuti) in cluster, Job 1
Tempo(minuti)
0
1
2
3
4
5
6
7
8
9
10
11
0.48
0.3
0.48
0.96
0.47
3.11
0.99
1.2
5.73
3
3.1
9.93
File 1/20 (100 MB)
File 1/2 (1 GB)
File Normale (2 GB)
File x3 (6 GB)
Map Reduce
Hive
Spark


Tempi(minuti) in cluster, Job 2
Tempo(minuti)
0
1
2
3
4
5
6
7
8
9
0.5
0.66
0.42
0.63
0.97
1.51
1.26
1.32
3.18
3.47
2.65
7.96
File 1/20 (100 MB)
File 1/2 (1 GB)
File Normale (2 GB)
File x3 (6 GB)
Map Reduce
Hive
Spark


Tempi(minuti) in cluster, Job 3
Tempo(minuti)
7.5
7.0
6.5
6.0
5.5
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0.43
0.53
0.21
0.52
1.39
0.88
0.56
4.04
1.85
1.27
7.1
4.62
File 1/20 (100 MB)
File 1/2 (1 GB)
File Normale (2 GB)
File x3 (6 GB)
Map Reduce
Hive
Spark

Job 1 - MapReduce (minuti)	Locale	Cluster
Un ventesimo del file originale	**0.26**	0.48
Metà del file originale	1.86	**0.96**
File originale	4	**0.99**
File originale triplicato	6.46	**3**

Job 1 - Hive (minuti)	Locale	Cluster
Un ventesimo del file originale	1.46	**0.30**
Metà del file originale	3.45	**0.47**
File originale	6.46	**1.20**
File originale triplicato	16.83	**3.1**

Job 1 - Spark (minuti)	Locale[1]	Locale[8]	Cluster
Un ventesimo del file originale	0.86	**0.39**	0.48
Metà del file originale	5.81	**2.66**	3.11
File originale	10.75	6.53	**5.73**
File originale triplicato	21.78	13.6	**9.93**

Job 2 - MapReduce (minuti)	Locale	Cluster
Un ventesimo del file originale	**0.16**	0.50
Metà del file originale	1.50	**0.63**
File originale	2.30	**1.26**
File originale triplicato	12	**3.47**

Job 2 - Hive (minuti)	Locale	Cluster
Un ventesimo del file originale	1.97	**0.66**
Metà del file originale	4.63	**0.97**
File originale	7.50	**1.32**
File originale triplicato	21.61	**2.65**

Job 2 - Spark (minuti)	Locale[1]	Locale[8]	Cluster
Un ventesimo del file originale	0.58	**0.34**	0.42
Metà del file originale	3.16	**1.51**	**1.51**
File originale	6.61	3.37	**3.18**
File originale triplicato	18.71	8.2	**7.96**

Job 3 - MapReduce (minuti)	Locale	Cluster
Un ventesimo del file originale	**0.15**	0.43
Metà del file originale	1.31	**0.52**
File originale	4.5	**0.56**
File originale triplicato	5.66	**1.27**

Job 3 - Hive (minuti)	Locale	Cluster
Un ventesimo del file originale	1.46	**0.53**
Metà del file originale	2.40	**1.39**
File originale	10.60	**4.04**
File originale triplicato	16.99	**7.10**

Job 3 - Spark (minuti)	Locale[1]	Locale[8]	Cluster
Un ventesimo del file originale	0.53	0.28	**0.21**
Metà del file originale	2.50	1.20	**0.88**
File originale	6.30	2.59	**1.85**
File originale triplicato	12	7.5	**4.62**

4.2 Considerazioni

In generale, i tempi sono stati influenzati da una SSD quasi piena ed una partizione dedicata a Linux ridotta - la cpu performante ha migliorato leggermente la situazione.
Le esecuzioni su Spark sono state effettuate sia tramite single thread che multi-thread(8) per notare le effettive potenzialità, infatti si possono presentare variazioni di tempi anche dimezzati, o diminuiti del 60%.
Si può notare come nel job1, Spark e Hive siano l'esecuzione meno performante - alcuni miglioramenti potrebbero essere utlizzare mapPartitions() al posto di map(), evitare di usare funzioni definite dall'utente o utilizzare i dati immagazzinati in cache per quanto riguarda Spark - mentre per Hive si può ricorre all'uso di viste e partizioni. Si può notare che nel job2 i tempi risultano essere leggermente maggiori rispetto agli altri due job, questo dovuto probabilmente al join obbligato dei file stocks e stock_prices per ricavare il settore.
Nello specifico, spark e hive mantengono tempi di esecuzione simili per tutti gli input. Al contrario map-reduce risulta essere quello più efficiente: circa il 50% più veloce delle altre due tecnologie.

Infine, nel job 3, si ottengono buone prestazioni - il solo Hive presenta dei picchi, che possono essere abbassati tramite l'uso di viste e partizioni sui campi.
Per quanto riguarda le esecuzioni su cluster si possono osservare tempi migliori dovuti sia alla modalità cluster ma anche grazie a specifiche hardware migliori rispetto a quelle usate in locale.
Da aggiungere che per Hive e Map Reduce sono stati usati i cluster di AWS, mentre per Spark è stato utilizzato yarn in modalità cluster, per questo si può notare che le prestazioni cluster di Spark sono state meno efficaci delle controparti su Hive e Map Reduce. Per file di dimensioni piccole ci sono miglioramenti se non tempi uguali, ma all'aumentare delle dimensioni si può notare come in ambiente cluster i tempi abbiano un tempo lineare, mentre nel locale si ha un andamento esponenziale.
Alla conclusione di questo progetto, ci si ritiene soddisfatti dei risultati e delle nuove esperienze sperimentate con questi nuovi strumenti.