Rapporto Finale sul Primo Progetto del corso Big Data

May 24, 2020

0.0.1 Progettazione e realizzazione di job in MapReduce, Hive e Spark

Progetto a cura del gruppo **Data-FaSt** (Farioli Davide, Stojani Marjo)

0.0.2 Indice trattazione

0.1 1. Specifiche Hardware e Software

L'esecuzione dei Job è stata svolta sia in macchina locale che su Cluster AWS

I Job in MapReduce e Spark sono stati realizzati con il linguaggio di programmazione **Python**

0.1.1 Macchina Locale

Specifiche Hardware:

- **Processor:** Intel Core i5 4th Gen 4310U
- **Memory:** 8GB
- **Storage:** 256GB (SSD)

Specifiche Software:

- **SO:** Linux Mint 19
- **Python:** 3.7.*
- **Java:** 1.8
- **Hadoop:** 3.2.1
- **Hive:** 2.3.7
- **Spark:** 3.0.0

0.1.2 Cluster AWS

Specifiche Hardware:

- **Istanza EMR:** m5.xlarge
- **Nodes:** 1 Master, 2 Core
- **Storage:** S3 bucket

Specifiche Software:

- **Python:** 3.7.*
- **Java:** 1.8
- **Hadoop:** 3.2.1
- **Hive:** 3.1.2
- **Spark:** 3.0.0

0.2 2. Specifiche dataset

Il Dataset di riferimento **Daily Historical Stock Prices** 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: historical_stock_prices.csv e historical_stocks.csv

historical_stock_prices.csv è composto da 20973889 righe e 8 colonne con i seguenti campi:

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

il file non presenta campi con valore nullo

il minimo valore nel campo 'date' è: '1970-01-02'

il massimo valore nel campo 'date' è: '2018-08-24'

[19]:
```
historical_stock_prices.head()
```

[19]:

	ticker	open	close	adj_close	low	high	volume	date
0	AHH	11.50	11.58	8.493155	11.25	11.68	4633900	2013-05-08
1	AHH	11.66	11.55	8.471151	11.50	11.66	275800	2013-05-09
2	AHH	11.55	11.60	8.507822	11.50	11.60	277100	2013-05-10
3	AHH	11.63	11.65	8.544494	11.55	11.65	147400	2013-05-13
4	AHH	11.60	11.53	8.456484	11.50	11.60	184100	2013-05-14

historical_stocks.csv è composta da 6460 righe e 5 colonne con i seguenti campi:

- **ticker:** simbolo univoco dell'azione
- **exchange:** NYSE o NASDAQ
- **name:** nome dell'azienda
- **sector:** settore dell'azienda

- **industry:** industria di riferimento per l'azienda

il file presenta i campi 'sector' e 'industry' con valori nulli

ad un'azienda (campo 'name') possono essere associati anche diversi ticker

un'azienda può essere associata a diversi settori

[22]:
```
historical_stocks.head()
```

[22]:

	ticker	exchange	name	sector	industry
0	PIH	NASDAQ	1347 PROPERTY INSURANCE HOLDINGS, INC.	FINANCE	PROPERTY-CASUALTY INSURERS
1	PIHPP	NASDAQ	1347 PROPERTY INSURANCE HOLDINGS, INC.	FINANCE	PROPERTY-CASUALTY INSURERS
2	TURN	NASDAQ	180 DEGREE CAPITAL CORP.	FINANCE	FINANCE/INVESTORS SERVICES
3	FLWS	NASDAQ	1-800 FLOWERS.COM, INC.	CONSUMER SERVICES	OTHER SPECIALTY STORES
4	FCCY	NASDAQ	1ST CONSTITUTION BANCORP (NJ)	FINANCE	SAVINGS INSTITUTIONS

0.3 3. Realizzazione Job1

0.3.1 3.1 Specifiche

Realizzare un job che sia in grado di generare le statistiche di ciascuna azione tra il 2008 e il 2018 indicando, per ogni azione: il simbolo, la variazione della quotazione (differenza percentuale arrotondata tra i prezzi di chiusura iniziale e finale dell'intervallo temporale), il prezzo minimo, quello massimo e il volume medio nell'intervallo, ordinando l'elenco in ordine decrescente di variazione della quotazione

0.3.2 3.2 MapReduce

Linguaggio di programmazione utilizzato: **Python** 3.7

Per la realizzazione del Job1 in MapReduce è stato sufficiente 1 stage, sono quindi stati implementati 1 mapper e 1 reducer

Mapper Per realizzare il Job1 è necessario utilizzare solo il file **historical_stock_prices.csv** dunque il mapper svolge solo un'operazione di filtraggio dei dati in base alla data del ticker, accettando solo quelle di interesse (comprese tra il 2008 e il 2018) e passando al reducer solo le colonne 'ticker,close,volume,date' di cui ticker sarà la chiave per lo shuffle and sort; in questa maniera vengono scartate tutte le righe e colonne di dati non necessarie ai fini dell'obiettivo del job riducendo sin da subito anche il traffico di dati all'interno del file system distribuito

```
for line in input:
    ticker,open,close,adj_close,low,high,volume,date = line
    year = yearOf(date)
    if((year >= 2008) and (year <= 2018)):
        print(ticker,close,volume,date)
```

Mapper: pseudocodifica

Reducer Il Reducer prende in input i valori filtrati dal Mapper e riordinati dalla fase di shuffle and sort; ogni stream in input ha dunque i seguenti campi: 'ticker,close,volume,date'. Viene utilizzata una struttura dati di supporto per registrare i dati relativi ad ogni ticker in base al suo obiettivo: cumulare il volume per poi poterne ricavare la media complessiva, registrare il prezzo di chiusura con la data più piccola ed il prezzo di chiusura con la data più grande per poterne ricavare la variazione percentuale, registrare il prezzo minimo e il prezzo massimo.

```
STRUCT = {}
for line in input:
    ticker,close,volume,date = line
    cumulate volume in STRUCT(key=this ticker);
    if close is the lowest: register in STRUCT(key=this ticker);
    if close is the highest: register in STRUCT(key=this ticker);
    if close has the lowest date: register in STRUCT(key=this ticker);
    if close has the highest date: register in STRUCT(key=this ticker);

for object in STRUCT():
    average_volume = cumulated_volume / count
    variation = ((close_with_highest_date / close_with_lowest_date ) * 100 ) - 100
    insert(ticker,variation,average_volume,lowest,highest) in output_struct
SORT output_struct by variation
print first 10 lines
```

Reducer: pseudocodifica

0.3.3 3.3 Hive

Per la realizzazione del Job1 in Hive è necessario creare una tabella dove vengono inseriti i dati del file in input **historical_stock_prices.csv**. Successivamente vengono create diverse Viste:

nella Vista 'single_fields_statistics' vengono calcolati il volume medio, il prezzo minimo ed il prezzo massimo in modo semplice grazie al raggruppamento in base al ticker, ovviamente filtrando solo le righe per cui la data è compresa tra il 2008 e il 2018;

la Vista 'first_last_date' registra la minima e la massima data disponibile per ogni ticker, questa vista è utile per la creazione delle successive;

le Viste 'first_close' e 'last_close' registrano il prezzo relativo rispettivamente alla data minima e alla data massima per ogni ticker;

la Vista 'ticker_variation' calcola la variazione percentuale per ogni ticker sfruttando i valori ricavati dal join delle viste 'first_close' e 'last_close';

la Vista 'job1_result' effettua un join tra la 'ticker_variation' e la 'single_fields_statistics' in base al ticker ottenendo così tutti i valori utili per l'output che si potranno ottenere con una semplice SELECT *

```
set hive.auto.convert.join = false;

CREATE TABLE IF NOT EXISTS prices (
                                    ticker STRING,
                                    open DOUBLE,
                                    close INT,
                                    adj_close DOUBLE,
                                    low DOUBLE,
                                    high DOUBLE,
                                    volume DOUBLE,
                                    ticker_date DATE)

ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
TBLPROPERTIES("skip.header.line.count"="1");

LOAD DATA LOCAL INPATH '/home/hadoop/historical_stock_prices.csv'
OVERWRITE INTO TABLE prices;


DROP VIEW IF EXISTS single_fields_statistics;
CREATE OR REPLACE VIEW single_fields_statistics AS
SELECT p.ticker, min(p.close) AS min_price, max(p.close) AS max_price, avg(p.volume) AS avg_volume
FROM prices AS p
WHERE p.ticker_date BETWEEN Date('2008-01-01') AND Date('2018-12-31')
GROUP BY p.ticker;

DROP VIEW IF EXISTS first_last_date;
CREATE OR REPLACE VIEW first_last_date AS
SELECT p.ticker, min(p.ticker_date) AS first_date, max(p.ticker_date) AS last_date
FROM prices AS p
WHERE p.ticker_date BETWEEN Date('2008-01-01') AND Date('2018-12-31')
GROUP BY p.ticker;

DROP VIEW IF EXISTS first_close;
CREATE OR REPLACE VIEW first_close AS
SELECT p.ticker, fd.first_date, p.close
FROM first_last_date AS fd
JOIN prices AS p ON (fd.ticker=p.ticker) AND (p.ticker_date=fd.first_date);

DROP VIEW IF EXISTS last_close;
CREATE OR REPLACE VIEW last_close AS
SELECT p.ticker, fd.last_date, p.close
FROM first_last_date AS fd
JOIN prices AS p ON (fd.ticker=p.ticker) AND (p.ticker_date=fd.last_date);

DROP VIEW IF EXISTS ticker_variation;
CREATE OR REPLACE VIEW ticker_variation AS
SELECT fc.ticker, (((lc.close-fc.close)/fc.close) * 100) AS variation
FROM first_close AS fc
JOIN last_close AS lc ON lc.ticker=fc.ticker;

DROP VIEW IF EXISTS job1_result;
CREATE OR REPLACE VIEW job1_result AS
SELECT  sts.ticker, tv.variation, sts.min_price, sts.max_price, sts.avg_volume
FROM single_fields_statistics AS sts
JOIN ticker_variation AS tv ON tv.ticker=sts.ticker
ORDER BY tv.variation DESC;

INSERT OVERWRITE LOCAL DIRECTORY 'output/job1/'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
SELECT * FROM job1_result LIMIT 10
```

0.3.4 3.4 Spark

```
# Job 1

# Crea un nuovo Spark Contest per l'applicazione
sc = SparkContext(SparkConfiguration)

# Carica il dataset dei prezzi in un rdd
dataset = sc.textFile(filePath)

# Converti ogni linea del dataset in una lista, scarta il primo valore
formatted_rdd = dataset.map(line->splitLine()).filter(line != dataset.first())

# Filtra i dati, prendendo solo le righe con una data superiore al 2008
rdd_year_filtered = formatted_rdd_values.filter(line.data.year > '2008')

# Calcola il prezzo minimo, aggregando per la chiave "ticker" e salvandolo nell'rdd dei prezzi minimi
min_price = rdd_year_filtered.reduceByKey(line->minPrice)

# Calcola il prezzo massimo, aggregando per la chiave "ticker" e salvandolo nell'rdd dei prezzi massimi
max_price = rdd_year_filtered.reduceByKey(line->maxPrice)

# Calcola il volume medio, aggregando per la chiave "ticker" e salvando il risultato nell'rdd dei volumi medi
avg_volume = rdd_year_filtered.reduceByKey(sumLinesVolume).map(sumLinesVolume/lines)

# Aggrega i valori sul campo ticker e prende il primo prezzo di chiusura, salvandolo nell'rdd dei ticker-primo
prezzo
first_price = rdd_year_filtered.reduceByKey(extractFirsPrice())


# Aggrega i valori sul campo ticker e prende l'ultimo prezzo di chiusura, salvandolo nell'rdd dei ticker-primo
prezzo
last_price = rdd_year_filtered.reduceByKey(extractLastPrice())

# Calcola la variazione percentuale per ogni ticker, rispetto al prezzo della prima data e dell'ultima data,
estrandoli dai precedenti rdd creati
variation = last_price.join(first_price).map(line -> ((end_price - start_price) / start_price ) * 100)

# Si uniscono gli rdd precedenti in un ultimo rdd di output, accomunandoli per il campo (ticker)
output = max_price.join(min_price).join(avg_volume).join(variation)

# Formatta l' rdd di output in modo da avere, per ogni linea d'output
# [ticker, variazione della quotazione, prezzo minimo, prezzo massimo, volume medio]
# e riordina in maniera decrescente sul campo variazione di ogni record
output_formatted = output.map(outputLine -> outputFormatLine).sortBy(output.variazione, ascending=False)

# Stampo i primi 10 valori dell'output
print(output_formatted.take(10))
```

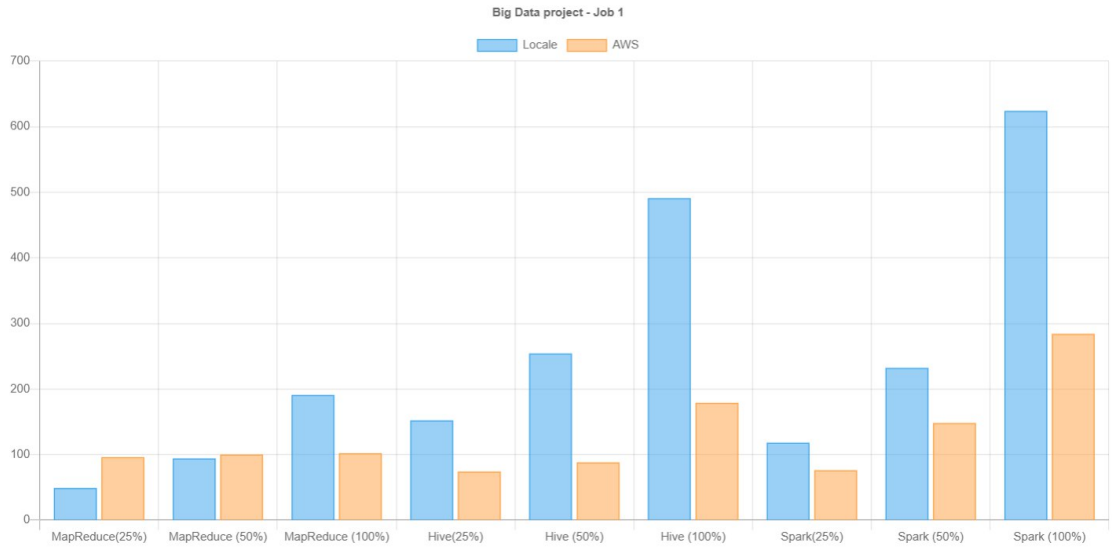
0.3.5 3.5 Risultati

ticker	variation	min_low	max_high	volumeMean
EAF	267757%	0.0020000000949949	22.3700008392334	1245139.0384615385
ORGS	217416%	0.00313999992795289	19.7999992370605	8570.962343096235
PUB	179900%	0.00899999961256981	135.0	34449.40070921986
RMP	121082%	0.0299999993294477	78.5498504638672	120487.04767063922
CTZ	120300%	0.00499999988824129	26.7299995422363	52050.27308066084
CCD	111250%	0.0149999996647239	25.5499992370605	105416.8926615553
SAB	110429%	1.39849996566772	318800.0	1695378.0163179915
KE	99400%	0.0149999996647239	22.2000007629395	73249.84615384616
LN	96700%	0.0149999996647239	49.6300010681152	394344.92964071856
GHG	64850%	0.00499999988824129	24.3099994659424	607659.2926292629

0.3.6 3.6 Grafico

Il grafico seguente mostra il tempo di esecuzione (l'ascissa indica il tempo in secondi) del Job1 differenziato per tecnologia utilizzata (mapReduce, Hive o Spark), grandezza del dataset in input

(25%, 50% o 100%)e luogo fisico di esecuzione (macchina locale o Cluster AWS).



0.4 4. Realizzazione Job2

0.4.1 4.1 Specifiche

Realizzare un job che sia in grado di generare, per ciascun settore, il relativo "trend" nel periodo 2008-2018 ovvero un elenco contenete, per ciascun anno nell'intervallo: il volume annuale medio delle azioni del settore, la variazione annuale media delle aziende del settore e la quotazione giornaliera media delle aziende del settore.

0.4.2 4.2 MapReduce

Linguaggio di programmazione utilizzato: **Python** 3.7

Per la realizzazione del Job2 in MapReduce sono serviti 2 stage, sono quindi stati implementati 2 mapper e 2 reducer. Il primo stage necessario per il join dei due file di input, il secondo per l'elaborazione dell'output.

0.4.3 - Stage 1

Mapper 1 Per realizzare il Job2 è necessario utilizzare sia il file **historical_stock_prices.csv** sia il file **historical_stocks.csv**; dunque il mapper dello stage 1, oltre a svolgere un'operazione di filtraggio dei dati in base alla data del ticker, svolge un'operazione di riconoscimento dell'origine dell'input, passaggio fondamentale per la cooperazione con il reducer che potrà trattare le righe di input in maniera adeguata per ricavare i dati da elaborare. Un modo per riconoscere a quale file di input appartiene lo stream in esame è controllare il numero dei campi, visto che i file di input hanno un numero di colonne diverso tra loro. L'output generato dal mapper sarà composto dal formato 'ticker,close,volume,date,sector' di cui i primi due campi compongono la chiave, strategia utilizzata per ordinare in base al ticker e passare come primo dato utile il settore del ticker.

```
for line in input:

      if(len(fields) == 8):
          # fields are from historical_stock_prices.csv file

          ticker,open,close,adj_close,low,high,volume,date = line

          year = yearOf(date)

          if((year >= 2008) and (year <= 2018)):

              print('{}\t{}\t{}\t{}\t-'.format(ticker,close,volume,date))

      else:
          # fields are from historical_stocks.csv file

          ticker,exchange,name,sector,industry = fields

          if(sector != 'N/A'):
              print('{}\t-\t-\t-\t{}'.format(ticker,sector))
```

Mapper 1: pseudocodifica

Reducer 1 Il Reducer dello stage 1 si occupa del effettivo join dei dati. Sfruttando la fase di shuffle and sort su due chiavi, il reducer riconosce il settore del ticker con il primo stream e lo associa a tutte le righe di dati del ticker

```
for line in input:

      ticker,close,volume,date,sector = line

      if(sector != '-'):
          current_sector = sector
          is_sector_line = True
      else:
          is_sector_line = False

      if not is_sector_line:

          print('{}\t{}\t{}\t{}\t{}'.format(current_sector,ticker,close,volume,date))
```

Reducer 1: pseudocodifica

0.4.4 - Stage 2

Mapper 2 La fase di mapping dello stage 2 prevede una semplicissima lettura dei dati di input senza operare filtraggi mandando in output stream in formato 'sector,ticker,close,volume,date', in modo da operare il successivo shuffle and sort sul campo 'sector'

```
for line in input:
    sector,ticker,close,volume,date = line
    print('{}\t{}\t{}\t{}\t{}'.format(sector,ticker,close,volume,date))
```

Mapper 2: pseudocodifica

Reducer 2 Il Reducer dello stage 2 utilizza diverse strutture dati di supporto utili per le 2 principali fasi: calcolo dei valori medi annuali('volume', 'close' e variazione percentuale) per ogni ticker, calcolo della media annuale dei valori medi annuali di tutti i ticker di un settore.

La prima dictionary ha come chiave la tupla (ticker,year) e come valori vengono cumulati il volume e il close per calcolarne la media, il close con data minima per quel anno e il close con data massima per quel anno.

La seconda dictionary ha come chiave la tupla (sector,year) e come valori vengono comulati il volume medio, il close medio e la variation media dei ticker associati al settore per quel anno.

```
trend_dict = {} # dict of ticker per year trend
sectors_dict = {} # dict to mantain ticker-sector link
final_dict = {} # dict of sector per year trend

for line in input:

    sector,ticker,close,volume,date = line

    year = yearOf(date)

    # dict to mantain ticker-sector link
    if(ticker not in sectors_dict):
        sectors_dict[ticker] = sector

    #  insert new ticket-year data
    if((ticker,year) not in trend_dict):
        trend_dict[(ticker,year)]=[volume,close,1,date,close,date,close]
    else:
        # ticker-year already in dict
        cumulate util values: volume, close, count in trend_dict

        find ticker-year initial and final close value

for (ticker,year) in trend_dict.keys():

    calculate average values for ticker-year
    avg_volume = cumalated_volume / count
    avg_close = cumulated_close / count

    get final_close_value from dict by ticker,year
    get initial_close_value from dict by ticker,year

    avg_variation = round((final_close/initial_close)*100 - 100)

    # get ticker's corresponding sector
    sector = sectors_dict[ticker]

    #  insert new sector-year data for every ticker avg values
    if((sector,year) not in final_dict):
        final_dict[(sector,year)]=[avg_volume,avg_variation,avg_close,1]
    else:
        # sector-year already in dict
        cumulate ticker-year avg values in final_dict

for (sector,year) in final_dict.keys():

    calculate average values for sector-year

    print('{}\t{}\t{}\t{}\t{}'.format(sector,year,final_volume,final_variation,final_close))
```

Reducer 2: pseudocodifica

0.4.5 4.3 Hive

Per la realizzazione del Job2 in Hive è necessario creare due tabelle dove vengono inseriti i dati dei file in input **historical_stock_prices.csv** e **historical_stocks.csv**. Successivamente vengono create diverse Viste:

nella Vista 'joined_dataset' viene effettuato il join tra le due tabelle appena popolate filtrando solo le righe per cui la data è compresa tra il 2008 e il 2018 e mantenendo solo le colonne di interesse: 'ticker,volume,close,ticker_year,sector'

la Vista 'avg_volume_close' utilizza la vista 'joined_dataset' per calcolare il volume e il close medio raggruppando per sector e year

le Vista 'min_date_year' e 'max_date_year' registrano la minima e la massima data disponibile per ogni year per ogni ticker, queste viste sono utili per la creazione delle successive viste

'min_close_year' e 'max_close_year' per ricavare i rispettivi close per poi ricavare la variazione annuale dei ticker nella view 'ticker_variation';

le Viste 'min_close_year' e 'max_close_year' registrano il prezzo relativo rispettivamente alla data minima e alla data massima per ogni ticker;

la Vista 'sector_variation' calcola la variazione percentuale media delle variazioni percentuali dei ticker per year, associandola al relativo sector;

la Vista 'job2_result' effettua un join tra la 'sector_variation' e la 'avg_volume_close' in base al sector ottenendo così tutti i valori utili per l'output che si potranno ottenere con una semplice SELECT *

```
set hive.auto.convert.join = false;

CREATE TABLE IF NOT EXISTS prices (ticker STRING,
                                   open DOUBLE,
                                   close DOUBLE,
                                   adj_close DOUBLE,
                                   low DOUBLE,
                                   high DOUBLE,
                                   volume INT,
                                   ticker_date DATE)

ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
TBLPROPERTIES("skip.header.line.count"="1");

LOAD DATA LOCAL INPATH '/home/hadoop/historical_stock_prices.csv'
OVERWRITE INTO TABLE prices;

CREATE TABLE IF NOT EXISTS stocks (ticker STRING,
                                   stock_exchange STRING,
                                   name STRING,
                                   sector STRING,
                                   industry STRING)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES (
    "separatorChar" = ",",
    "quoteChar" = "\"")
STORED AS TEXTFILE
TBLPROPERTIES("skip.header.line.count"="1");

LOAD DATA LOCAL INPATH '/home/hadoop/historical_stocks.csv'
OVERWRITE INTO TABLE stocks;

DROP VIEW IF EXISTS joined_dataset;
CREATE OR REPLACE VIEW joined_dataset AS
SELECT p.ticker AS ticker, volume, close, YEAR(ticker_date) AS ticker_year, sector
FROM prices p JOIN stocks s ON p.ticker=s.ticker
WHERE sector != 'N\A' AND ticker_date >= '2008-01-01' AND ticker_date <= '2018-12-31';

DROP VIEW IF EXISTS avg_volume_close;
CREATE OR REPLACE VIEW avg_volume_close AS
SELECT sector, ticker_year, AVG(volume) AS avg_volume, AVG(close) AS avg_close
FROM joined_dataset
GROUP BY sector, ticker_year
ORDER BY sector, ticker_year;

DROP VIEW IF EXISTS min_date_year;
CREATE OR REPLACE VIEW min_date_year AS
SELECT ticker, min(ticker_date) AS min_date, YEAR(ticker_date) AS ticker_year
FROM prices
WHERE ticker_date >= '2008-01-01' AND ticker_date <= '2018-12-31'
GROUP BY ticker, YEAR(ticker_date);

DROP VIEW IF EXISTS max_date_year;
CREATE OR REPLACE VIEW max_date_year AS
SELECT ticker, max(ticker_date) AS max_date, YEAR(ticker_date) AS ticker_year
FROM prices
WHERE ticker_date >= '2008-01-01' AND ticker_date <= '2018-12-31'
GROUP BY ticker, YEAR(ticker_date);

DROP VIEW IF EXISTS min_close_year;
CREATE OR REPLACE VIEW min_close_year AS
SELECT p.ticker AS ticker, ticker_year, close
FROM prices p JOIN min_date_year m ON p.ticker=m.ticker AND p.ticker_date=m.min_date;

DROP VIEW IF EXISTS max_close_year;
CREATE OR REPLACE VIEW max_close_year AS
SELECT p.ticker AS ticker, ticker_year, close
FROM prices p JOIN max_date_year m ON p.ticker=m.ticker AND p.ticker_date=m.max_date;

DROP VIEW IF EXISTS ticker_variation;
CREATE OR REPLACE VIEW ticker_variation AS
SELECT  mi.ticker AS ticker, mi.ticker_year as ticker_year, ROUND(((ma.close/mi.close) *100) - 100) AS variation
FROM min_close_year mi JOIN max_close_year ma ON mi.ticker=ma.ticker AND mi.ticker_year=ma.ticker_year;

DROP VIEW IF EXISTS sector_variation;
CREATE OR REPLACE VIEW sector_variation AS
SELECT sector, j.ticker_year AS sector_year, AVG(t.variation) AS variation
FROM joined_dataset j JOIN ticker_variation t ON j.ticker= t.ticker AND j.ticker_year=t.ticker_year
GROUP BY sector, j.ticker_year;

DROP VIEW IF EXISTS job2_result;
CREATE OR REPLACE VIEW job2_result AS
SELECT a.sector, a.ticker_year AS sector_year, a.avg_volume, s.variation, a.avg_close
FROM avg_volume_close a JOIN sector_variation s ON a.sector=s.sector AND a.ticker_year=s.sector_year;


INSERT OVERWRITE LOCAL DIRECTORY 'output/job2/'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
SELECT * FROM job2_result ORDER BY sector, sector_year;
```

0.4.6 4.4 Spark

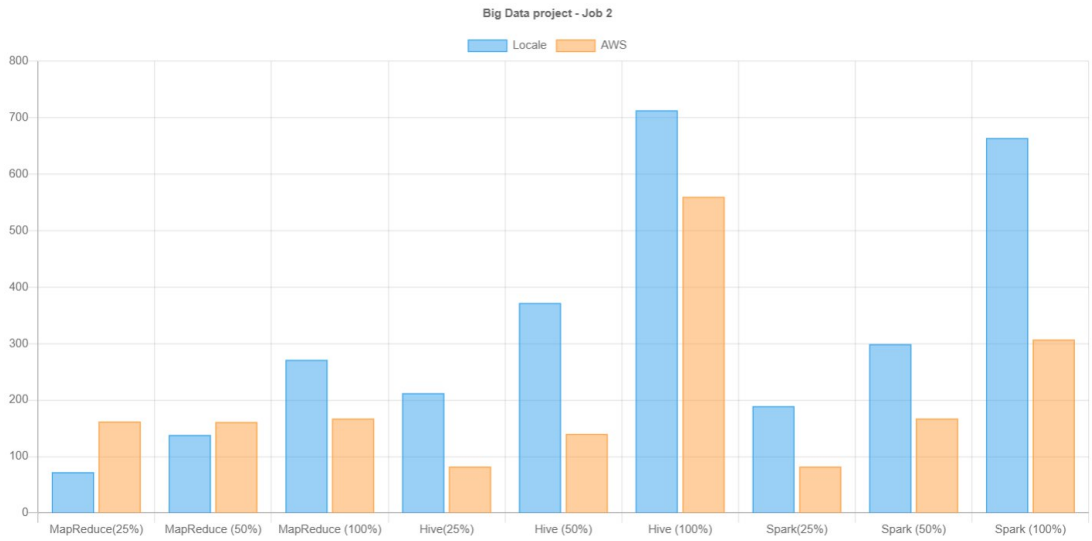
```
# Job 2

# Crea un nuovo Spark Contest per l'applicazione
sc = SparkContext(SparkConfiguration)

# Importa, formatta ed elimina la prima riga del dataset stocks
stocks = sc.textFile(stocks_path).map(line->splitLine()).filter(line != stocks.first())


# Importa, formatta ed elimina la prima riga del dataset price, filtra i record minori del 2008
prices = sc.textFile(price_path).map(line->splitLine()).filter(line !=price.first()).filter(line.data.year > '2008')

# Unisce i due rdd precedenti per creare il dataset necessario al job
# Ogni record ha come chiave il settore e l'anno, in modo da poter aggregare
dataset = stocks.join(prices).map(line->formatted_line)

# Calcola il volume medio, aggregando per la chiave (Settore, anno) e salvando il risultato nell'RDD dei volumi medi
avg_volume = dataset.reduceByKey(sumLinesVolume).map(sumLinesVolume/lines)

# Rdd d'appoggio in cui vengono salvati, per ogni (settore, anno) i relativi prezzi di chiusura ad inizio anno.
first_price = dataset.reduceByKey(extractFirsPrice())

# Rdd d'appoggio in cui vengono salvati, per ogni (settore, anno) i relativi prezzi di chiusura a fine anno.
last_price = dataset_rdd.reduceByKey(extractLastPrice())

# Calcola la variazione percentuale per ogni (settore, anno) rispetto al prezzo della prima data e dell'ultima data,
estrandoli dai precedenti rdd creati
variation = last_price.join(first_price).map(line -> ((last_price - first_price) / first_price ) * 100)

# Rdd in cui si calcola per ogni aggregazione di record (settore, anno) la relativa quotazione annuale media
avg_quotation = dataset.reduceByKey(sumLinesClosePrice).map(sumLinesClosePrice/lines)


# Si uniscono gli rdd precedenti nell'RDD di output, accomunandoli per la chiave (settore, anno)
output = avg_volume.join(variation).join(avg_quotation)

# Formatta l' rdd di output in modo da avere, per ogni linea d'output
# [Settore, anno, volume medio dell'anno, variazione dell'anno, quotazione media dell'anno]
output_formatted = output.map(outputLine -> outputFormatLine)

# Stampo i primi 10 valori dell'output
print(output_formatted.take(10))
```

0.4.7 4.5 Risultati

sector	year	avg_volume	avg_variation	avg_close
BASIC INDUSTRIES	2011	1484332	-11%	34
BASIC INDUSTRIES	2012	1268549	17%	33
BASIC INDUSTRIES	2013	1309870	25%	36
BASIC INDUSTRIES	2016	1436512	32%	31
BASIC INDUSTRIES	2015	1293918	-13%	34
BASIC INDUSTRIES	2009	1797216	76%	33
BASIC INDUSTRIES	2014	1320498	5%	37
BASIC INDUSTRIES	2017	1250242	20%	36
BASIC INDUSTRIES	2008	1714212	-41%	36
BASIC INDUSTRIES	2018	1183226	1%	40
BASIC INDUSTRIES	2010	1526043	26%	34

0.4.8 4.6 Grafici

Il grafico seguente mostra il tempo di esecuzione (l'ascissa indica il tempo in secondi) del Job2 differenziato per tecnologia utilizzata (mapReduce, Hive o Spark), grandezza del dataset in input (25%, 50% o 100%)e luogo fisico di esecuzione (macchina locale o Cluster AWS).



0.5 5. Realizzazione Job3

0.5.1 5.1 Specifiche

Realizzare un job in grado di generare gruppi di aziende le cui azioni hanno avuto lo stesso trend in termini di variazione annualenell'ultimo triennio disponibile, indicando le aziende e il trend comune (es. {Apple, Intel, Amazon}: 2016:-1%, 2017:+3%,2018:+5%).

0.5.2 5.2 MapReduce

Linguaggio di programmazione utilizzato: **Python** 3.7

Per la realizzazione del Job3 in MapReduce sono serviti 2 stage, sono quindi stati implementati 2 mapper e 2 reducer. Il primo stage necessario per il join dei due file di input, il secondo per l'elaborazione dell'output.

0.5.3 - Stage 1

Mapper 1 Per realizzare il Job3 è necessario anche qui utilizzare sia il file **historical_stock_prices.csv** sia il file **historical_stocks.csv**; dunque il mapper dello stage 1, oltre a svolgere un'operazione di filtraggio dei dati in base alla data del ticker, svolge un'operazione di riconoscimento dell'origine dell'input. L'output generato dal mapper sarà composto dal formato 'ticker,close,date,name'.

```
for line in input:
      if(len(fields) == 8):
          # condition verified if fields are from historical_stock_prices.csv file
          ticker,open,close,adj_close,low,high,volume,date = line
          year = yearOf(date)
          if((year >= 2016) and (year <= 2018)):
              print('{}\t{}\t{}\t-'.format(ticker,close,date))
      else:
          # fields are from historical_stocks.csv file
          ticker,exchange,name,sector,industry = line
          print('{}\t-\t-\t{}'.format(ticker,name))
```

Mapper 1: pseudocodifica

Reducer 1 Il Reducer dello stage 1 si occupa del effettivo join dei dati. Sfruttando una struttura dati con chiave (ticker,year) registra i 'close' con data minima e data massima per un certo ticker in un certo anno, in questo modo si riesce a calcolare la variazione percentuale annuale dei ticker sin dal primo reducer in modo da diminuire il flusso di dati toale. L'output generato sarà così composto 'name,ticker,year,variation' quindi come chiave avremo il nome dell'azienda

```
for line in input:

    ticker,close,date,company = line
    #the first line should contain company information
    register company information
    year = yearOf(date)
    if((ticker,year) not in trend_dict):
        trend_dict[(ticker,year)]=[date,close,date,close]
    else:
        find close with min date and register in trend_dict
        find close with max date and register in trend_dict

for (ticker,year) in trend_dict.keys():
    calculate variation
    print('{}\t{}\t{}\t{}'.format(comp,ticker,year,variation))
```

Reducer 1: pseudocodifica

0.5.4 - Stage 2

Mapper 2 La fase di mapping dello stage 2 prevede una semplicissima lettura dei dati di input senza operare filtraggi mandando in output stream in formato 'name,ticker,close,volume,date', in modo da operare il successivo shuffle and sort sul campo 'name' riferito al nome dell'azienda

```
for line in input:
    company,ticker,year,variation = line
    print('{}\t{}\t{}\t{}'.format(company,ticker,year,variation))
```

Mapper 2: pseudocodifica

Reducer 2 Il Reducer dello stage 3 utilizza diverse strutture dati di supporto utili per le 3 principali fasi: calcolo della variazione percentuale media annua per ogni azienda, confronto con le variazioni percentuali annue delle altre aziende e raggruppamento di aziende con trend uguali

La prima dictionary ha come chiave la tupla (company,year) e come valori vengono cumulate le variazioni percentuali dei ticker associati all'azienda per calcolarne la media.

La seconda dictionary ha come chiave 'company' e come valore una lista delle variazioni percentuali medie calcolate e posizionate in ordine annuo in modo tale da poter confrontare direttamente le liste nella fase successiva per poter trovare le aziende con trend uguali.

La terza struttura dati è una lista di liste, dove il campo 1 viene popolato con una lista di aziende con trend uguali e il campo 2 contiene il relativo trend in ordine annuo

```
for line in input:

    company,ticker,year,variation = line

    # cumulate variation and count in dict
    avg_dict[(company,year)][0] += variation
    avg_dict[(company,year)][1] += 1

for (company,year) in avg_dict.keys():
    avg = round(avg_dict[(company,year)][0] / avg_dict[(company,year)][1])

    avg_list.append([company,year,avg])

for row in avg_list:

    company,year,variation = row

    if(company not in companies_dict):
        companies_dict[company] = [None, None, None]

    if (year == 2016):
        companies_dict[company][0] = variation
    elif (year == 2017):
        companies_dict[company][1] = variation
    else: # year == 2018
        companies_dict[company][2] = variation

for comp in companies_dict.keys():

    for name in companies_dict.keys():
        if(comp != name):
            are_simil_trends = compareTuples()
            if(are_simil_trends):
                register companies in list simil_trend_companies

for elem in simil_trend_companies:

    c = list of comanies
    x,y,z = variation2016,variation2017,variation2018

    print('{}:2016:{}%,2017:{}%,2018:{}%'.format(c,x,y,z))
```

Reducer 2: pseudocodifica

0.5.5 5.3 Hive

Per la realizzazione del Job3 in Hive è necessario creare due tabelle dove vengono inseriti i dati dei file in input **historical_stock_prices.csv** e **historical_stocks.csv**. Successivamente vengono create diverse Viste:

nella Vista 'joined_dataset' viene effettuato il join tra le due tabelle appena popolate filtrando solo le righe per cui la data è compresa tra il 2016 e il 2018 e mantenendo solo le colonne di interesse: 'ticker,volume,close,ticker_year,name'

le Vista 'min_date_year' e 'max_date_year' registrano la minima e la massima data disponibile per ogni year per ogni ticker, queste viste sono utili per la creazione delle successive viste 'min_close_year' e 'max_close_year' per ricavare i rispettivi close per poi ricavare la variazione annuale dei ticker nella view 'ticker_variation';

le Viste 'min_close_year' e 'max_close_year' registrano il prezzo relativo rispettivamente alla data minima e alla data massima per ogni ticker;

la Vista 'ticker_variation' calcola la variazione percentuale di ogni ticker per ogni anno;

la Vista 'company_variation' calcola la variazione percentuale media delle variazioni percentuali dei ticker per year, associandola alla relativa azienda;

la Vista 'company_trend' collezione le variazioni di ogni azienda in una lista in modo da avere i campi 'company' e 'trend', formato molto utile per la vista successiva;

la Vista 'company_trend' colleziona le variazioni di ogni azienda in una lista in modo da avere i campi 'company' e 'trend', formato molto utile per la vista successiva;

la Vista 'job3_result' sfrutta il formato della vista 'company_trend' per la ricerca delle aziende con trend uguali, basta infatti raggruppare per il campo 'trend' e collezionare le aziende associate in una lista.

```
set hive.auto.convert.join = false;

CREATE TABLE IF NOT EXISTS prices (ticker STRING,
                                   open DOUBLE,
                                   close DOUBLE,
                                   adj_close DOUBLE,
                                   low DOUBLE,
                                   high DOUBLE,
                                   volume INT,
                                   ticker_date DATE)

ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
TBLPROPERTIES("skip.header.line.count"="1");

LOAD DATA LOCAL INPATH '/home/marjo/Scrivania/UNI/magistrale/BigData/progetti/first/test/dataset/historical_stock_prices.csv'
OVERWRITE INTO TABLE prices;

CREATE TABLE IF NOT EXISTS stocks (ticker STRING,
                                   stock_exchange STRING,
                                   name STRING,
                                   sector STRING,
                                   industry STRING)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES (
    "separatorChar" = ",",
    "quoteChar" = "\"")
STORED AS TEXTFILE
TBLPROPERTIES("skip.header.line.count"="1");

LOAD DATA LOCAL INPATH '/home/marjo/Scrivania/UNI/magistrale/BigData/progetti/first/test/dataset/historical_stocks.csv'
OVERWRITE INTO TABLE stocks;


DROP VIEW IF EXISTS joined_dataset;
CREATE OR REPLACE VIEW joined_dataset AS
SELECT p.ticker AS ticker, close, YEAR(ticker_date) AS ticker_year, s.name AS company
FROM prices p JOIN stocks s ON p.ticker=s.ticker
WHERE ticker_date >= '2016-01-01' AND ticker_date <= '2018-12-31';

DROP VIEW IF EXISTS min_date_year;
CREATE OR REPLACE VIEW min_date_year AS
SELECT ticker, min(ticker_date) AS min_date, YEAR(ticker_date) AS ticker_year
FROM prices
WHERE ticker_date >= '2016-01-01' AND ticker_date <= '2018-12-31'
GROUP BY ticker, YEAR(ticker_date);

DROP VIEW IF EXISTS max_date_year;
CREATE OR REPLACE VIEW max_date_year AS
SELECT ticker, max(ticker_date) AS max_date, YEAR(ticker_date) AS ticker_year
FROM prices
WHERE ticker_date >= '2016-01-01' AND ticker_date <= '2018-12-31'
GROUP BY ticker, YEAR(ticker_date);

DROP VIEW IF EXISTS min_close_year;
CREATE OR REPLACE VIEW min_close_year AS
SELECT p.ticker AS ticker, ticker_year, close
FROM prices p JOIN min_date_year m ON p.ticker=m.ticker AND p.ticker_date=m.min_date;

DROP VIEW IF EXISTS max_close_year;
CREATE OR REPLACE VIEW max_close_year AS
SELECT p.ticker AS ticker, ticker_year, close
FROM prices p JOIN max_date_year m ON p.ticker=m.ticker AND p.ticker_date=m.max_date;

DROP VIEW IF EXISTS ticker_variation;
CREATE OR REPLACE VIEW ticker_variation AS
SELECT  mi.ticker AS ticker, mi.ticker_year as ticker_year, ROUND(((ma.close/mi.close) *100) - 100) AS variation
FROM min_close_year mi JOIN max_close_year ma ON mi.ticker=ma.ticker AND mi.ticker_year=ma.ticker_year;

DROP VIEW IF EXISTS company_variation;
CREATE OR REPLACE VIEW company_variation AS
SELECT company, j.ticker_year AS company_year, ROUND(AVG(t.variation)) AS variation
FROM joined_dataset j JOIN ticker_variation t ON j.ticker= t.ticker AND j.ticker_year=t.ticker_year
GROUP BY company, j.ticker_year
ORDER BY company, j.ticker_year;

DROP VIEW IF EXISTS company_trend;
CREATE OR REPLACE VIEW company_trend AS
SELECT company, collect_list(company_year) AS last_years, collect_list(cast(variation as string)) AS trend
FROM company_variation
GROUP BY company;

DROP VIEW IF EXISTS job3_result;
CREATE OR REPLACE VIEW job3_result AS
SELECT collect_list(company) AS companies, concat_ws(',', trend) as trends
FROM company_trend
GROUP BY trend;


INSERT OVERWRITE LOCAL DIRECTORY 'output/job3/'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
SELECT concat_ws(',', companies) AS comps, trends FROM job3_result WHERE size(companies) > 1;
```

0.5.6 5.4 Spark

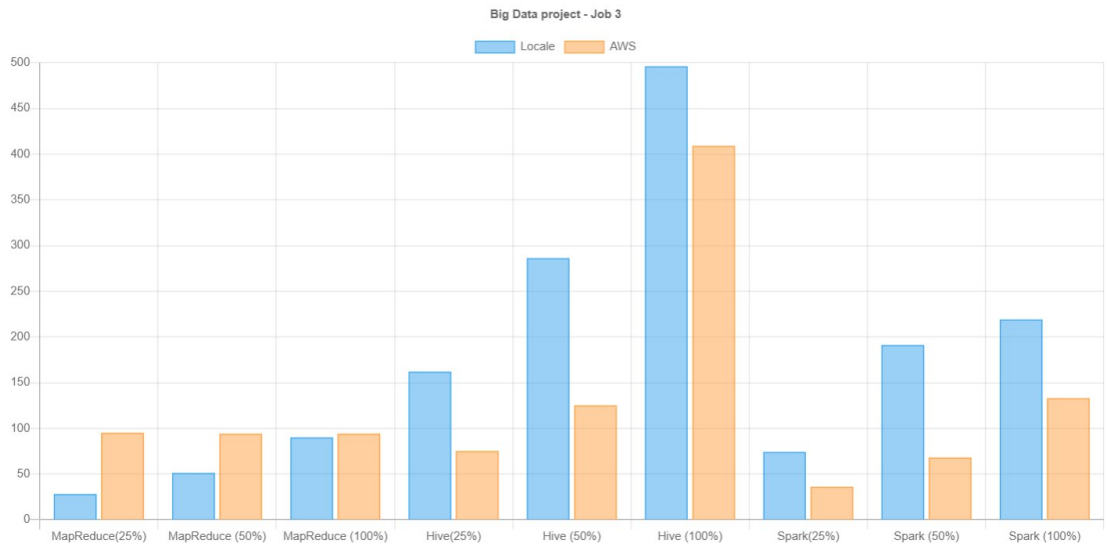
```
# Job 3

# Crea un nuovo Spark Contest per l'applicazione
sc = SparkContext(SparkConfiguration)

# Importa, formatta ed elimina la prima riga del dataset stocks
stocks = sc.textFile(stocks_path).map(line->splitLine()).filter(line != stocks.first())


# Importa, formatta ed elimina la prima riga del dataset price, elimina i record minori del 2016
prices = sc.textFile(price_path).map(line->splitLine()).filter(line !=price.first()).filter(line.data.year > '2015')

# Unisce i due rdd precedenti per creare il dataset necessario al job
# Ogni record ha come chiave l'azienda e l'anno, in modo da poter aggregare successivamente
dataset = stocks.join(prices).map(line->formatted_line)

# Rdd d'appoggio in cui vengono salvati, per ogni (azienda, anno) i relativi prezzi di chiusura ad inizio anno.
first_price = dataset.reduceByKey(extractFirsPrice())

# Rdd d'appoggio in cui vengono salvati, per ogni (azienda, anno) i relativi prezzi di chiusura a fine anno.
last_price = dataset_rdd.reduceByKey(extractLastPrice())

# Usa gli rdd precedenti per calcolare per ogni (azienda, anno) la variazione annuale
variation = last_price.join(first_price).map(line -> ((last_price - first_price) / first_price ) * 100)


# Rdd che aggrega per azienda le variazioni annuali e le trasforma per ottenre ognir record nel foramto
# ( (trend2016, trend2017, trend2018), company )
ordered_results = variation.groupByKey().map(line -> (azienda, sortYearPercentageVariationList(line.trends)))

# Forma l'output raggruppando le aziende utilizzando come chiave (trend2016, trend2017, trend2018).
output = rdd_ordered_results.groupByKey().map(line -> (trend, list[company]))

# Stampo i primi 10 valori dell'output
print(output.take(10))
```

0.5.7 5.5 Risultati

```
companies                                                trends
['ACCENTURE PLC', 'VIRTUS LIFESCI BIOTECH PRODUCTS ETF']:2016:31.0%,2017:12.0%,2018:13.0%
['ACER THERAPEUTICS INC.', 'PAN AMERICAN SILVER CORP.']:2016:-38.0%,2017:-6.0%,2018:24.0%
['ACTIVE ALTS CONTRARIAN ETF', 'PIMCO INCOME STRATEGY FUND']:2016:10.0%,2017:10.0%,2018:2.0%
['ALLIANCEBERNSTEIN HOLDING L.P.', 'CNB FINANCIAL CORPORATION']:2016:5.0%,2017:0.0%,2018:10.0%
['ALTRIA GROUP', 'FIRST TRUST SENIOR FLOATING RATE INCOME FUND II']:2016:10.0%,2017:2.0%,2018:3.0%
['ANWORTH MORTGAGE ASSET  CORPORATION', 'MEDLEY MANAGEMENT INC.']:2016:19.0%,2017:4.0%,2018:-10.0%
['APPLE INC.', 'SIX FLAGS ENTERTAINMENT CORPORATION NEW']:2016:20.0%,2017:0.0%,2018:-2.0%
['BLACKROCK CREDIT ALLOCATION INCOME TRUST', 'MFS MULTIMARKET INCOME TRUST']:2016:-4.0%,2017:4.0%,2018:-10.0%
['BLACKROCK HEALTH SCIENCES TRUST', 'QUEST DIAGNOSTICS INCORPORATED']:2016:30.0%,2017:8.0%,2018:12.0%
['BRANDYWINEGLOBAL GLOBAL INCOME OPPORTUNITIES FUND', 'NUVEEN CREDIT STRATEGIES INCOME FUND']:2016:13.0%,2017:-7.0%,2018:-5.0%
```

0.5.8 5.6 Grafici

Il grafico seguente mostra il tempo di esecuzione (l'ascissa indica il tempo in secondi) del Job3 differenziato per tecnologia utilizzata (mapReduce, Hive o Spark), grandezza del dataset in input (25%, 50% o 100%)e luogo fisico di esecuzione (macchina locale o Cluster AWS).



0.6 6. Conclusioni

La tabella seguente mostra i tempi di esecuzione (in secondi) di tutti i Job

Tecnologia	MapReduce (25%)		MapReduce (50%)		MapReduce (100%)		Hive (25%)		Hive (50%)		Hive (100%)		Spark (25%)		Spark (50%)		Spark (100%)	
Job	Locale	AWS	Locale	AWS	Locale	AWS	Locale	AWS	Locale	AWS	Locale	AWS	Locale	AWS	Locale	AWS	Locale	AWS
1	49	96	94	100	191	102	152	74	254	88	491	179	119	76	233	148	624	284
2	72	162	138	161	271	167	212	82	372	140	713	560	189	82	299	166	664	307
3	28	95	51	94	90	94	162	75	286	125	496	409	74	36	191	68	219	133

Osservando tutti i grafici e la tabella finale si evince che:

- Tutti i job eseguiti in Hive hanno un tempo di esecuzione molto più alto rispetto ai medesimi eseguiti in MapReduce o Spark.
- In MapReduce, nella fase di Mapping la scelta delle chiavi su cui verranno eseguite le operazioni di shuffle and sort è fondamentale in termini di tempo
- I vantaggi delle esecuzioni dei job su Cluster aumentano col crescere del dataset in input, per tutte le tecnologie utilizzate.