# Primo progetto Big Data 2020

Studenti: Marco Moauro (488240), Federico Roma (489348) Nome Gruppo: Moauroma

## Introduzione

Riportiamo di seguito alcune considerazioni sullo svolgimento del progetto e su come leggere la relazione.
Nella parte di *pseudocodice Spark*, abbiamo optato per mostrare per ogni operazione l'input e l'output ad esse associate, con questa notazione:

```
input → operazione(eventuali condizioni) → output
```

Nella parte *Hive* abbiamo utilizzato tabelle e non viste perché per lo svolgimento degli esercizi sono risultate più veloci.
Nei *grafici* abbiamo rappresentato la dimensione dell'input come percentuale rispetto al file originale (100% = 2gb). Per il caso 800% locale in alcuni casi i nostri pc non ce l'hanno fatta.

## JOB 1

### Pseudocodice MapReduce:

#### mapper

```
per ogni riga del file:
   filtra le righe il cui anno non è compreso nell'intervallo ['2008-01-01', '2018-12-31']
```

#### reducer

```
per ogni riga di input:
  creo la struttura TICKERS con per ogni ticker i seguenti campi:
    - prezzo di chiusura del ticker relativo alla data meno recente
    - prezzo di chiusura del ticker relativo alla data più recente
    - prezzo di chiusura minimo nell'intervallo considerato
    - prezzo di chiusura massimo nell'intervallo considerato
    - somma dei volumi del ticker nell'intervallo considerato
per ogni ticker di TICKERS calcolo:
  - la variazione della quotazione
  - volume medio
```

### Hive

```
CREATE TABLE IF NOT EXISTS tickers (ticker STRING, open FLOAT, close FLOAT, adjusted_close FLOAT,
low FLOAT, high FLOAT, volume FLOAT, data DATE)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE
LOCATION  'historical_stock_prices.csv'
TBLPROPERTIES("skip.header.line.count"="1");
LOAD DATA LOCAL INPATH 'historical_stock_prices.csv'
OVERWRITE INTO TABLE tickers;

CREATE TABLE IF NOT EXISTS tickers_grouped AS
SELECT ticker, min(data) AS start_date, max(data) AS end_date, min(close) as minor_price, max(close)
as major_price, avg(volume) as average_volume
FROM tickers
WHERE year(data) >= 2008
GROUP BY ticker;
```

```
SELECT tg.ticker, (((t2.close - t1.close) / t1.close) * 100) as percentage_variation,
tg.minor_price, tg.major_price, tg.average_volume
FROM tickers_grouped tg
JOIN tickers t1 ON tg.ticker = t1.ticker and tg.start_date = t1.data
JOIN tickers t2 ON tg.ticker = t2.ticker and tg.end_date = t2.data
ORDER BY percentage_variation DESC;
```

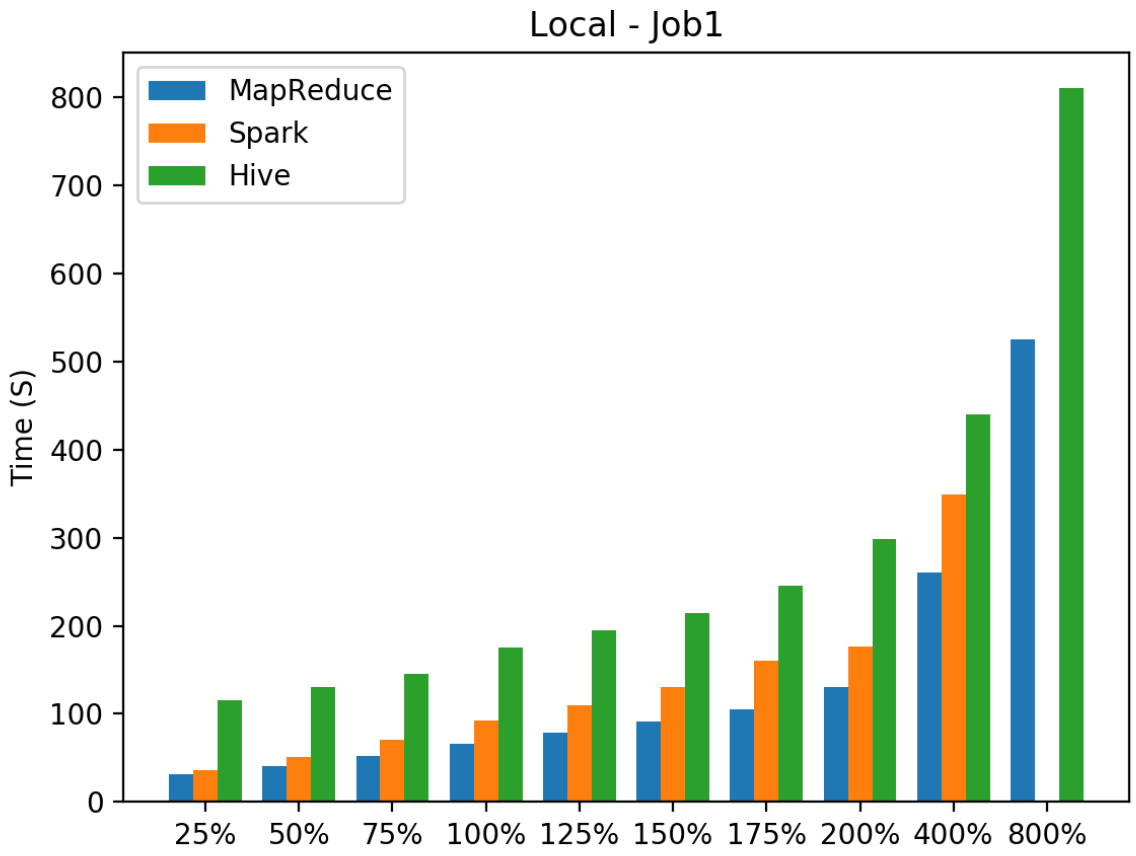
## Pseudocodice Spark

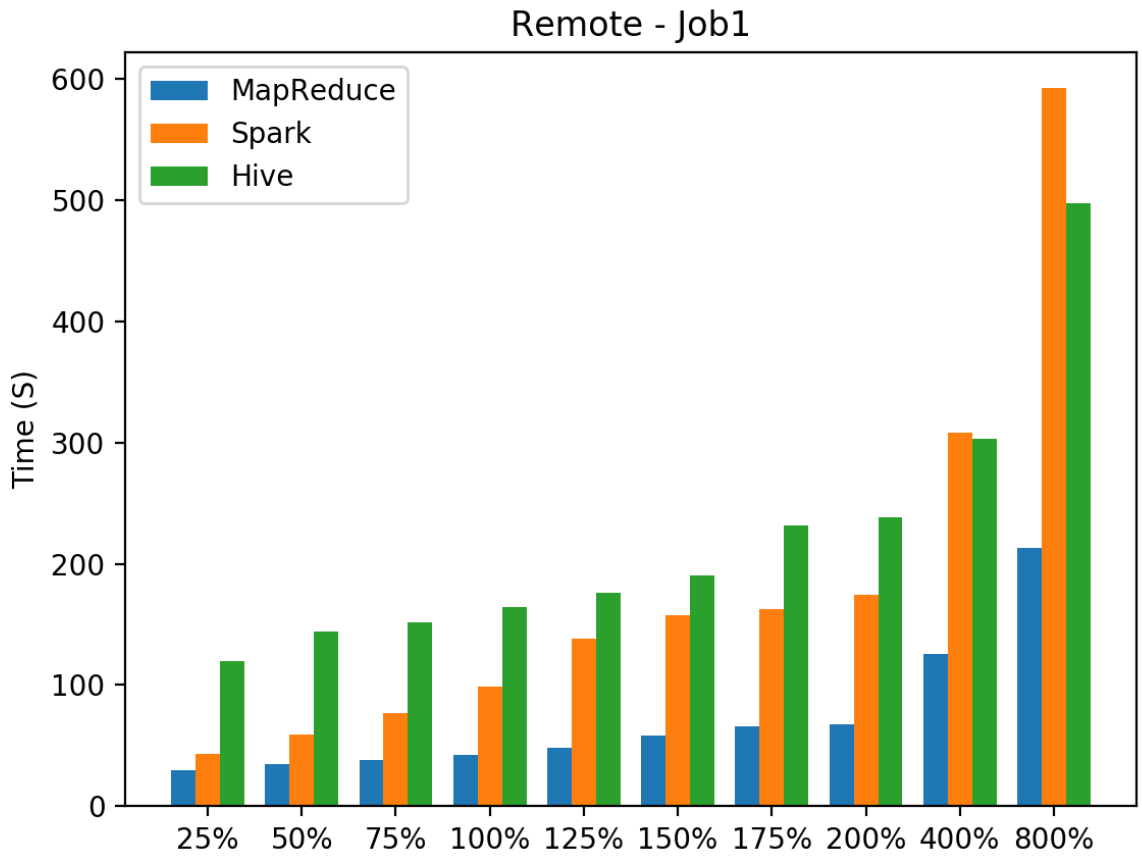
```
(ticker, open, close, adj_close, low, high, volume, date) -> FILTER(date > 2008) ->
-> (ticker, open, close, adj_close, low, high, volume, date) -> MAP ->
-> (ticker, ((close, date), (close, date), close, close, (volume, 1))) -> REDUCEBYKEY ->
-> (ticker, ("close-for-min-date", "min-date"), ("close-for-max-date", "max-date"), min-close, max-
close, (volumes_sum, count))) -> MAP ->
-> (ticker, "variation-of-quotation", min-close, max-close, avg-volume) -> SORT(ticker) ->
-> (ticker, "variation-of-quotation", min-close, max-close, avg-volume)
```

## TOP 10 risultati:

| ticker | variazione | prezzo minimo | prezzo massimo | volume |
|---|---|---|---|---|
| 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 |

## Tabella e grafici






| | 25% | 50% | 75% | 100% | 125% | 150% | 175% | 200% | 400% | 800% |
|---|---|---|---|---|---|---|---|---|---|---|
| **MR Locale** | 31 | 41 | 52 | 66 | 79 | 91 | 105 | 130 | 260 | 525 |
| **MR AWS** | 30 | 35 | 38 | 42 | 48 | 58 | 66 | 68 | 126 | 213 |
| **Hive Locale** | 115 | 130 | 145 | 175 | 195 | 215 | 245 | 299 | 440 | 810 |
| **Hive AWS** | 120 | 144 | 152 | 164 | 176 | 190 | 232 | 238 | 303 | 497 |
| **Spark Locale** | 36 | 51 | 71 | 93 | 110 | 130 | 160 | 176 | 349 | - |
| **Spark AWS** | 43 | 59 | 77 | 99 | 138 | 158 | 163 | 174 | 308 | 592 |

# JOB 2

## Pseudocodice MapReduce:

### mapper

```
per ogni riga del file:
   filtra le righe il cui anno non è compreso nell'intervallo ['2008-01-01', '2018-12-31']
```

### reducer

```
- leggendo il file historical_stocks.csv da hdfs creiamo la struttura STOCKS che ci consente di
recuperare il settore di un ticker.
- per ogni riga di input:
  creo la struttura SECTORS che per ogni <settore, anno> ha le seguenti informazioni:
    - volume complessivo
    - struttura TICKERS con i seguenti campi:
        - prezzo di chiusura del ticker relativo alla data meno recente
        - prezzo di chiusura del ticker relativo alla data più recente
        - prezzo di chiusura medio
```

```
- per ogni coppia <settore, anno> di SECTORS calcolo:
  - volume annuale medio
  - la variazione annuale media delle aziende del settore
  - quotazione giornaliera media
  - volume medio
```

## Hive

```
CREATE TABLE IF NOT EXISTS stock_prices (ticker STRING, open FLOAT, close FLOAT,adjustedThe
FLOAT,low FLOAT,high FLOAT,volume FLOAT,tickerdate DATE)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE
LOCATION  'historical_stock_prices.csv';

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

CREATE TABLE IF NOT EXISTS stocks (ticker STRING, exc STRING, name STRING, sector STRING, industry
STRING)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES (
   "separatorChar" = ",",
   "quoteChar"     = "\"",
   "skip.header.line.count"="1")
   STORED AS TEXTFILE
LOCATION  'historical_stocks.csv';
LOAD DATA LOCAL INPATH 'historical_stocks.csv'
OVERWRITE INTO TABLE stocks;


CREATE TABLE IF NOT EXISTS minmaxdate AS
SELECT s.sector,
       YEAR(sp.tickerdate) as tickeryear,
       s.name,
       sp.ticker,
       MIN(tickerdate) AS mindate,
       MAX(tickerdate) AS maxdate
FROM stock_prices sp
JOIN stocks s ON sp.ticker = s.ticker
WHERE tickerdate >= '2008-01-01'
GROUP BY s.sector, YEAR(sp.tickerdate), s.name, sp.ticker;


CREATE TABLE IF NOT EXISTS minmaxclose AS
SELECT mm.sector,
       mm.tickeryear,
       mm.name,
       AVG((((sp2.close - sp1.close) / sp1.close) * 100)) AS percentile
FROM minmaxdate mm
JOIN stock_prices sp1 ON mm.ticker = sp1.ticker AND mindate = sp1.tickerdate
JOIN stock_prices sp2 ON mm.ticker = sp2.ticker AND maxdate = sp2.tickerdate
GROUP BY mm.sector, mm.tickeryear, mm.name;


CREATE TABLE IF NOT EXISTS statsperticker AS
SELECT stocks.sector AS sector,
       YEAR(stock_prices.tickerdate) AS tickeryear,
       stocks.name,
       stock_prices.ticker AS ticker,
       SUM(volume) AS sumvolume,
       AVG(close) AS avgclose
FROM stock_prices
JOIN stocks ON stock_prices.ticker = stocks.ticker
WHERE tickerdate > '2008-01-01'
GROUP BY stocks.sector, YEAR(stock_prices.tickerdate), stocks.name, stock_prices.ticker;
```

```
CREATE TABLE IF NOT EXISTS statspername AS
SELECT sector,
       tickeryear,
       name,
       AVG(sumvolume) AS avgvolume,
       AVG(avgclose) AS avgclose
FROM statsperticker
GROUP BY sector, tickeryear, name;

CREATE TABLE IF NOT EXISTS results AS
SELECT m.sector,
       m.tickeryear,
       AVG(s.avgvolume),
       AVG(m.percentile),
       AVG(s.avgclose)
FROM minmaxclose m
JOIN statspername s on m.name = s.name and m.tickeryear = s.tickeryear
GROUP BY m.sector, m.tickeryear;
```

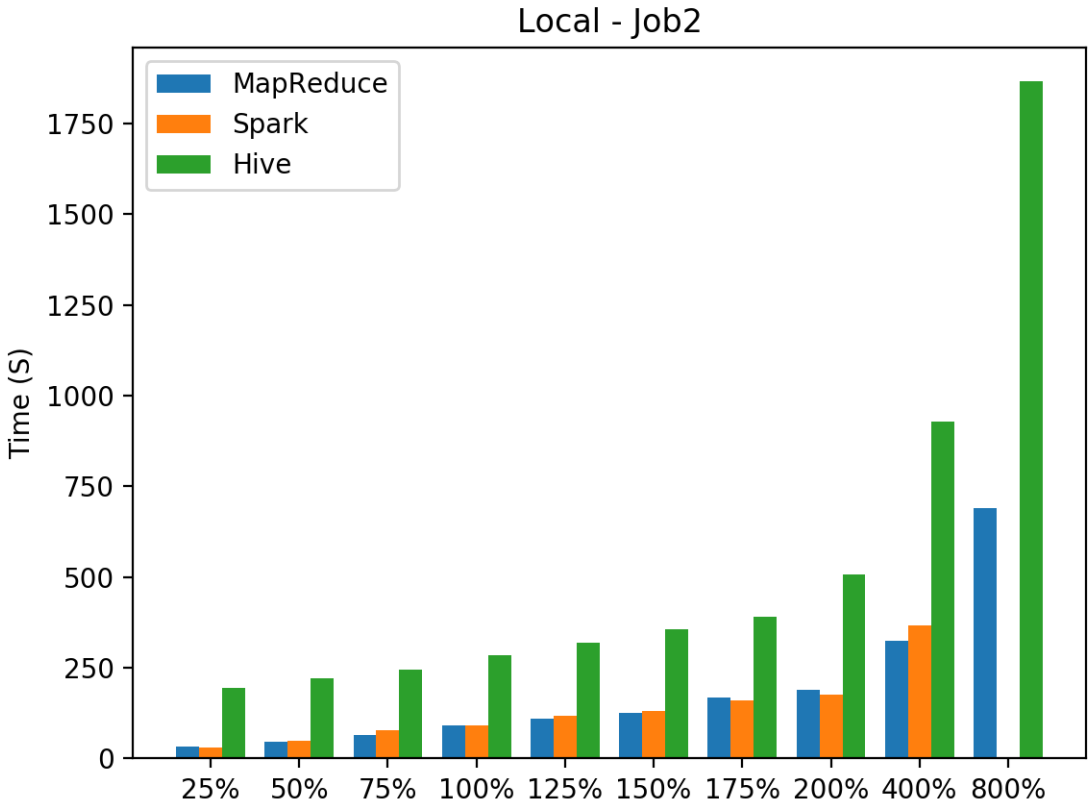
## Pseudocodice Spark

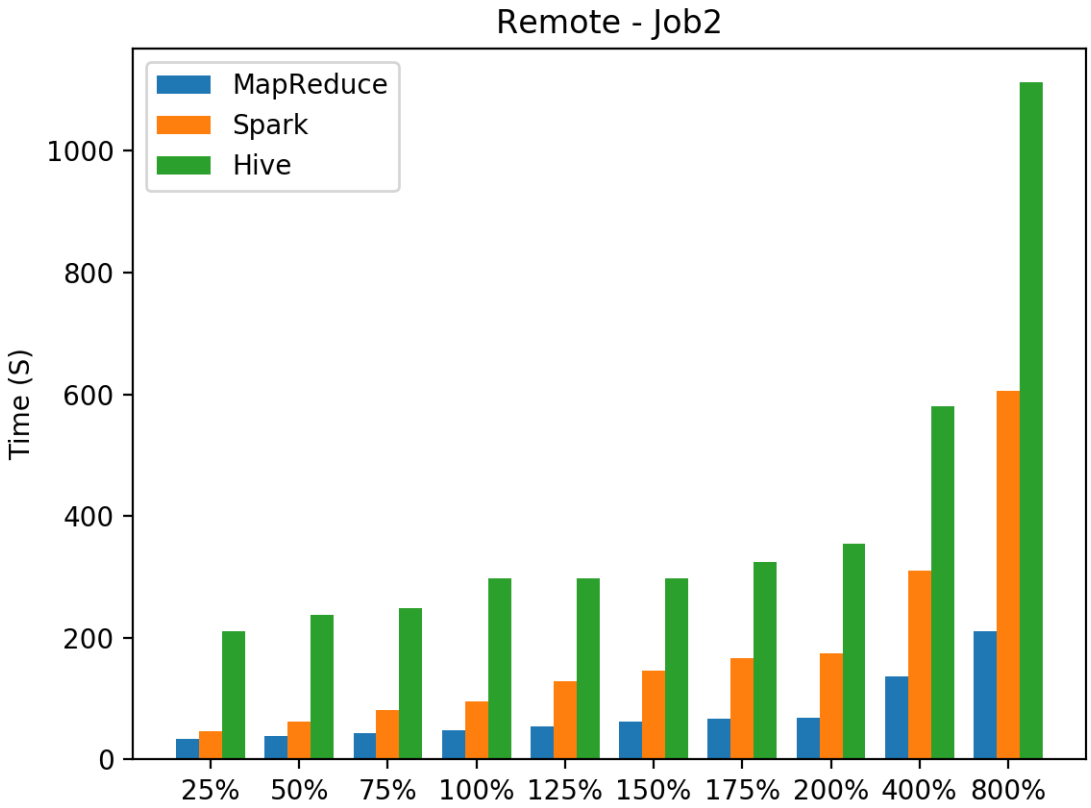
```
# STOCKS lo uso per recuperare il settore dal ticker
STOCKS = (ticker, exchange, name, sector, industry) -> MAP ->
-> (ticker, sector) -> COLLECTASMAP -> dizionario <ticker, sector>

(ticker, open, close, adj_close, low, high, volume, date) -> FILTER(date > 2008) ->
-> (ticker, open, close, adj_close, low, high, volume, date) -> MAP ->
-> ("ticker-year", (volume, (close, date), (close, date), close, 1)) -> REDUCEBYKEY ->
-> (ticker, (volume-sum, ("close-for-min-date", "min-date"), ("close-for-max-date", "max-date"),
close-sum, count))) -> MAP ->
-> ("name-year", (volume-sum, "variation-of-quotation", daily-quotation, 1)) -> REDUCEBYKEY ->
-> ("name-year", ("volume-sum-by-name", "variation-of-quotation-sum", daily-quotation-sum, count))
-> MAP ->
-> ("name-year", "avg-volume", "avg-variation-of-quotation", "avg-daily-quotation") -> SORT(name-
year) ->
-> ("name-year", "avg-volume", "avg-variation-of-quotation", "avg-daily-quotation")
```

## TOP 10 risultati:

| settore | vol. ann. medio | var. ann. media | quotazione giornaliera media |
|---|---|---|---|
| BASIC INDUSTRIES-2008 | 560950485.8010752 | -42.05% | 604.1003484428694 |
| BASIC INDUSTRIES-2009 | 589384165.1354166 | 82.32% | 224.26078211714685 |
| BASIC INDUSTRIES-2010 | 476571424.22772276 | 32.89% | 32.12626054734056 |
| BASIC INDUSTRIES-2011 | 448450099.39903843 | -11.34% | 443.94494439403115 |
| BASIC INDUSTRIES-2012 | 377483105.2511848 | 8.19% | 124.53215994029918 |
| BASIC INDUSTRIES-2013 | 367271657.58371043 | 18.85% | 238.5312945435339 |
| BASIC INDUSTRIES-2014 | 350472233.61538464 | -3.35% | 233.4369196485182 |
| BASIC INDUSTRIES-2015 | 388588042.26829267 | 123.46% | 41.4219678641751 |
| BASIC INDUSTRIES-2016 | 470968318.09411764 | 47.54% | 30.896128472415562 |
| BASIC INDUSTRIES-2017 | 383340553.0 | 16.69% | 35.15950676606339 |

**Tabella e grafici**



| | 25% | 50% | 75% | 100% | 125% | 150% | 175% | 200% | 400% | 800% |
|---|---|---|---|---|---|---|---|---|---|---|
| **MR Locale** | 33 | 46 | 65 | 92 | 110 | 126 | 168 | 190 | 325 | 690 |
| **MR AWS** | 34 | 38 | 44 | 48 | 54 | 63 | 67 | 69 | 137 | 210 |
| **Hive Locale** | 195 | 220 | 245 | 285 | 320 | 355 | 390 | 507 | 928 | 1865 |
| **Hive AWS** | 210 | 238 | 249 | 297 | 297 | 297 | 325 | 355 | 581 | 1112 |
| **Spark Locale** | 29 | 49 | 79 | 90 | 117 | 132 | 160 | 177 | 366 | - |
| **Spark AWS** | 47 | 72 | 81 | 95 | 129 | 146 | 166 | 174 | 311 | 605 |

# JOB 3

## Pseudocodice MapReduce:

### mapper

```
per ogni riga del file:
   filtra le righe il cui anno non è compreso nell'intervallo ['2016-01-01', '2018-12-31']
```

### reducer

```
- leggendo il file historical_stocks.csv da hdfs creiamo la struttura STOCKS che ci consente di
recuperare il nome dell'azienda di un ticker.
- per ogni riga di input:
  - creo la struttura TICKERS che per ogni ticker memorizza delle struttere per gli anni 2016, 2017
    e 2018 composte in questo modo:
    - prezzo di chiusura del ticker relativo alla data meno recente.
    - prezzo di chiusura del ticker relativo alla data più recente.
- per ogni ticker in TICKERS calcolo:
  - variazione della quotazione del 2016.
  - variazione della quotazione del 2017.
  - variazione della quotazione del 2018.
- raggruppo i ticker tramite il nome dell'azienda calcolando la variazione di quotazione media.
- raggruppo le aziende in base all'incremento degli ultimi tre anni ottenendo liste di aziende
simili.
```

## Hive

```
CREATE TABLE IF NOT EXISTS stock_prices
(
    ticker      STRING,
    open        FLOAT,
    close       FLOAT,
    adjustedThe FLOAT,
    low         FLOAT,
    high        FLOAT,
    volume      FLOAT,
    tickerdate  DATE
)
    ROW FORMAT DELIMITED
        FIELDS TERMINATED BY ','
    STORED AS TEXTFILE
    LOCATION 'historical_stock_prices.csv';

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

```
CREATE TABLE IF NOT EXISTS stocks
(
    ticker   STRING,
    exc      STRING,
    name     STRING,
    sector   STRING,
    industry STRING
)
    ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
        WITH SERDEPROPERTIES (
        "separatorChar" = ",",
        "quoteChar" = "\"",
        "skip.header.line.count" = "1")
    STORED AS TEXTFILE
    LOCATION 'historical_stocks.csv';
LOAD DATA LOCAL INPATH 'historical_stocks.csv'
    OVERWRITE INTO TABLE stocks;


CREATE TABLE IF NOT EXISTS minmaxdates AS
SELECT YEAR(tickerdate) AS tickeryear, ticker, MIN(tickerdate) AS mindate, MAX(tickerdate) AS
maxdate
FROM stock_prices
WHERE stock_prices.tickerdate >= '2016-01-01'
GROUP BY YEAR(tickerdate), ticker;

CREATE TABLE IF NOT EXISTS minmaxdates_filtered AS
SELECT m.ticker, m.tickeryear, m.mindate, m.maxdate
FROM minmaxdates m
JOIN
     (
         SELECT ticker, count(*) as count
         FROM minmaxdates
         GROUP BY ticker
     ) mc
     ON m.ticker = mc.ticker
WHERE mc.count = 3;

CREATE TABLE IF NOT EXISTS minmaxclose AS
SELECT m.tickeryear,
       m.ticker,
       (((sp2.close - sp1.close) / sp1.close) * 100) AS percentile
FROM minmaxdates_filtered m
         JOIN stock_prices sp1 on m.ticker = sp1.ticker and m.mindate = sp1.tickerdate
         JOIN stock_prices sp2 on m.ticker = sp2.ticker and m.maxdate = sp2.tickerdate;

CREATE TABLE IF NOT EXISTS percentiles_by_company AS
SELECT s.name, m.tickeryear, ROUND(AVG(m.percentile)) AS percentile
FROM minmaxclose m
JOIN stocks s on m.ticker = s.ticker
GROUP BY s.name, m.tickeryear
ORDER BY s.name desc, m.tickeryear asc;

CREATE TABLE IF NOT EXISTS triplets AS
SELECT name, collect_list(percentile) AS triplet
FROM percentiles_by_company
GROUP BY name;

CREATE TABLE IF NOT EXISTS similars AS
SELECT triplet, collect_list(name) AS companies
FROM triplets
WHERE size(triplet) = 3
GROUP BY triplet;

select companies, triplet
from similars
WHERE size(companies) > 1
ORDER BY triplet asc;
```

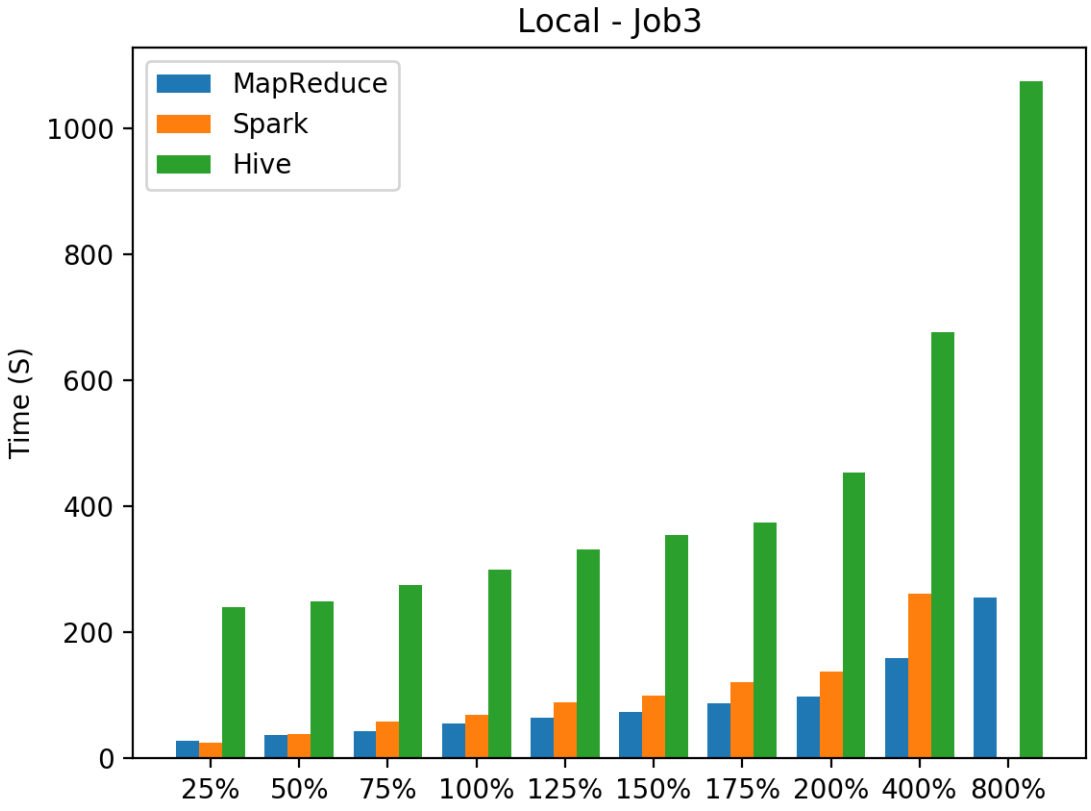
## Pseudocodice Spark

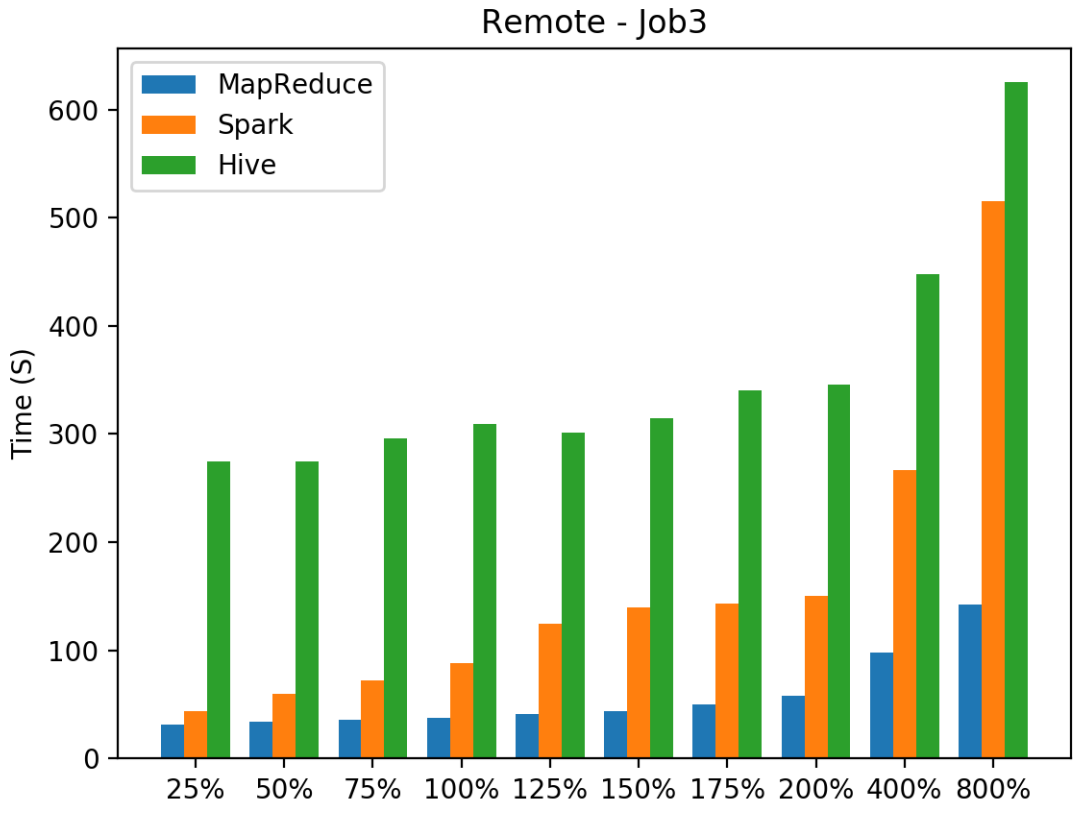
```
# STOCKS lo uso per recuperare il nome dal ticker
STOCKS = (ticker, exchange, name, sector, industry) -> MAP ->
-> (ticker, name) -> COLLECTASMAP -> dizionario <ticker, name>

(ticker, open, close, adj_close, low, high, volume, date) -> FILTER(date > 2016) ->
-> (ticker, open, close, adj_close, low, high, volume, date) -> MAP ->
-> ("ticker-year", (close, date), (close, date), date) -> REDUCEBYKEY ->
-> ("ticker-year", ("close-for-min-date", "min-date"), ("close-for-max-date", "max-date")) -> MAP ->
-> ("name-year", "variation-of-quotation", 1) -> REDUCEBYKEY ->
-> ("name-year", "variation-of-quotation-sum", count) -> MAP ->
-> (name, (year, "variation-of-quotation-avg")) -> REDUCEBYKEY ->
-> (name, years_variations) -> FILTER(years_variations.length == 3) ->
-> (name, years_variations) -> MAP ->
-> ("variation2016_variation2017_variation2018", name) -> REDUCEBYKEY ->
-> ("variation2016_variation2017_variation2018", names) ->
SORT("variation2016_variation2017_variation2018") ->
-> ("variation2016_variation2017_variation2018", names)
```

## TOP 10 risultati:

```
['STURM, RUGER & COMPANY, INC.', 'LINCOLN EDUCATIONAL SERVICES CORPORATION']:
    2016: -14.0%, 2017: 4.0%, 2018: 8.0%
['BLACKROCK MUNIHOLDINGS FUND, INC.', 'BLACKROCK MUNICIPAL INCOME QUALITY TRUST']:
    2016: -6.0%, 2017: 2.0%, 2018: -7.0%
['MFS GOVERNMENT MARKETS INCOME TRUST', 'CHINA MOBILE (HONG KONG) LTD.']:
    2016: -5.0%, 2017: -4.0%, 2018: -8.0%
['NUVEEN PENNSYLVANIA QUALITY MUNICIPAL INCOME FUND', 'NUVEEN SELECT MATURITIES MUNICIPAL FUND']:
    2016: -5.0%, 2017: 1.0%, 2018: -3.0%
['INVESCO MUNICIPAL TRUST', 'BLACKROCK MUNIYIELD FUND, INC.']:
    2016: -5.0%, 2017: 2.0%, 2018: -5.0%
['FIRST TRUST ALTERNATIVE ABSOLUTE RETURN STRATEGY ETF', 'MFS MUNICIPAL INCOME TRUST']:
    2016: -4.0%, 2017: 3.0%, 2018: -2.0%
['PIMCO NEW YORK MUNICIPAL INCOME FUND II', 'NUVEEN MICHIGAN QUALITY MUNICIPAL INCOME FUND']:
    2016: -3.0%, 2017: -1.0%, 2018: -5.0%
['PIMCO STRATEGIC INCOME FUND, INC.', 'TRIPLEPOINT VENTURE GROWTH BDC CORP.']:
    2016: -3.0%, 2017: 4.0%, 2018: 4.0%
['THE GDL FUND', 'BLACKROCK INCOME TRUST INC. (THE)']:
    2016: -1.0%, 2017: -2.0%, 2018: -7.0%
['EATON VANCE HIGH INCOME 2021 TARGET TERM TRUST', 'VANGUARD MORTGAGE-BACKED SECURITIES ETF',
'VANGUARD INTERMEDIATE-TERM TREASURY ETF', 'ISHARES MBS ETF', 'ISHARES GNMA BOND ETF']:
    2016: -1.0%, 2017: 0.0%, 2018: -2.0%
```

**Tabella e grafici**



| | 25% | 50% | 75% | 100% | 125% | 150% | 175% | 200% | 400% | 800% |
|---|---|---|---|---|---|---|---|---|---|---|
| **MR Locale** | 28 | 37 | 43 | 56 | 65 | 74 | 87 | 98 | 159 | 255 |
| **MR AWS** | 31 | 34 | 36 | 38 | 41 | 44 | 50 | 58 | 98 | 142 |
| **Hive Locale** | 240 | 250 | 275 | 300 | 332 | 355 | 375 | 454 | 676 | 1074 |
| **Hive AWS** | 275 | 275 | 296 | 309 | 310 | 315 | 340 | 346 | 448 | 625 |
| **Spark Locale** | 25 | 39 | 58 | 69 | 89 | 100 | 121 | 138 | 261 | - |
| **Spark AWS** | 44 | 60 | 72 | 88 | 125 | 140 | 143 | 150 | 267 | 515 |